香港城市大學（東莞）
City University of Hong Kong
(Dongguan)

Celebrating the Establishment in Year 2024

Lecture 12: Course Review

CS6493 Natural Language Processing

Instructor: Linqi Song

Outline

- Course review
- LOQ
- Final exam arrangements
- Q&A

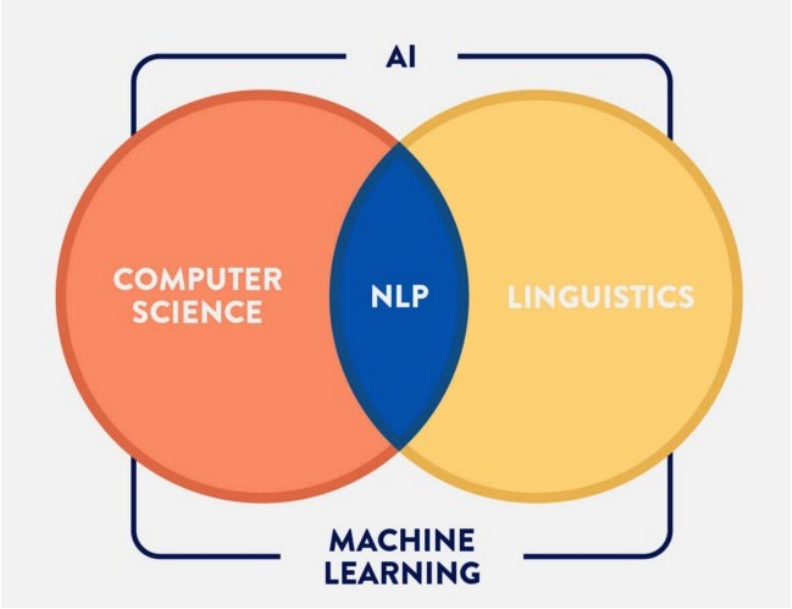
Course review

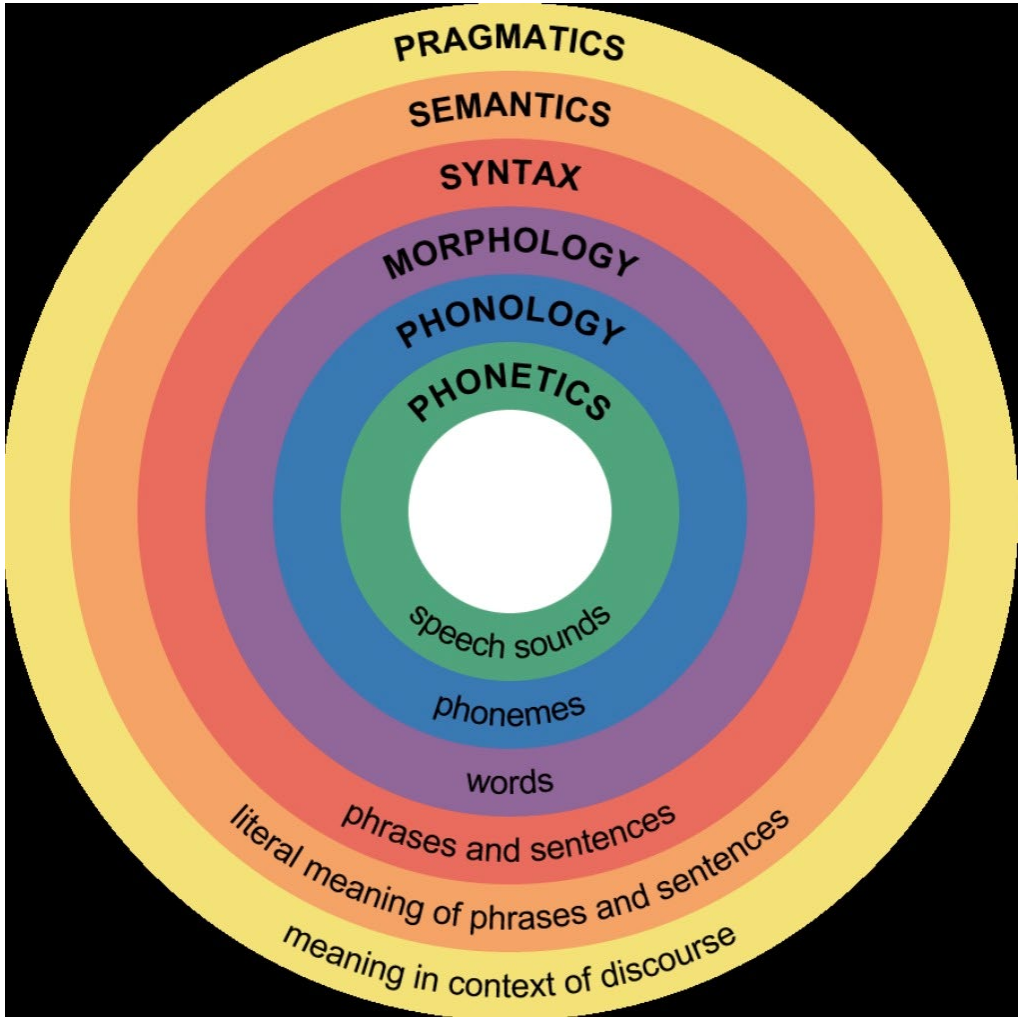
L1 Introduction

- 0. Course logistics
- 1. Recent applications
- 2. History of NLP
- 3. Challenges
- 4. How NLP works?

What is Natural Language Processing?

- A branch of Artificial Intelligence.
- Make computers to learn, process and manipulate natural languages to interact with humans.





Bird’s-eye view of this course

- Basics: Linguistics, language models, word embeddings
- Tasks: NLU, NLG, machine translation, question answering, dialogue, text classification
- Large language models: Transformers, pretraining (e.g., BERT, GPT), prompting and alignment, LLM agent, efficient finetuning, RAG

Preprocessing of data

- Tokenization
  - Tokenization is the process of breaking up text document into individual words called tokens.
  - Tokens can be either words, characters, or sub-words (oov, BPE).
- Stop words removal
- Stemming
- Lemmatization
- Vectorization (N-gram, BOW, TF-IDF)

L2 Language models

- 1. Language model definition & applications
- 2. Model construction
  - Statistical language models
  - Neural language models
- 3. Language model evaluation

What is a language model?

- A language model is a probability distribution over sequences of words. Given such a sequence, say of length *T*, it assigns a probability $P(x^{(1)}, x^{(2)}, \dots x^{(T)})$ to the whole sequence.

- To determine whether a given ordering of words sounds like natural language

- The famous infinite monkey theorem
  - A monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, such as the complete works of William Shakespeare.
  - How can we judge the performance of each typing?

    E.g., two orderings

    'NLP is about how machines understand and process natural languages.'

    'machines NLP about languages understand how and is natural process.'

Another interpretation of a language model

- The task of language model could be regarded as assigning probability to a piece of text.
- For example, consider a piece of text $(x^{(1)}, x^{(2)}, \ldots x^{(T)})$, then according to the Language Model, the probability of the text could

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

A language model can also be interpreted as predicting what word comes next

n-gram model

- Every word in Σ is assigned some probability, conditoned on a fixed-length history (n – 1).
- An n-gram is a chunk of *n* consecutive words.

the students opened their _____

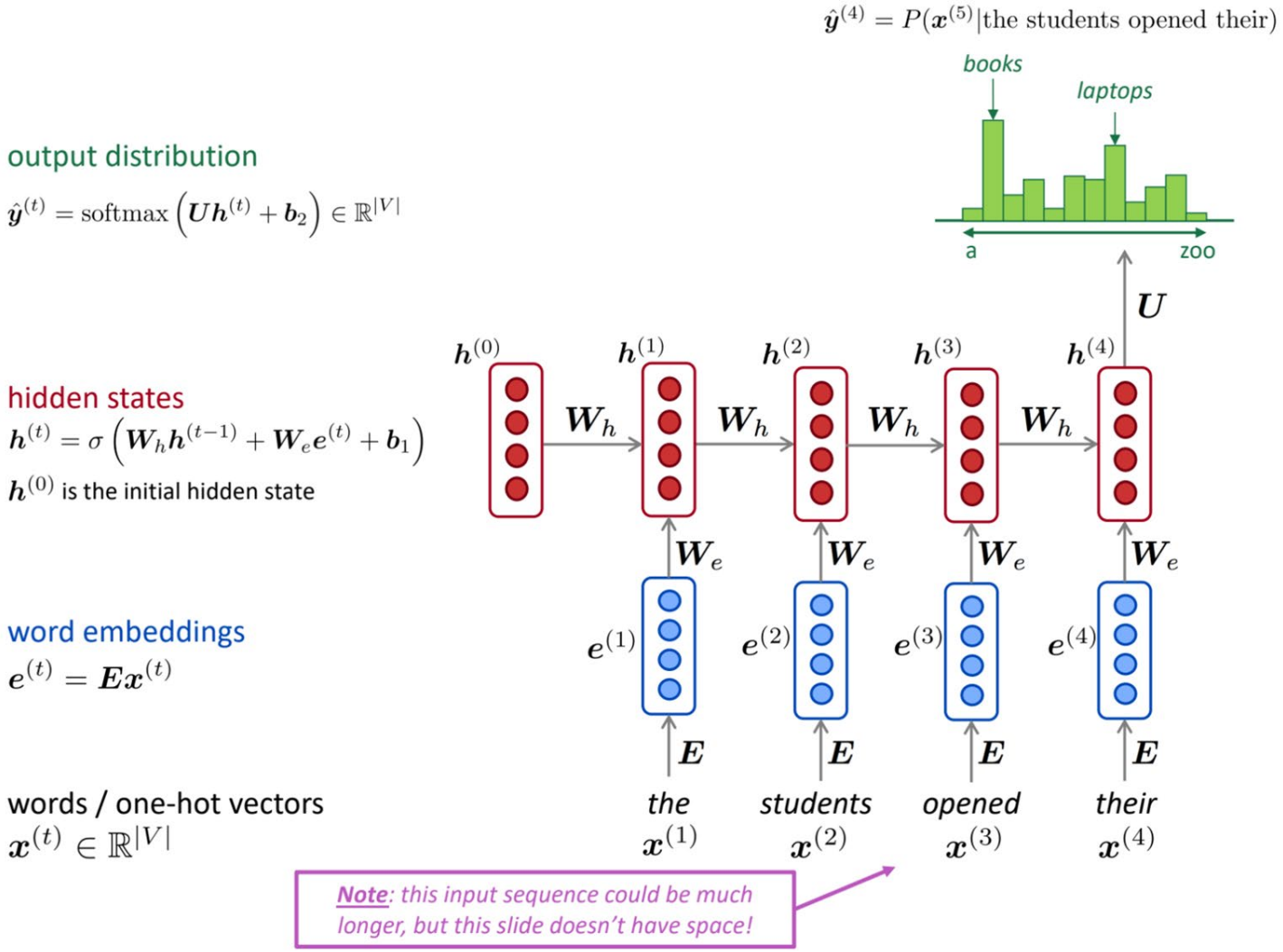
unigrams: “the”, “students”, “opened”, ”their”

bigrams: “the students”, “students opened”, “opened their”

trigrams: “the students opened”, “students opened their”

4-grams: “the students opened their”

RNN-based language model

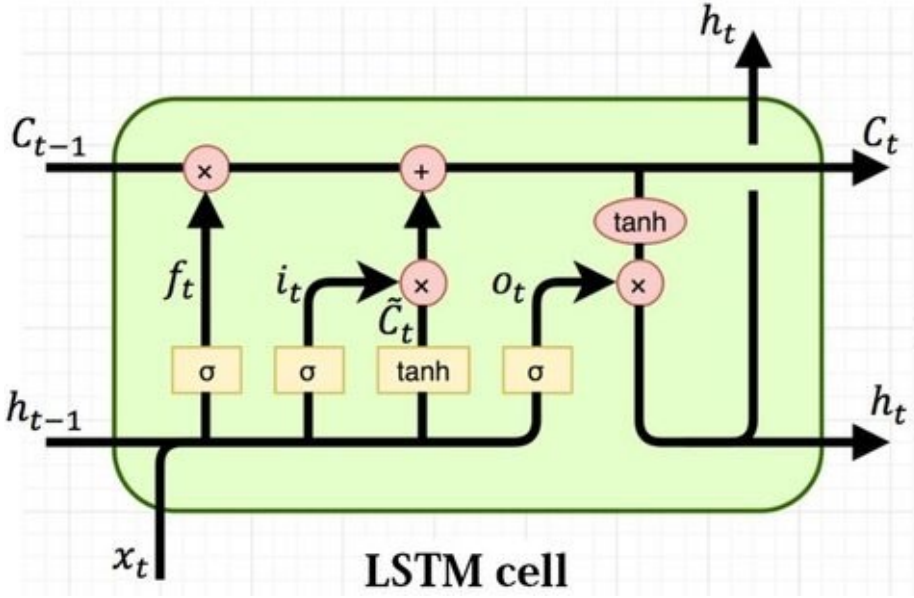


RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

LSTM-based LM

- Long Short-Term Memory, a special RNN to slove vanish gradient phenomenon (Hochreiter and Schmidhuber, 1997).
- Instead of adding up the history, it uses a set of gating units to



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$: input gate

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$: forget gate

$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$: output gate

$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$: new content of input t

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$: cell state

$$h_t = \tanh(C_t) * o_t$$: hidden state

Evaluating language models

- The standard evaluation metrics for LM is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- Perplexity can be derived directly from the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

- Lower perplexity is better.

L3 Word embedding

- 1. Word embedding definition and principles
- 2. Embedding methods – word2vec
  - Continuous bag-of-words
  - Skip-gram
- 3. Improve training efficiency
  - Negative sampling
  - Hierarchical softmax
- 4. Other word embedding methods
  - GloVe
- 5. Contextualized word embeddings (ELMo)

One-hot vector: discrete symbols

- A localist representation
- One-hot vectors
  - one 1, the rest 0s

hotel = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
motel = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 ,0 ,0 ,0, 0, 0, 0, 0]

- Vector diemension = number of words in vocabulary

Distributional hypothesis

- When a word *w* appears in a text, its context is the set of words that appear nearby (with a fixed-size window)
- Use the many contexts of *w* to build up a representation of *w*
- In the example, the context words will represent ***banking***

Example: *context words represent banking*

... government debt problems turning into **banking** crises as happened in 2009…

...saying that Europe needs unified **banking** regulation to replace the hodgepodge…

...India has just given its **banking** system a shot in the arm...

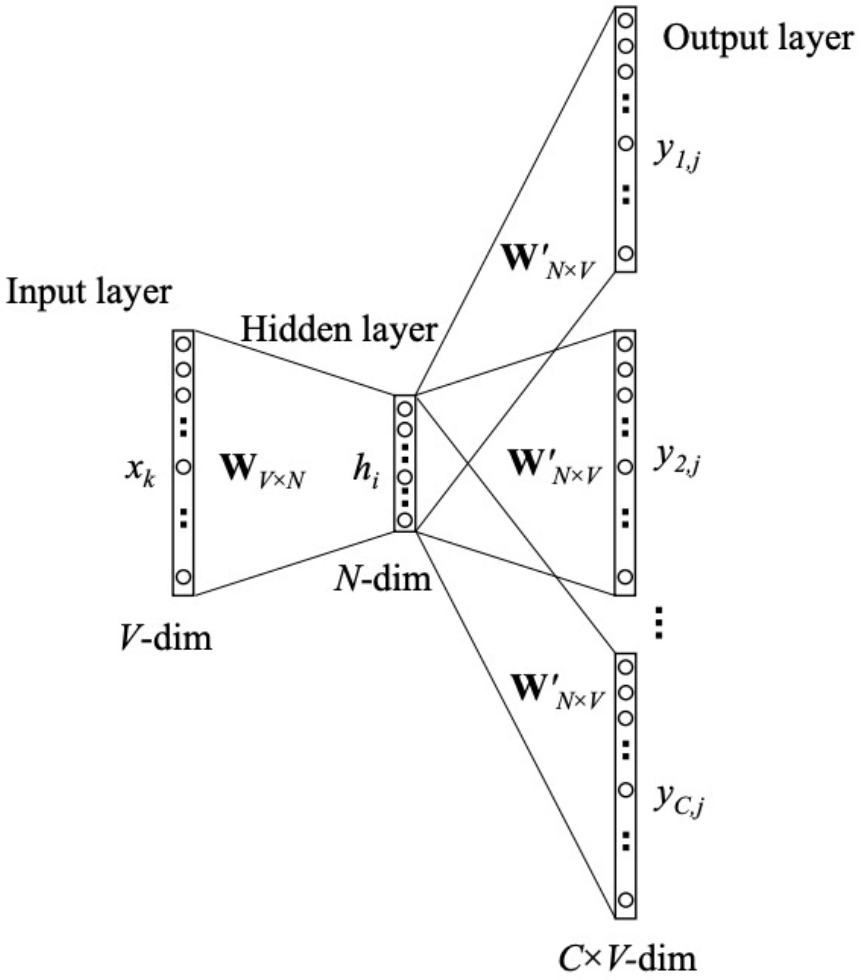
Word2vec

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

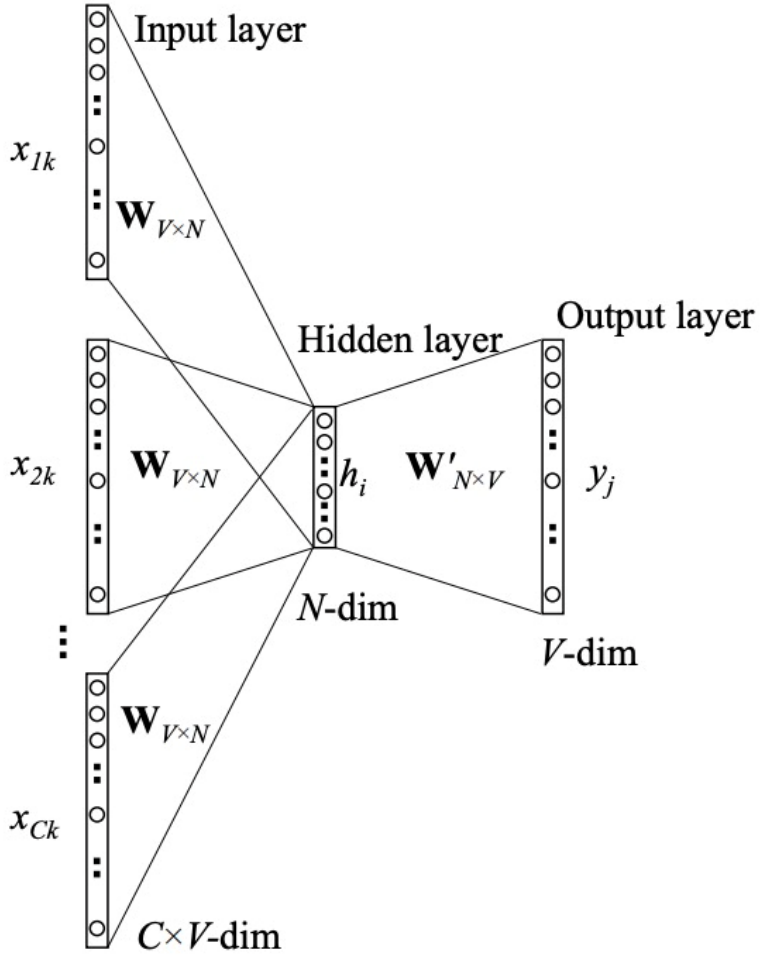
Idea:

- **Input**: Given a large corpus of text (e.g., a bunch of sentences or documents)
- **Output**: Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a target word (or center word) w_t and several context words w_c
- Use the similarity of the word vectors for w_t and w_c
  - Skip-gram: to calculate the probability of *context words* w_c given *the target word* w_t
  - Continuous bag of words (CBOW): to calculate the probability of target word w_t given context words w_c
- Keep adjusting the word vectors to maximize the probability

Skip-gram vs CBOW



Skip-gram network structure



CBOW model

Skip-gram's optimization problem

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Improve the efficiency in training

Reason:

- The size of vocabulary V is impressively large
- Evaluation of the objective function would take $O(V)$ time

Solution:

- Negative sampling
- Hierarchical softmax

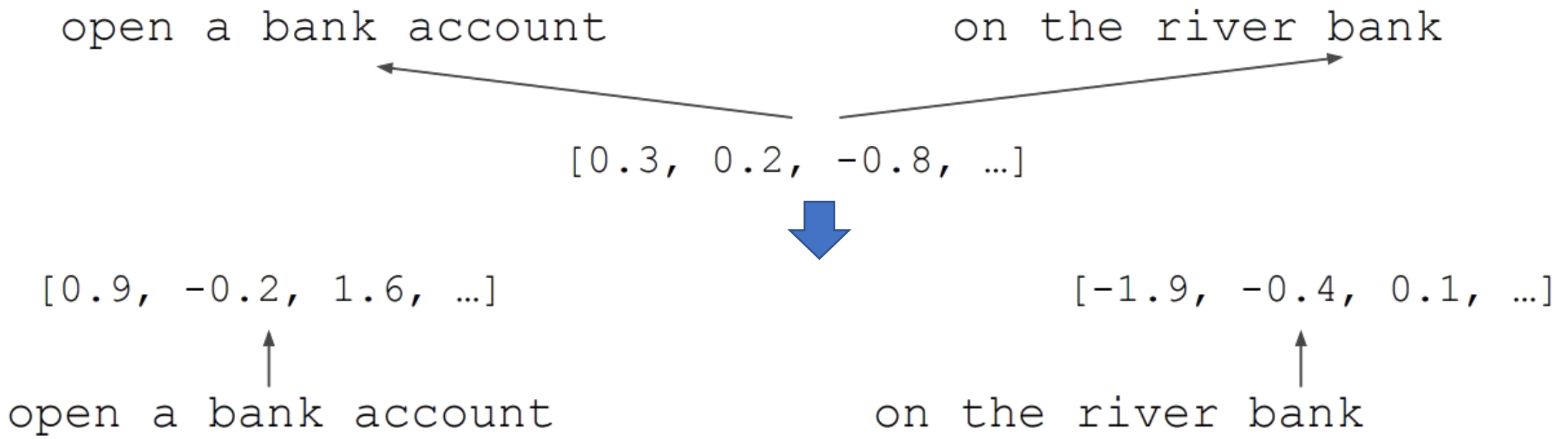
Comparison:

- Hierarchical softmax tends to be better for infrequent words.
- Negative sampling works better for frequent words and lower dimensional vectors.

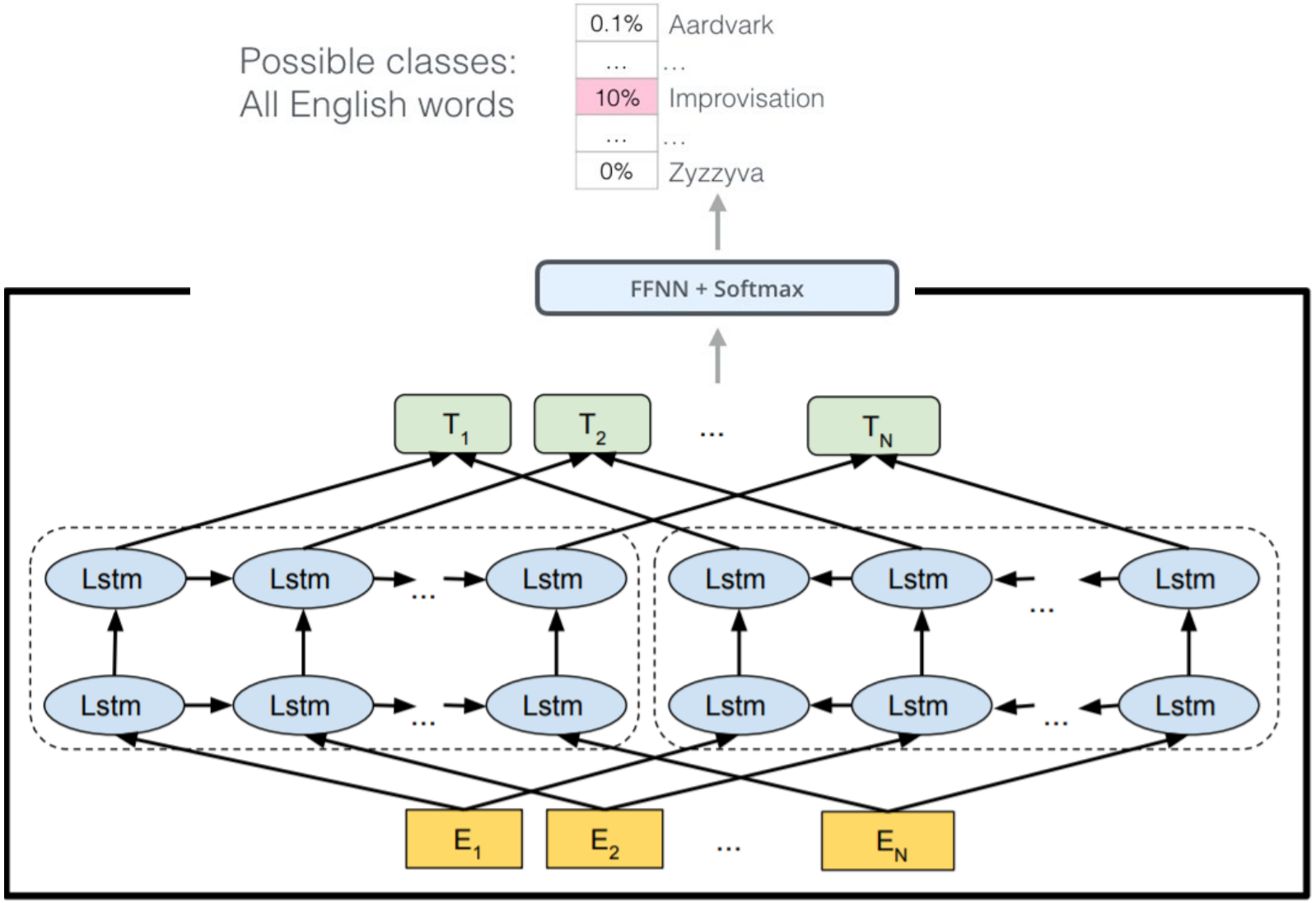
Contextualized embeddings

ELMo: Deep contextualized word representations, AI2 & Univ. Washington, 2018

From context independent embeddings to context dependent embeddings



ELMo uses a bi-directional LSTM to pre-train the language model



L4 Transformers and pretraining-finetuning

- 1. Attention
- 2. Transformer
- 3. BERT
- 4. GPT

Attention: formal description

- Given a query vector $\boldsymbol{q}$, and a set of key-value pairs (all vectors) $\{(\boldsymbol{k}_i, \boldsymbol{v}_i)\}_{i=1}^{L}$, we first calculate the similarity/attention score between the query and each key:

$$s_i = similarity(\boldsymbol{q}, \boldsymbol{k}_i).$$

- Normalize the similarity score to be between 0 and 1, and they sum up to 1. These are called attention distribution. One way is to use the softmax operation.

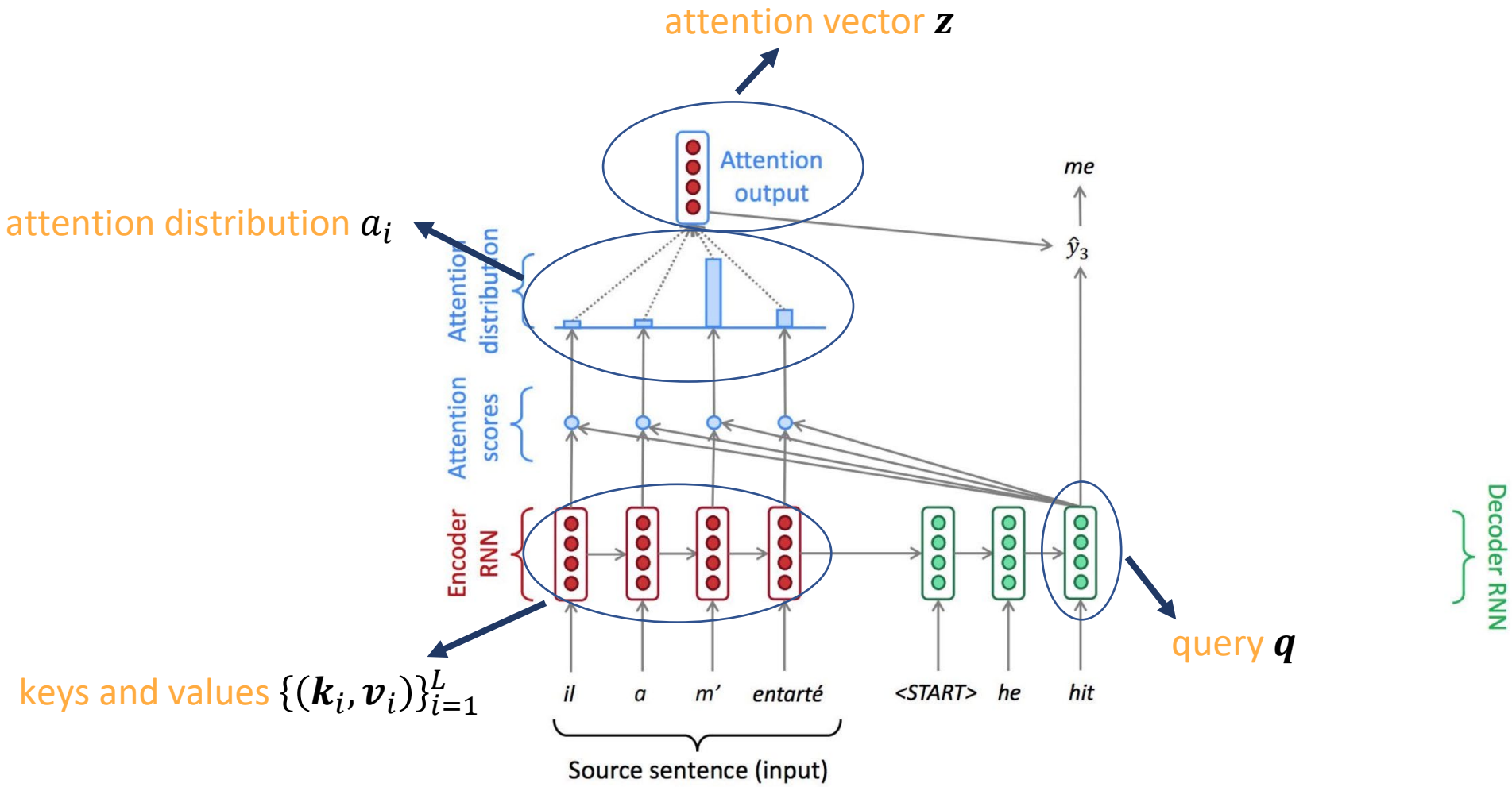
$$a_i = softmax(s_i) = \frac{\exp(s_\mathrm{i})}{\sum_{j=1}^{L} \exp(s_\mathrm{j})}.$$

- Compute the attention/context vector $\boldsymbol{z}$ as a weighted sum of values.

$$\boldsymbol{z} = \sum_{i=1}^{L} a_i \boldsymbol{v}_i.$$

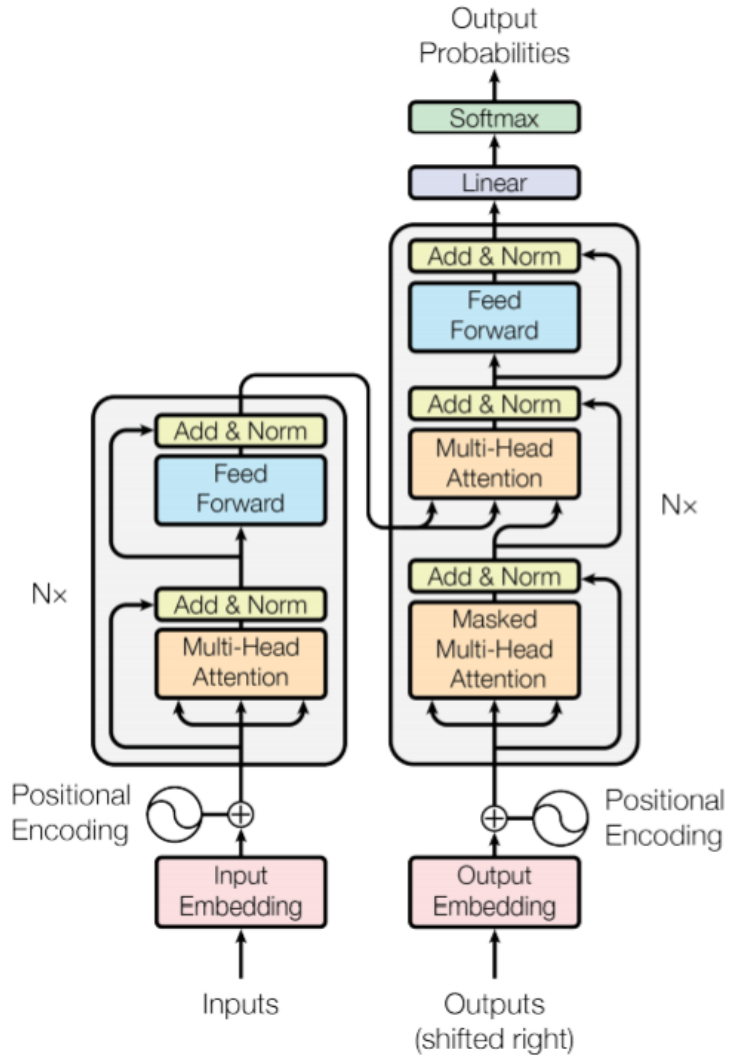
- In general, keys and values could be different and the same as well, such as in machine translations.

Attention: example

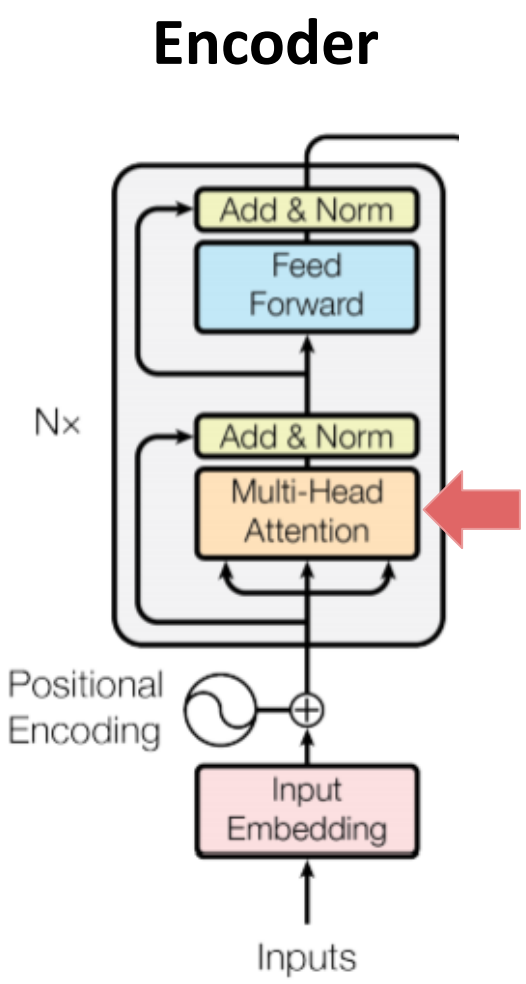


Transformer - formal description

- Encoder: maps an input sequence of symbol representations $(x_1, x_2, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, z_2, \ldots, z_n)$
- Decoder: given z, generates an output sequence $(y_1, y_2, \ldots, y_m)$, one element at a time.
- At each step the model is auto-regressive, consuming the previously generated symbols as additional input.
- Stacked self-attention and point-wise, fully connected layers for both the encoder and decoder.



Transformer: attention in encoder

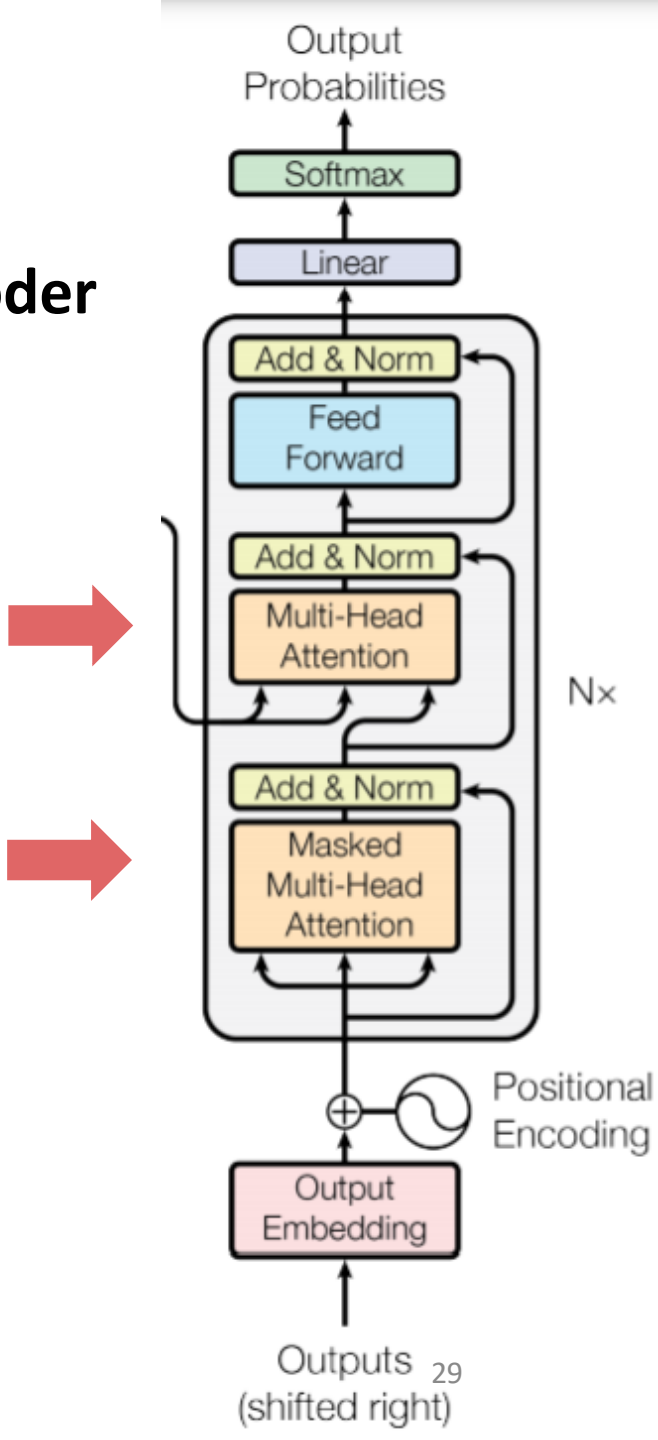


- In a self-attention layer all of the keys, values and queries come from the output of the previous layer or the raw input embedding (with hidden dimension d_{model}) in the encoder.

- Each position in the encoder can attend to all positions in the previous layer of the encoder.

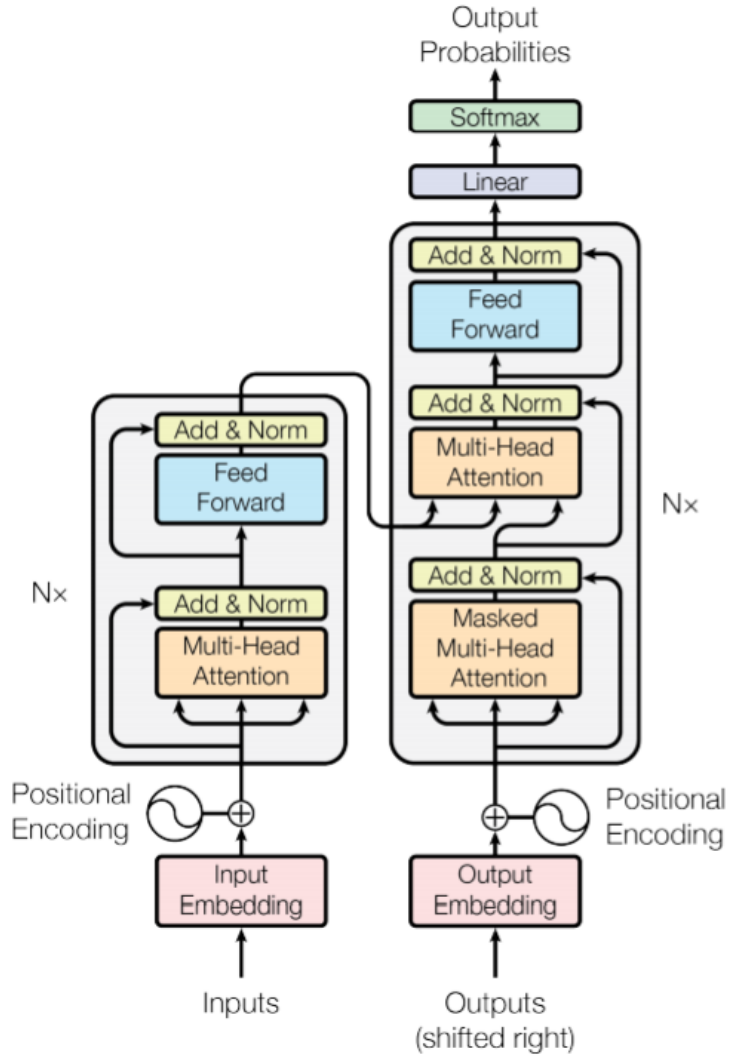
Transformer: attention in decoder

- Similarly, each position attends to all positions in the decoder up to and including that position.
- Encoder-decoder attention: keys and values come from the top encoder output; queries come from the output of the masked multi-head attention (with hidden dimension d_{model}) in the decoder.
- Masked self-attention
  - Keys, values and queries come from the output of the previous layer or the raw output embedding (with hidden dimension d_{model}) in the decoder.
  - Prevent leftward information flow to preserve the auto-regressive property. Decoder self-attention is only allowed to attend to earlier positions in the output sequence, implemented by masking out (setting to $-\infty$) future positions (tokens).



Transformer: other details

- Each of the layers in the encoder and decoder contains a fully connected feed-forward network (FFN), which is applied to each position separately and identically.
  - $FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$
- Positional encodings: input positional information of the sequence.
- Residual connection: learning the changing part
- Learned embeddings are used to convert the input tokens and output tokens to vectors of dimension d_{model}. The same weight matrix are shared in the two embedding layers.



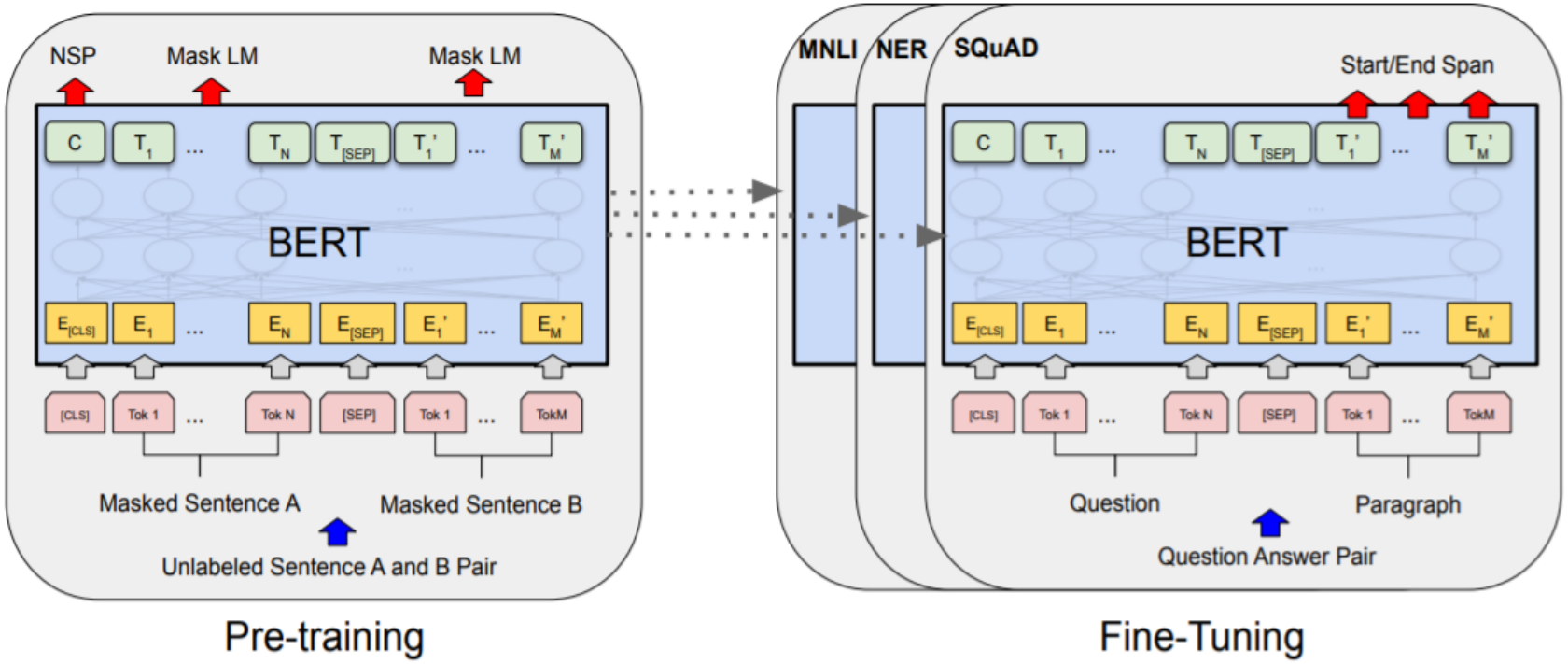
BERT architecture

- BERT architecture consist of multi-layer bidirectional transformer encoders
- BERT model provided in the paper came in two sizes

BERT_BASE	BERT_LARGE
Layers = 12	Layers = 24
Hidden size = 768	Hidden size = 1024
Self-attention heads = 12	Self-attention heads = 16
Total parameters = 110M	Total parameters = 340M

BERT: pretraining + finetuning

- Apart from output layers, the same architectures are used in both pre-training and fine-tuning.
- The same pre-trained model parameters are used to initialize models for different down-stream tasks.
- During fine-tuning, all parameters are fine-tuned.



What contributes to BERT's success?

- Innovation in model pre-training: 2 supervised tasks for pre-training instead of traditional unidirectional language models
  - Masked Language Model (MLM)
  - Next Sentence Prediction (NSP)
  - The training loss is the sum of the mean MLM likelihood and the mean NSP likelihood
- Compatibility to various tasks
  - Just fine-tune BERT Model for specific tasks to achieve state-of-the-art performance
  - BERT advances the state-of-the-art for eleven NLP tasks

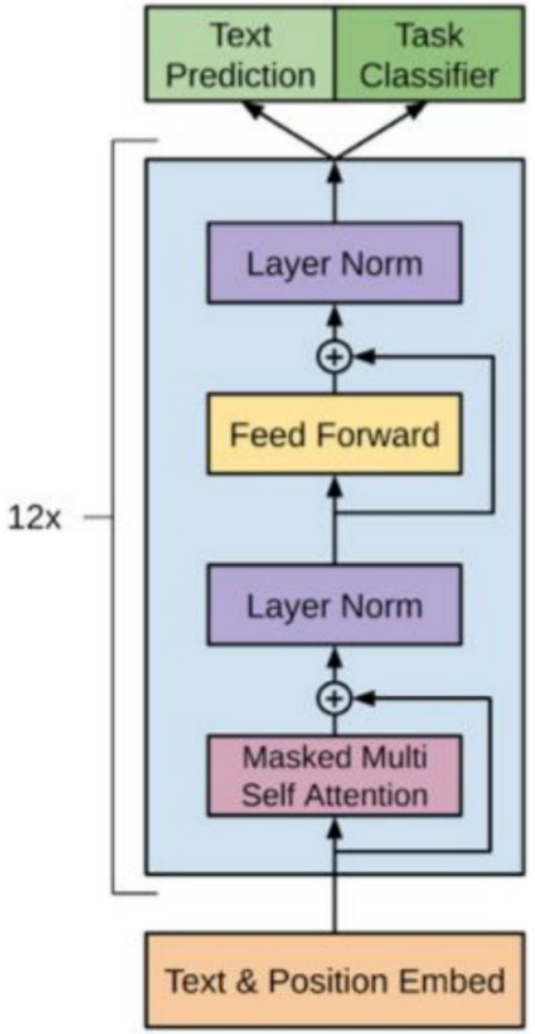
GPT framework (1)

- Multi-layer transformer decoder
  - first layer: $h_0 = UW_e + W_p$
  - the l-th layer: $h_l = transformer\_block(h_{l-1}), \forall l \in [1, n]$
  - Unsupervised pre-training, similar to embdeddings such as word2Vec
  - Given tokens *U*, maximize the contextual conditional probability

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

$$P(u) = \texttt{softmax}(h_n W_e^T)$$

where *k* is the window size; $h_n W_e^T$ is the score for each word.



GPT framework (2)

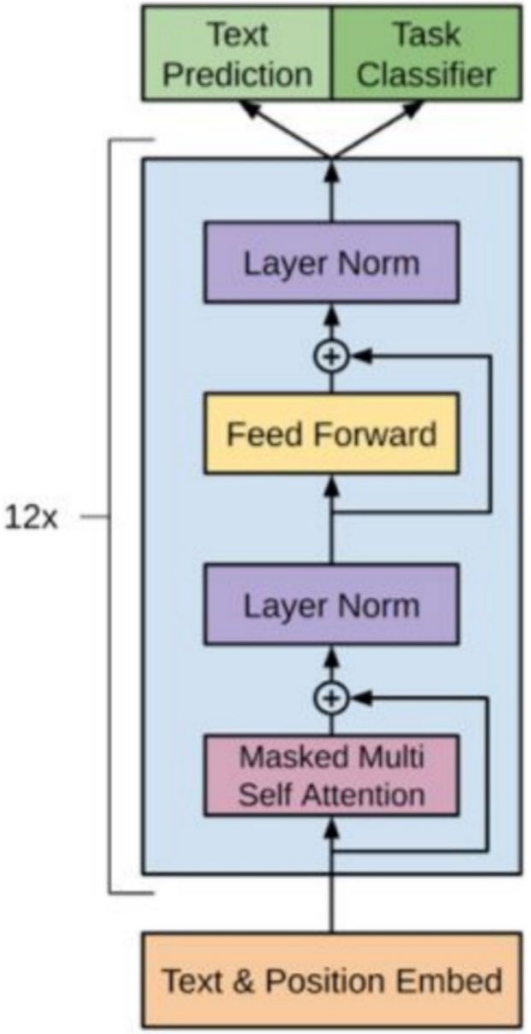
- Supervised fine-tuning
  - keep the pre-trained transformers
  - replace the final linear layer W_e with W_y,
  - Given data inputs X and labels y, maximize

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

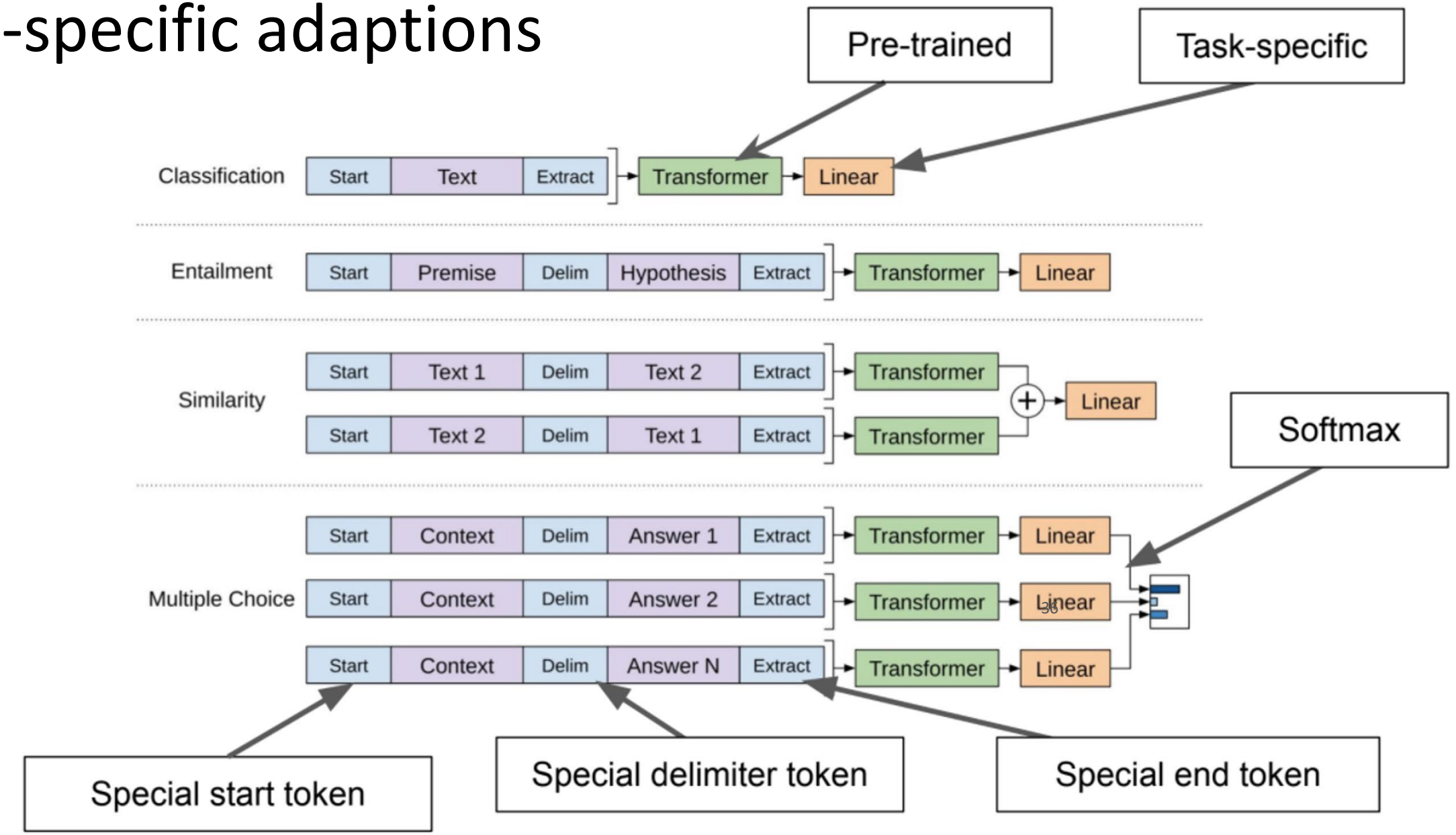
$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1,\ldots,x^m).$$

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

auxiliary training objective



Task-specific adaptions



Pre-trained model comparison: ELMo, GPT, and BERT

	Architecture	Pre-training	Downstream tasks
ELMo	Bi-directional LSTM language model	Unsupervised corpus. Learn both words and linguistic context features that support downstream tasks.	Feature-based tasks and task-specific models.
GPT	Uni-directional transformer decoder	Unsupervised corpus. Each specific task requires discriminative fine-tuning.	Model-based tasks and task-agnostic models.
BERT	Bi-directional transformer encoder	Unsupervised corpus. 2 unsupervised tasks: MLM and NSP. Each specific task requires discriminative fine-tuning.	Model-based tasks and task-agnostic models.

L5&L6 NLP Tasks

1. NLP tasks: NLU vs. NLG
2. Text classification
3. Question answering
4. Machine translation
5. Dialog

NLU vs. NLG

- NLU focuses on comprehending and extracting meaning from natural language input. It involves tasks such as text classification and question answering (reading comprehension).
- NLG focuses on generating human-like text that conveys information or communicates effectively. It involves tasks such as machine translation and dialogue generation.
- For today's lecture, we will focus on two classic NLU tasks, i.e., text classification and question answering.

NLP tasks – NLU vs. NLG

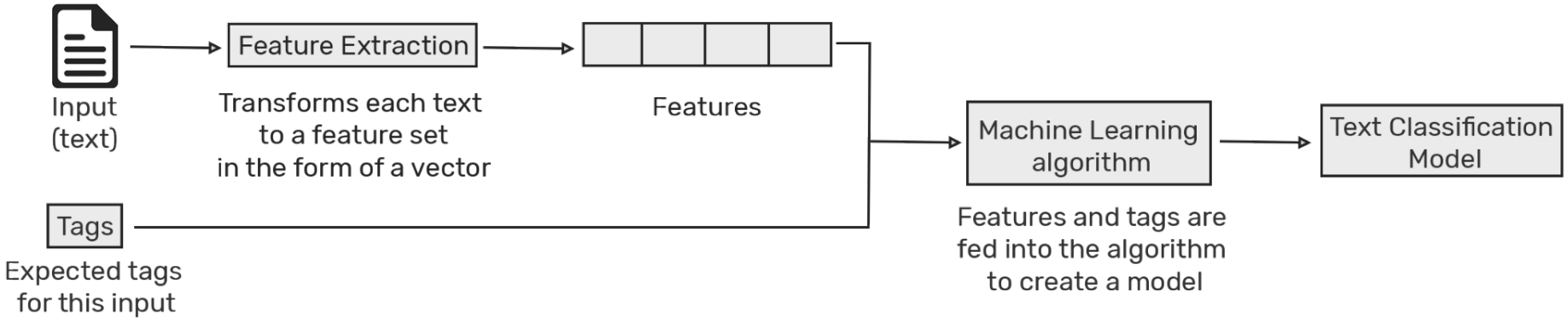
Word Tagging	Sentence Parsing	Text Classification	Text Pair Matching	Text Generation
Word segmentation	Constituency parsing	Sentiment analysis	Semantic textual similarity	Language modeling
Shallow syntax-chunking	Semantic parsing	Text classification	Natural language inference	Machine translation
Named entity recognition	Dependency parsing	Temporal processing	Relation prediction	Simplification
Part-of-speech tagging		Coreference resolution		Summarization
Semantic role labeling				Dialogue
Word sense disambiguation				Question answering

Text classification tasks

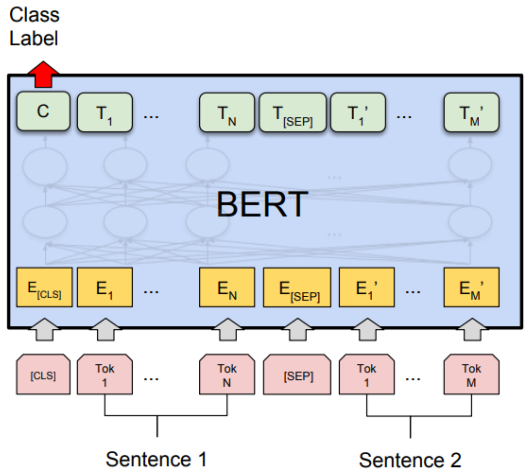
- Textual data sources
  - Textual data can come from different sources, including web data, emails, chats, social media, tickets, insurance claims, user reviews, and questions and answers from customer services
- Task examples
  - Spam detection, sentiment analysis, news categorization, user intent classification, content moderation

Text classification methods

- Feature extraction + classification



- End-to-end model



Text classification - network selection

- How to select the appropriate network for your task?
  - Select a pretrained language model (PLM), such as BERT, or GPT.
  - Domain adaptation. Adapting the PLM using in-domain data by continual pre-training the selected general-domain PLM. For domains with abundant unlabeled text, such as biomedicine, pretraining language models from scratch might also be a good choice.
  - Task-specific model design. One or more task-specific layers are added on the top to generate the final output for the target task.
  - Task-specific fine-tuning. The task-specific layers can be either trained alone with the PLM fixed or trained together with the PLM, sometimes a multi-task training may be a good choice.
  - Model compression. PLMs are expensive to serve. They often need to be compressed via e.g., knowledge distillation to meet the latency and capacity constraints in real-world applications.

Text classification - evaluation

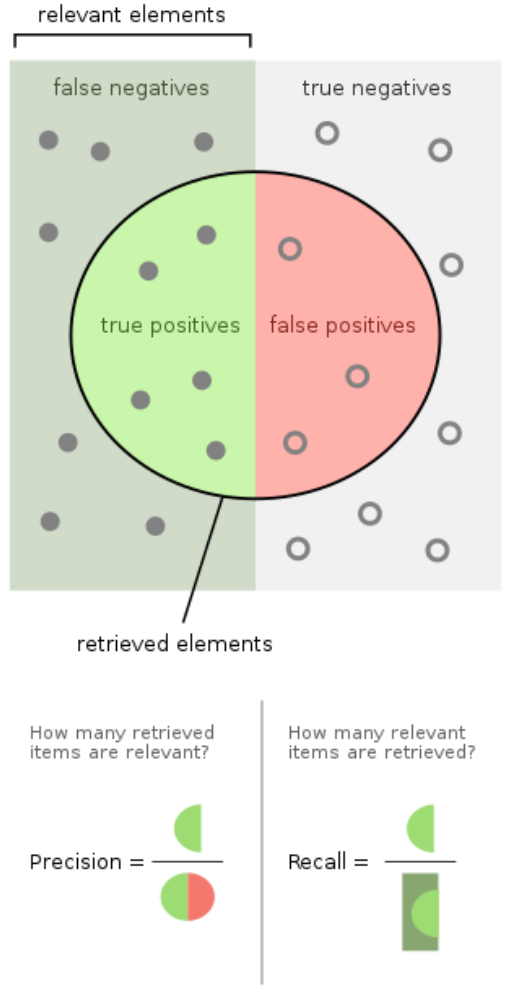
- Classification
  - Precision-recall, F1 score

$$\textbf{Precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \quad \textbf{Recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}, \quad \textbf{F1-score} = \frac{2\,\mathrm{Prec}\,\mathrm{Rec}}{\mathrm{Prec}+\mathrm{Rec}}$$

- Mean Reciprocal Rank (MRR)

$$\textbf{MRR} = \frac{1}{|Q|}\sum_{i=1}^{Q}\frac{1}{rank_i}$$

$rank_i$ refers to the rank position of the *first* relevant document for the *i*-th query

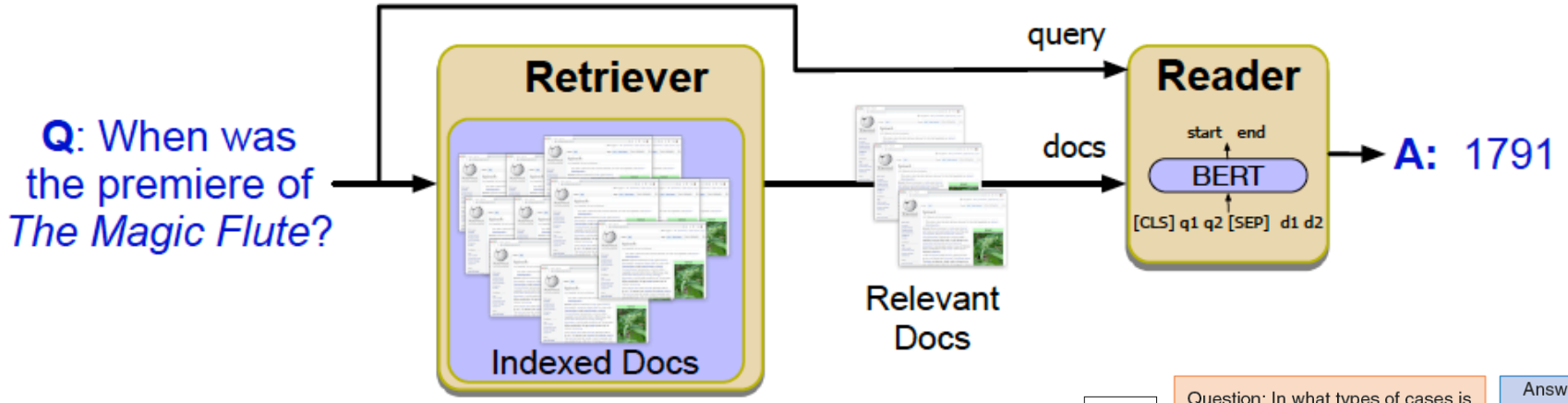


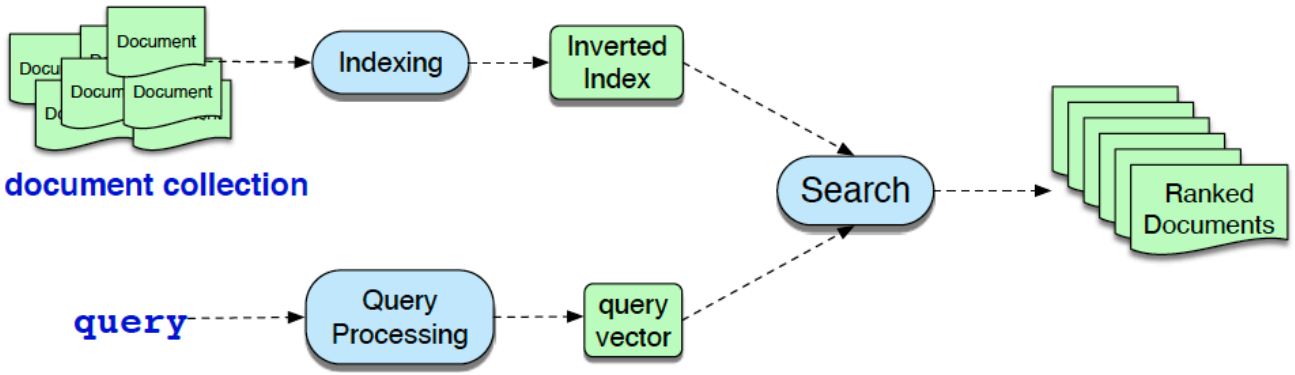
Factoid QA

- Factoid questions: questions that can be answered with simple facts expressed in short texts, like the following:
  - Where is the Louvre Museum located?
  - What is the average age of the onset of autism?
- Information-retrieval (IR) based QA (open domain question QA)
  - Given a user question, information retrieval is used to find relevant passages (vast amount of text on the web or in collections of scientific papers like PubMed). Then neural reading comprehension algorithms read these retrieved passages and draw an answer directly from spans of text.
- Knowledge-based QA
  - To build a semantic representation of the query (logic query). These meaning representations are then used to query databases of facts.
  - E.g., What states border Texas? -> λ x.state(x)$\wedge$borders(x; Texas).

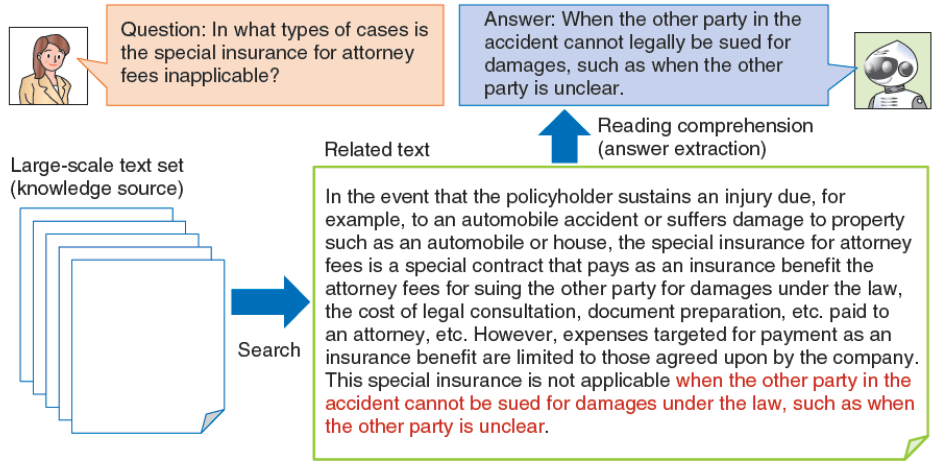
IR-based QA

- Two steps: IR + RC





IR



RC

Reading comprehension (RC)

- The ability to read and understand unstructured text and then answer questions about it.
- Probelm:
  - Input: context (passage of text) and query
  - output: answer
    - abstractive: free-form answer (more generation)
    - extractive: substring of the content (more understanding)
- Connection to Question-Answering
  - QA: a task
  - RC: a possible approach to solve QA
  - Other possible solutions to QA: knowledge-based intormation retrieval, keywords detection mechanism, etc.

Stanford question answering dataset (SQuAD)

- Rajpurkar et al., 2016, SQuAD 1.1
- SQuAD dataset motivation:
  - High quality human-written databases not very large (on the order 10^3 in size)
  - Cloze-form questions better, but not very natural
    - Semi-synthetic (As in Cloze)
    - Not explicit question answering
  - Heuristically created → noisy
- SQuAD features:
  - Questions posed by crowdworkers on a set of Wikipedia articles
  - 100,000 query-context-answer triples
  - Extractive question answering: all answers a span of text
- Why is SQuAD better?
  - Human-written, human curated → less noisy than CNN/DM
  - Not Cloze-form
  - Step towards better language understanding

SQuAD 2.0 no answer example

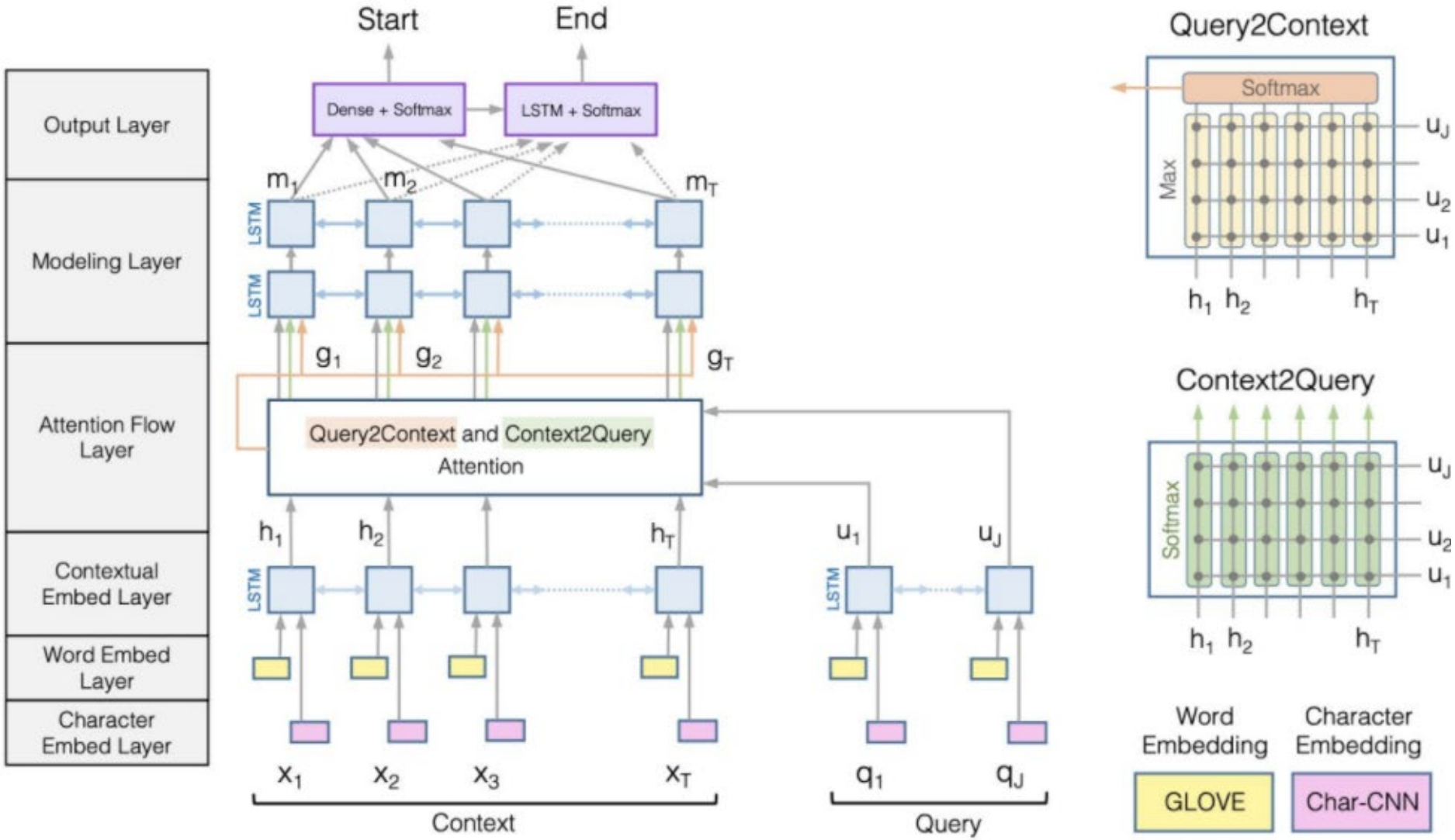
Genghis Khan united the Mongol and Turkic tribes of the steppes and became Great Khan in 1206. He and his successors expanded the Mongol empire across Asia. Under the reign of Genghis' third son, Ögedei Khan, the Mongols destroyed the weakened Jin dynasty in 1234, conquering most of northern China. Ögedei offered his nephew Kublai a position in Xingzhou, Hebei. Kublai was unable to read Chinese but had several Han Chinese teachers attached to him since his early years by his mother Sorghaghtani. He sought the counsel of Chinese Buddhist and Confucian advisers. Möngke Khan succeeded Ögedei's son, Güyük, as Great Khan in 1251. He

When did Genghis Khan kill Great Khan?

Gold Answers: <No Answer>

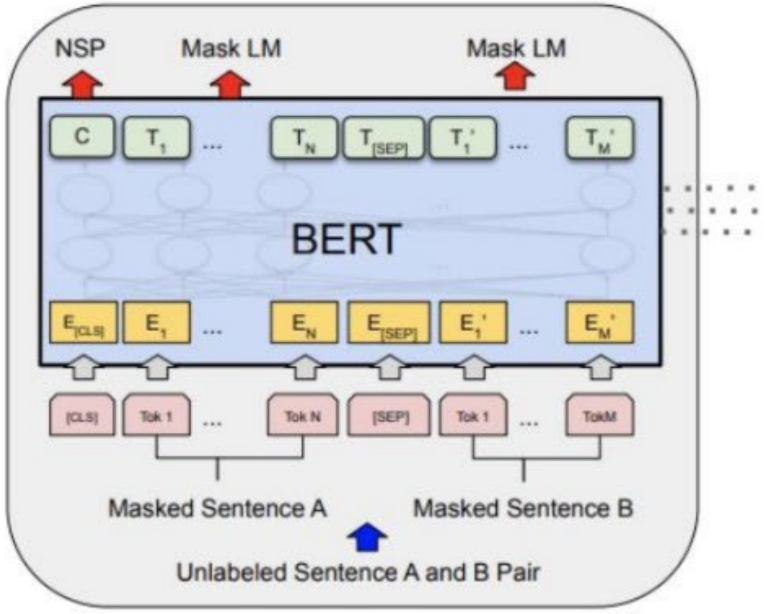
Prediction: 1234 [from Microsoft nlnet]

BiDAF architecture (Seo et al., 2017)

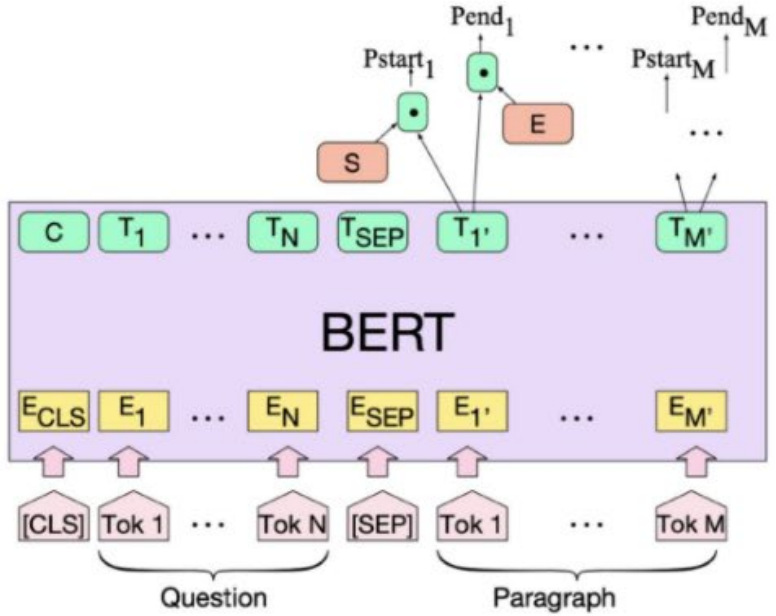


Current SOTA: pre-trained models

- Fine-tuning BERT
  - Two vectors S, E (fine-tuning parameters) to generate probabilities of start, end of each token.



Pretraining



Finetuning

$$Pstart_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$

$$Pend_i = \frac{e^{E \cdot T_i}}{\sum_j e^{E \cdot T_j}}$$

Performance metrics

- Training: log likelihood of correct start/end indices $L(\theta) = -\frac{1}{N}\sum_{i}^{N} \log(\mathbf{p}^1_{y^1_i}) + \log(\mathbf{p}^2_{y^2_i})$

- Testing: choose start-end index pair(i, j with i < j maximizing p1(i) * p2(j)
  - Remove all articles (a, an, the)
  - Exact Match (EM): choosing exactly the same start and end index as some gold answer
  - F1: treat predicted and gold answers as bags of tokens, then take harmonic mean of precision and recall

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}}\right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{precision} = \frac{\#\text{ of correctly predicted tokens}}{\#\text{ of predicted tokens}} \qquad \text{recall} = \frac{\#\text{ of correctly predicted tokens}}{\#\text{ of gold tokens}}$$

MT model evaluation – BLEU

BLEU (Bilingual Evaluation Understudy)

- BLEU compares the machine-written translation (candidate sentence) to one or several human-written translation(s) (reference sentences), and computes a similarity score based on:
  - n-gram precision p_n (usually for 1, 2, 3 and 4-grams)
  - A brevity penalty for too-short system translations

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}.$$

(c: candidate sentence length, r: reference sentence length)

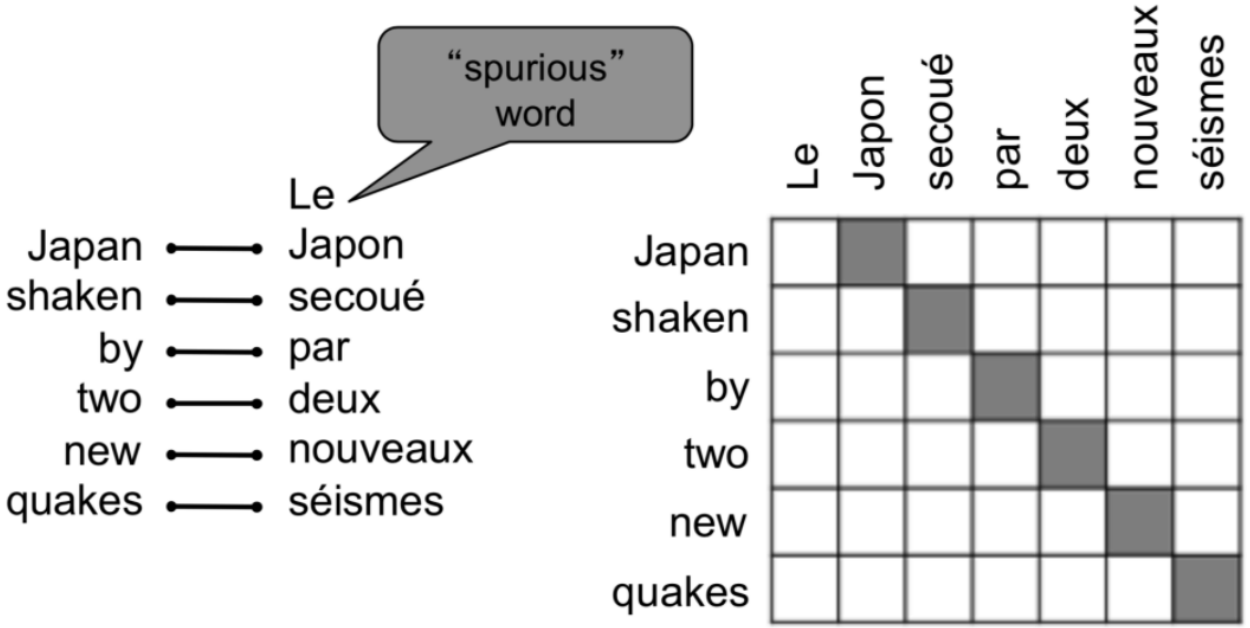
Then,

$$\mathrm{BLEU} = \mathrm{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right).$$

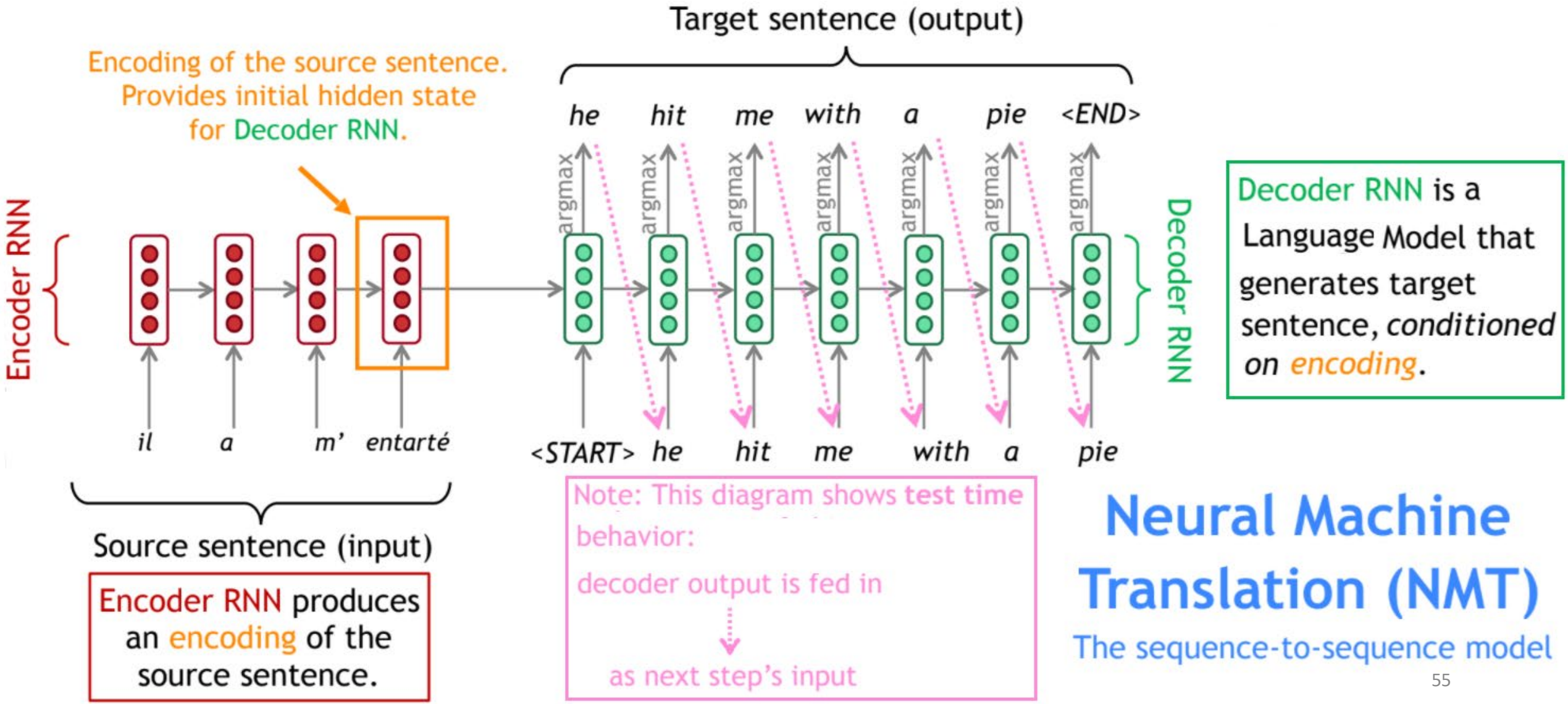
(*n*-gram precision p_n, with weights w_n)

Alignemnt details

- Alignment is the correspondence between particular words in the translated sentence pair.
- Typological differences between languages lead to complicated alignments.
- Some words have **no** counterpart.



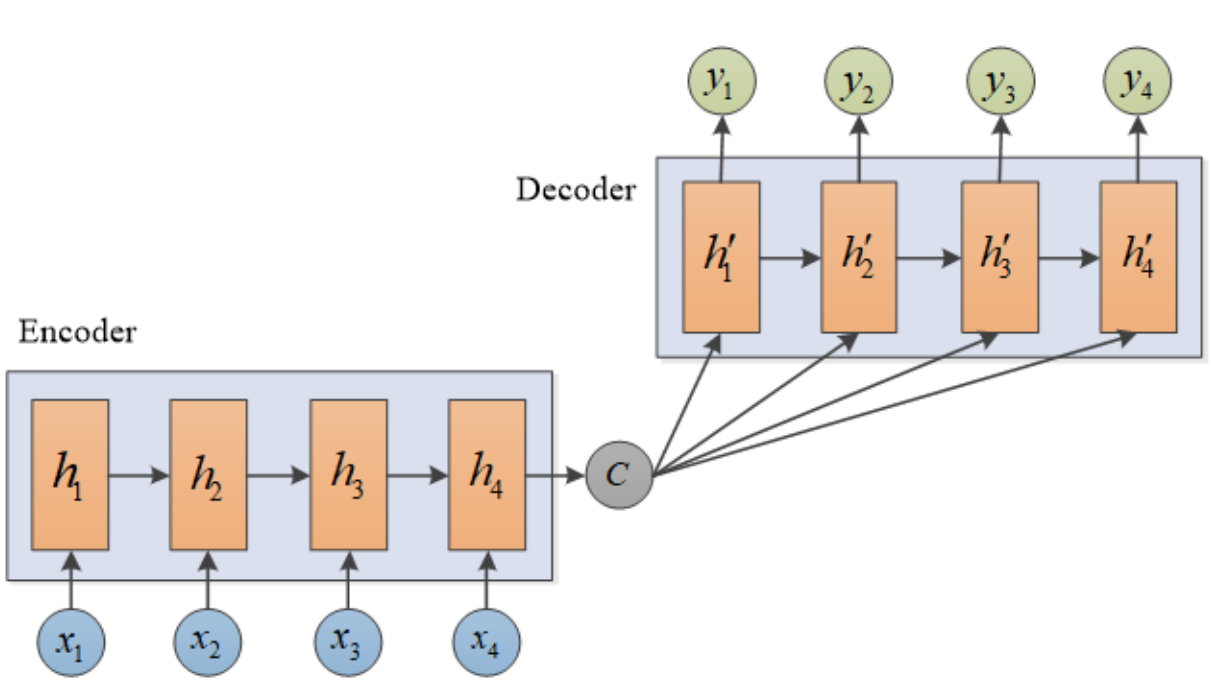
NMT with seq2seq model

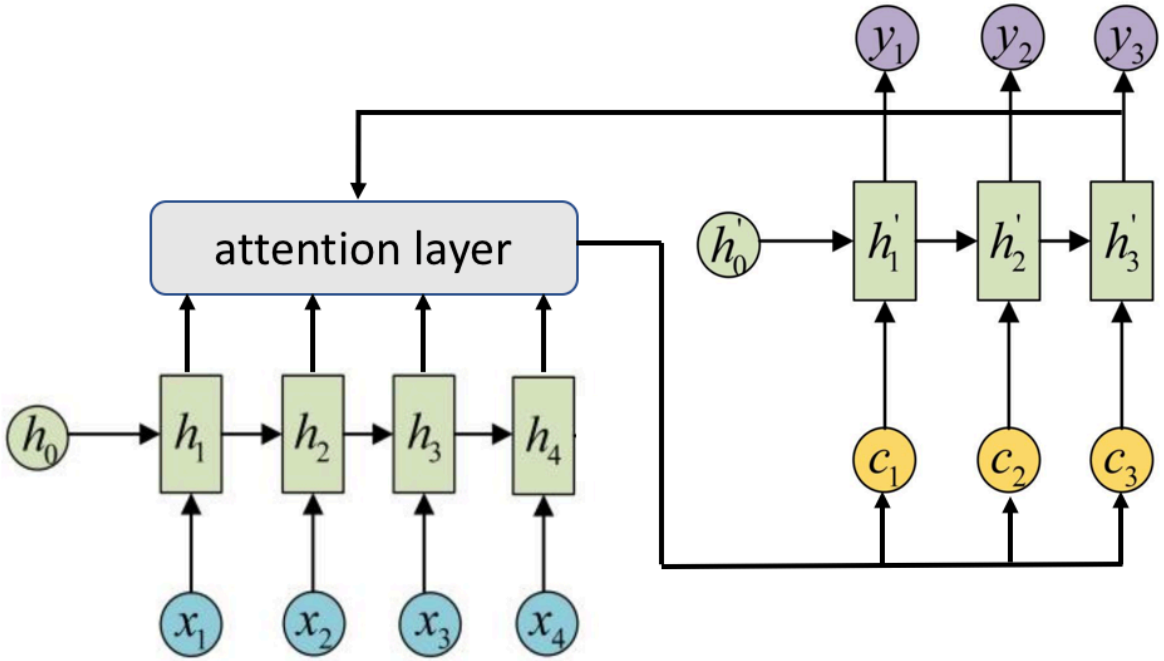


Seq2seq with vs. without attention

Without attention: conditioned on the same context (hidden state) C

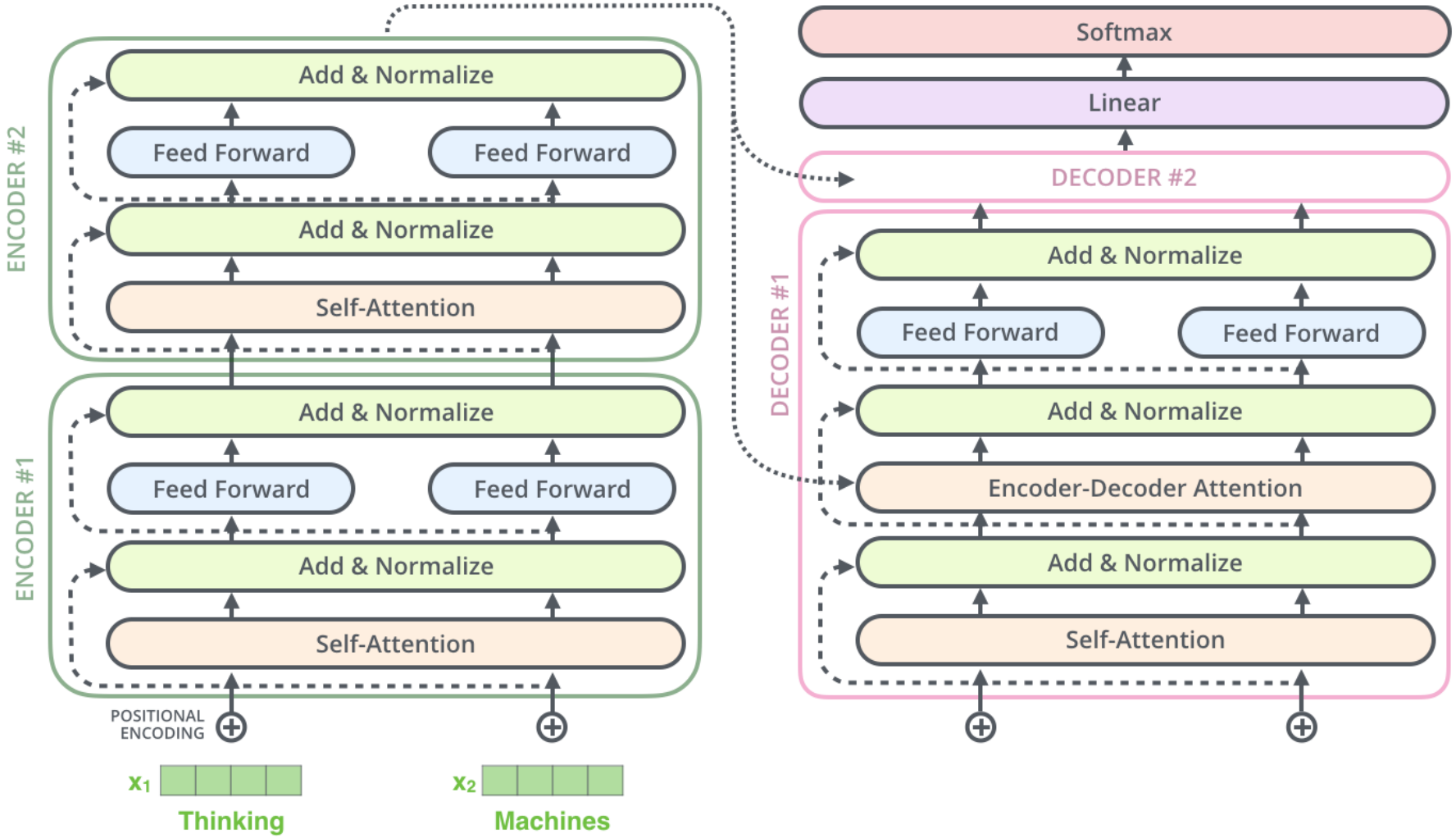
With attention: conditioned on varying context (hidden state) $c_1, c_2, c_3, \ldots$





Transformer-based MT

Encc imilar
to se



Decoding

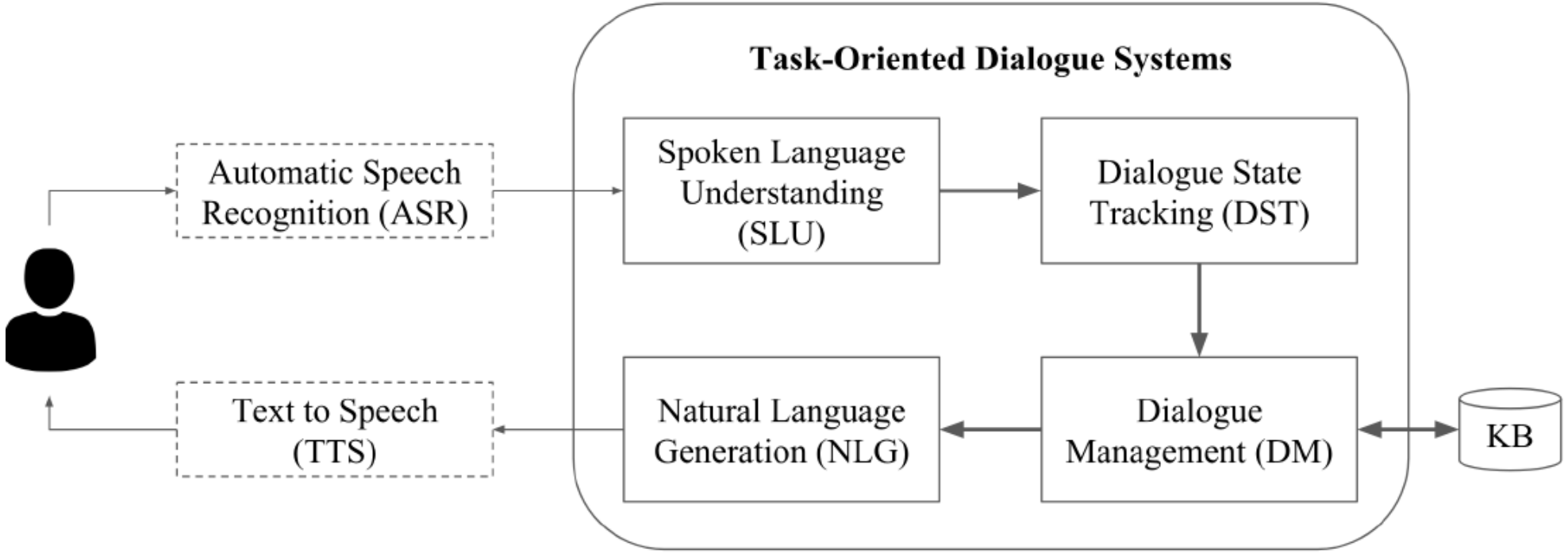
- Generally, the textual generation output is token-wise probability distribution. Decoding in textual generation is the process of converting the model's probabilistic output into coherent and meaningful text sequences.
- Greedy decoding
- Exhaustive search
- Beam search

Categories of dialogue tasks

- Task-oriented
  - open- or close- domain
  - aim at recognize the task of the user and execute corresponding tasks to accomplish the goal
  - E.g., booking a restaurant, booking movie tickets, checking account balance, etc.
- Chitchat
  - open-domain
  - aim at respond to the user input in a conversational manner
  - E.g., making socially engaging conversations

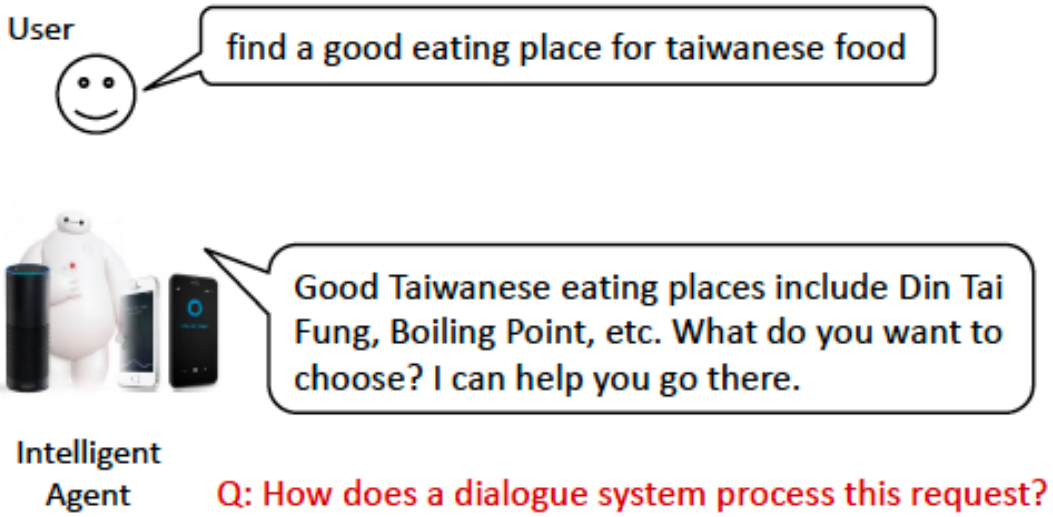
Task-oriented dialogue systems

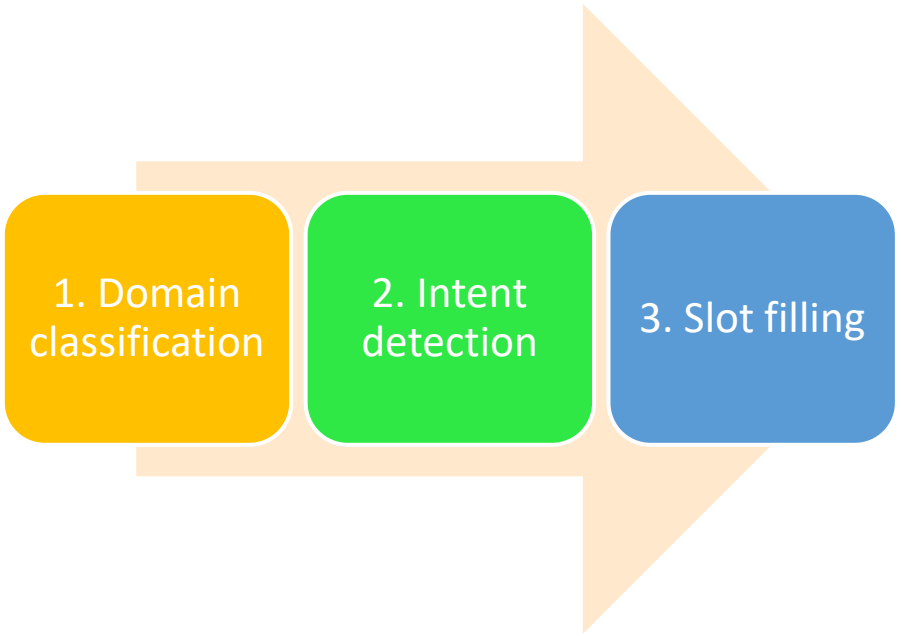
ASR (optional) -> NLU -> DST -> DM (<-> knowledge base) -> NLG -> TTS (optional)



NLU in dialogue systems

Pipelined tasks

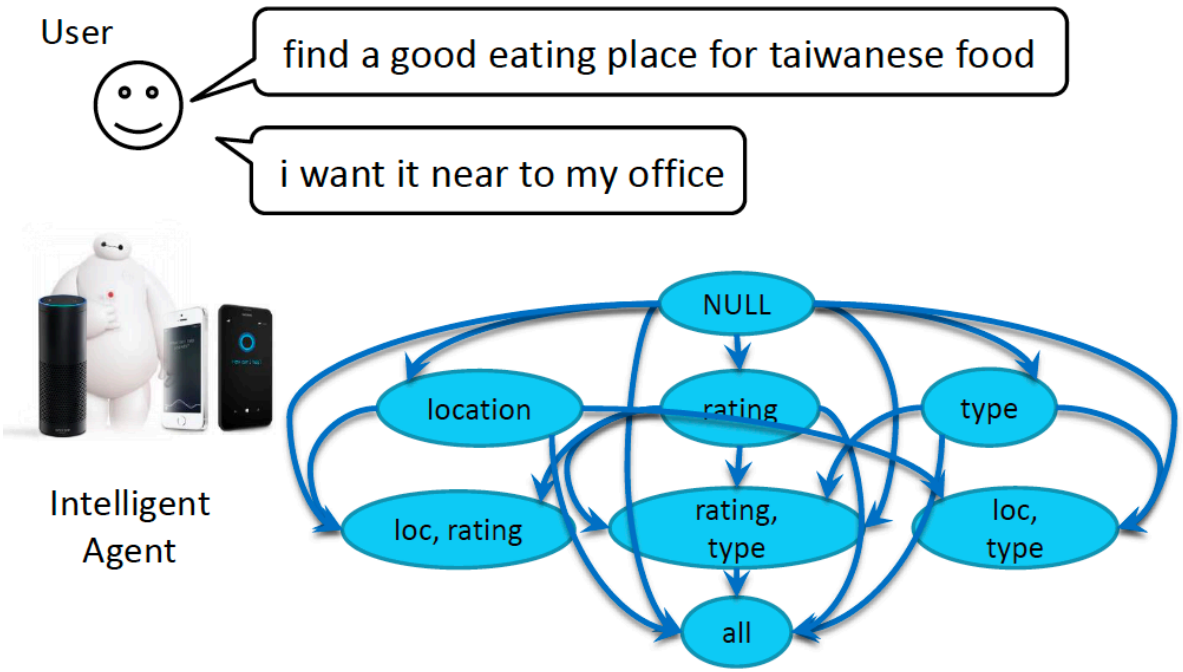




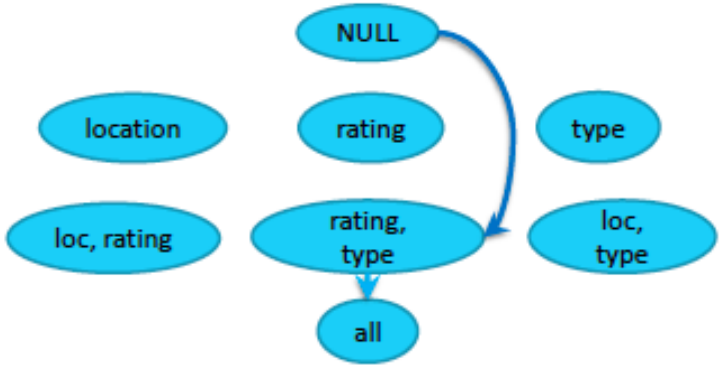
Dialogue state tracking (DST)

Requires handcrafted state machines (states, inputs, transition, outputs), like in a reinforcement learning scenario.

Observe the current dialogue state.



FIND_RESTAURANT
rating="good"
type="taiwanese"



Dialogue policy learning (DLP) for agent action

- DLP: determine what actions to choose.
- Enough information to generate an output utterance (location="Taipei 101")
  - "The nearest one is at Taipei 101"
- Not enough information? Make a request (location)
  - "Where is your home?"
- Not quite sure? Confirm (type="taiwanese")
  - "Did you want Taiwanese food?"

NLG in dialogue systems

- NLG: generate natural language or GUI given the selected dialogue action for interactions

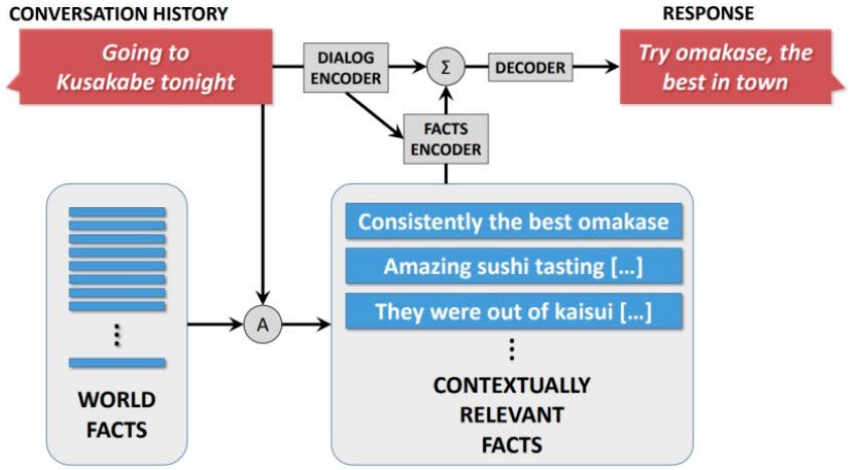
- Enough information to generate an output utterance (location="Taipei 101")
  - "The nearest one is at Taipei 101"

- Not enough information? Make a request (location)
  - "Where is your home?"

- Not quite sure? Confirm (type="taiwanese")
  - "Did you want Taiwanese food?"

Chitchat dialogue systems with knowledge ground

- Ghazvininejadet al., 2017, A knowledge-grounded neural conversation model



A: **Looking forward to trying** *@pizzalibretto* **tonight! my expectations are high**.
B: **Get** *the rocco salad*. **Can you eat** *calamari*?

A: **Anyone in** *Chi* **have a** *dentist office* **they recommend? I'm never going back to** *[...]* **and would love a reco!**
B: **Really looved** *Ora* **in** *Wicker Park*.

A: **I'm at** *California Academy of Sciences*
B: **Make sure you catch** *the show* **at** *the Planetarium*. **Tickets are usually limited.**

A: **I'm at** *New Wave Cafe*.
B: **Try to get to** *Dmitri's* **for dinner**. **Their** *pan fried scallops* **and** *shrimp scampi* **are to die for.**

A: **I just bought:** *[...] 4.3-inch portable GPS navigator* **for my wife, shh, don't tell her**.
B: **I heard this brand** *loses battery power*.

Evaluation

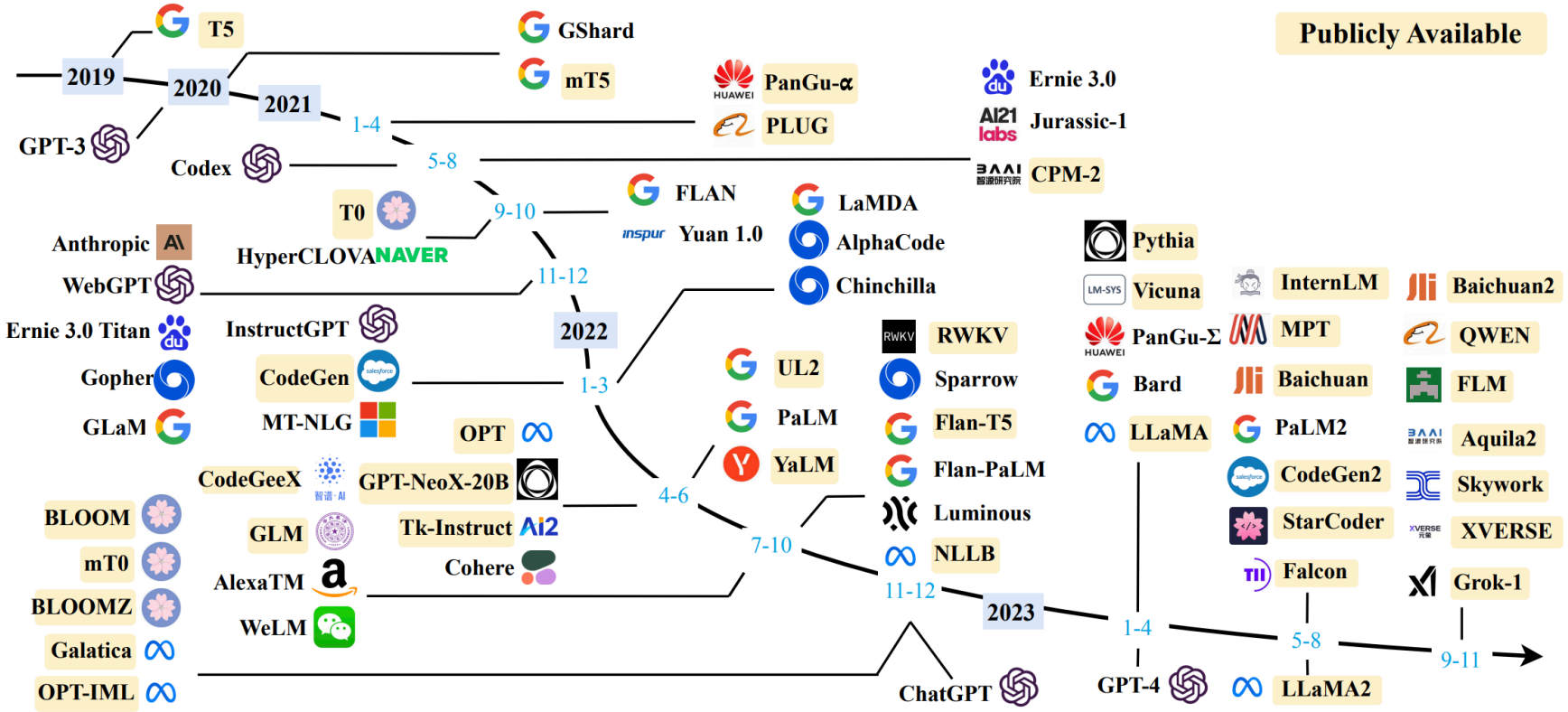
- Ideally want human evaluation, but it is not feasible for training, expensive even for evaluation
- Heuristics: perplexity, BLEU, dialogue length/diversity, fluency, engagingness, and consistency
- Still an open question

L7 LLMs

- 1. Large language model definition
- 2. Scaling law and emergent abilities
- 3. Alignment

LLMs have powerful textual processing abilities

- LLMs can write and debug computer programs, mimic the style of celebrity CEOs and write business pitches, compose music, teleplays, fairy tales and student essays, answer test questions (sometimes, depending on the test, at a level above the average human test-taker), write poetry and song lyrics, emulate a Linux system; simulate entire chat rooms, play games like tic-tac-toe and simulate an ATM.

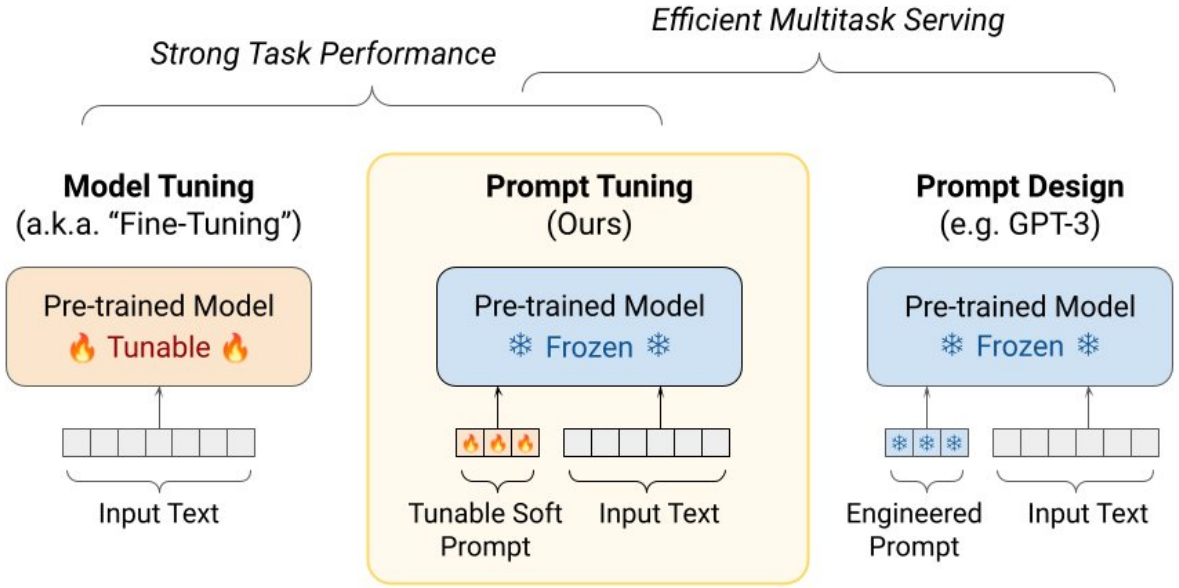


Comparisons between SLMs and LLMs

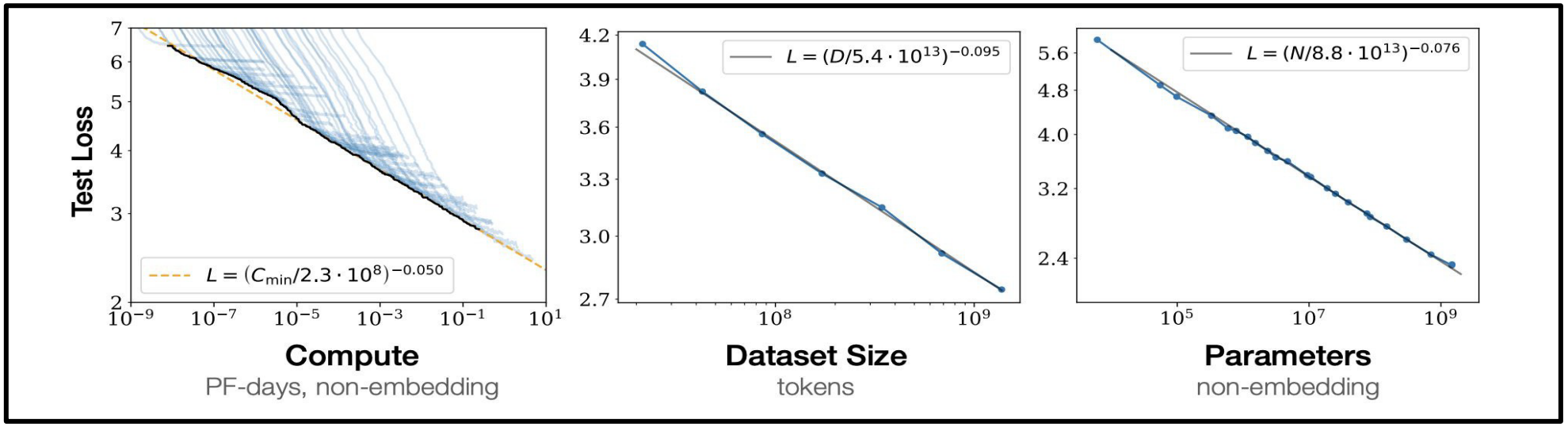
Aspect	Small Language Models	Large Language Models
Size	Can have less than 15 million parameters.	Can have hundreds of billions of parameters.
Computational Requirements	Can use mobile device processors.	Can require hundreds of GPU processors.
Performance	Can handle simple tasks.	Can handle complex, diverse tasks.
Deployment	Easier to deploy in resource-constrained environments.	Deployment often requires substantial infrastructure.
Training	Can be trained a week.	Training can take months.

Paradigm shift

- **From pretraining-finetuning to prompting**
- **Pretraining-finetuning:** pretrained model is tunable and adapted downstream tasks
- **Prompting:** pretrained model is often frozen and some input prefix will be added to use the model to perform different tasks



Scaling law of LLMs



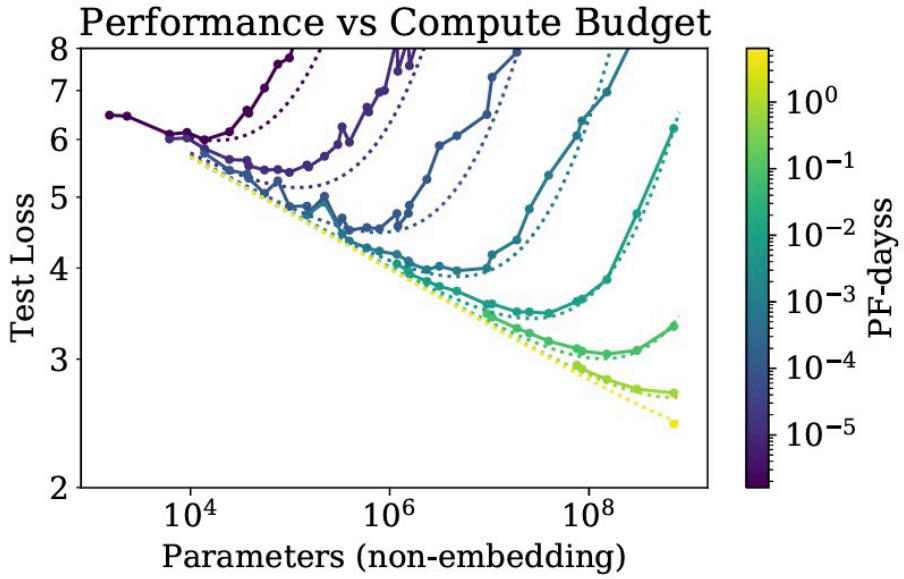
Performance depends strongly on scale, weakly on model shape: Model performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width. (Section 3)

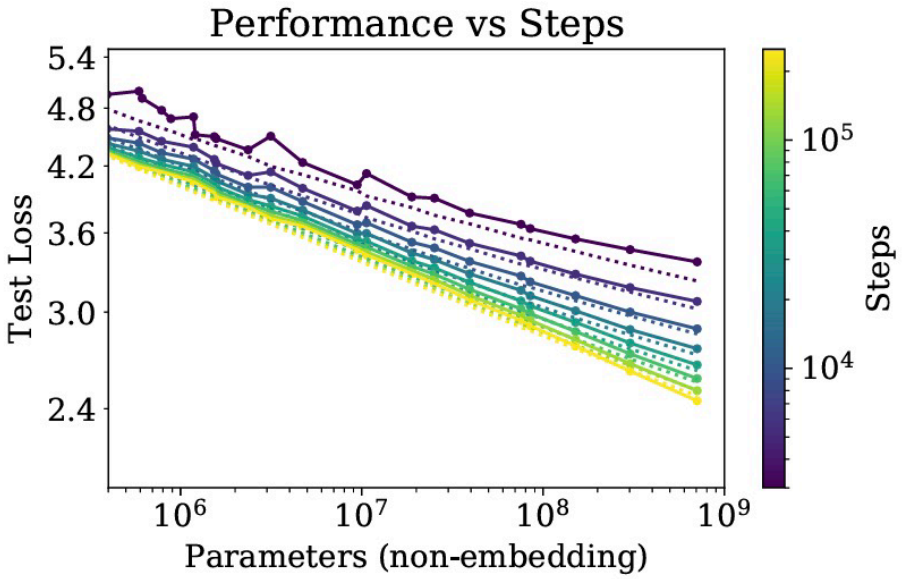
Smooth power laws: Performance has a power-law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two, with trends spanning more than six orders of magnitude (see Figure 1). We observe no signs of deviation from these trends on the upper end, though performance must flatten out eventually before reaching zero loss. (Section 3)

Scaling law of LLMs

Universality of overfitting: Performance improves predictably as long as we scale up N and D in tandem, but enters a regime of diminishing returns if either N or D is held fixed while the other increases. The performance penalty depends predictably on the ratio $N^{0.74}/D$, meaning that every time we increase the model size 8x, we only need to increase the data by roughly 5x to avoid a penalty. (Section 4)

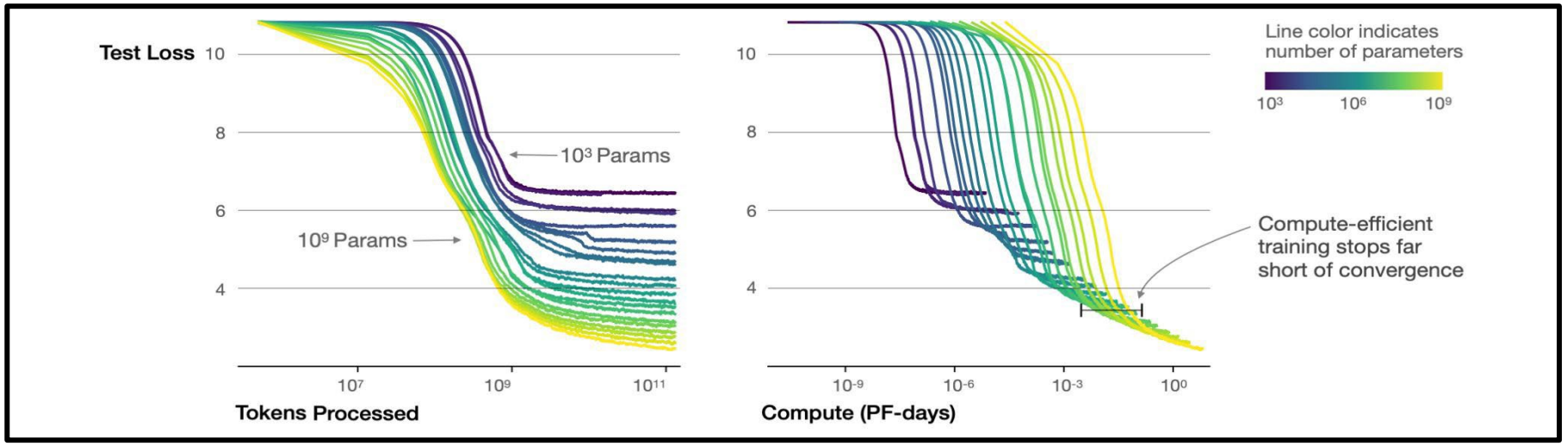
Universality of training: Training curves follow predictable power-laws whose parameters are roughly independent of the model size. By extrapolating the early part of a training curve, we can roughly predict the loss that would be achieved if we trained for much longer. (Section 5)





Scaling law of LLMs

Sample efficiency: Large models are more sample-efficient than small models, reaching the same level of performance with fewer optimization steps (Figure 2) and using fewer data points (Figure 4).

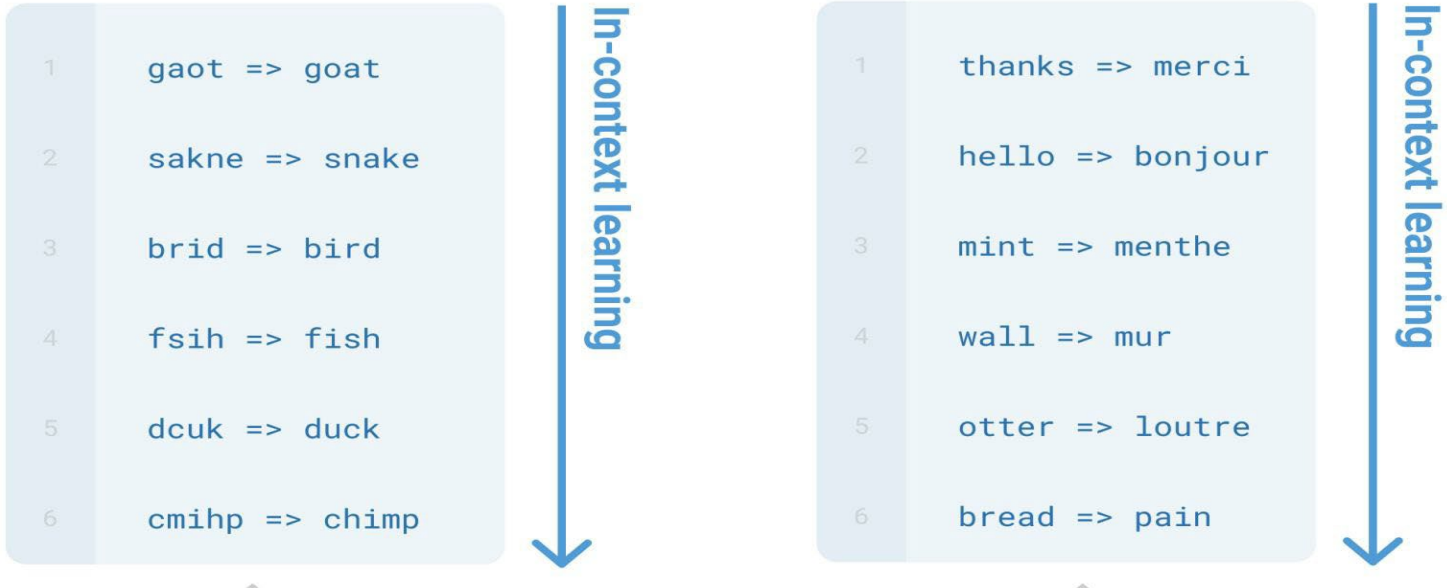


Emergent abilities: zero-shot/few-shot learning

GPT-3 (175B parameters; Brown et al., 2020)

- Another increase in size (1.5B -> **175B**)
- and data (40GB -> **over 750GB, Wikipedia, books, journals, Reddit links, Common Crawl, and other data**)
- Only training language models, amazingly achieve task-solving ability for other tasks (text generation, machine translation, reading comprehension)!

Zero shot may be too hard. Few shot learning (In-context learning) performs well!



Emergent abilities: 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. ✓

[Wei et al., 2022; also see Nye et al., 2021]

InstructGPT: Reinforcement Learning from Human Feedback

30k tasks

Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

Explain the moon landing to a 6 year old

A labeler demonstrates the desired output behavior.

Some people went to the moon...

This data is used to fine-tune GPT-3 with supervised learning.

SFT

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

Explain the moon landing to a 6 year old

A: Explain gravity...

B: Explain war...

C: Moon is natural satellite of...

D: People went to the moon...

A labeler ranks the outputs from best to worst.

D > C > A = B

This data is used to train our reward model.

RM

D > C > A = B

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

Write a story about frogs

The policy generates an output.

PPO

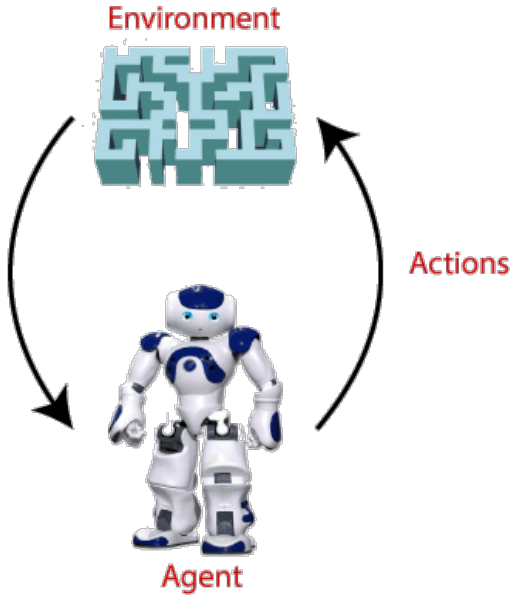
Once upon a time...

The reward model calculates a reward for the output.

RM

The reward is used to update the policy using PPO.

r_k



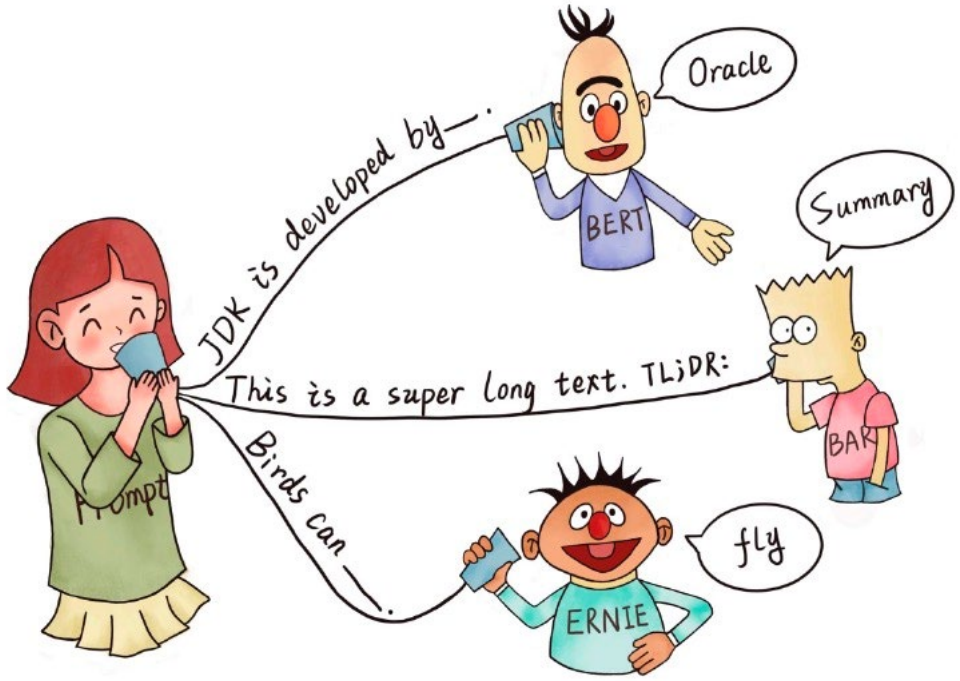
[Ouyang et al., 2022]

L8 Prompt Learning and Alignment

- 1. Prompt Learning
- 2. Chain of Thought Prompting
- 3. Alignment

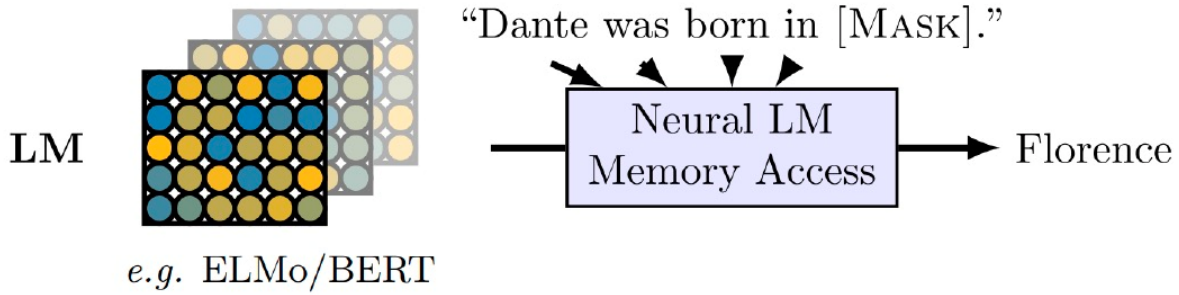
What is a prompt?

- A prompt is a piece of text inserted in the input examples, so that the original task can be formulated as a (masked) language modeling problem.
- For example, we want to classify the sentiment of the movie review *"No reason to watch"*, we can append a prompt "It was" to the sentence, getting *"No reason to watch.* It was ____." It is natural to expect a higher probability from the language model to generate "terrible" than "great".

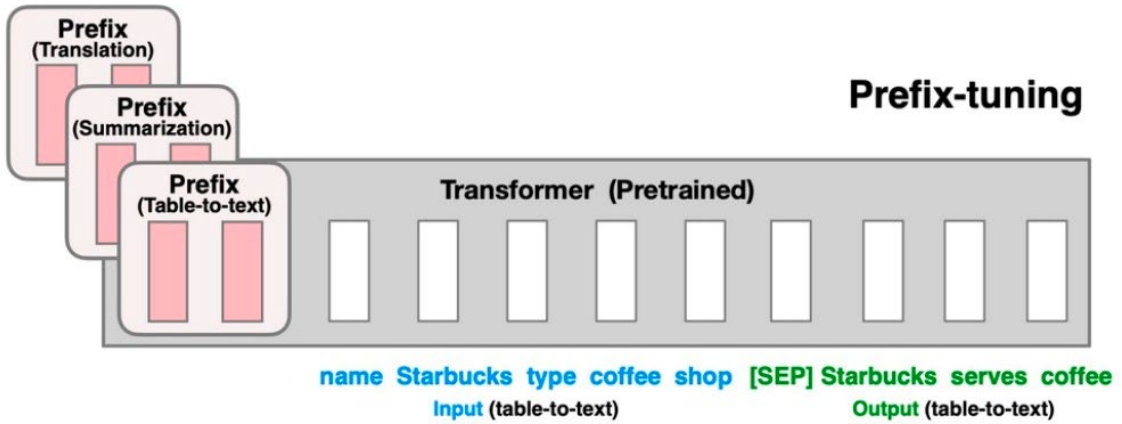


Prompt shape

- Cloze prompts: fill in the blanks of a textual string

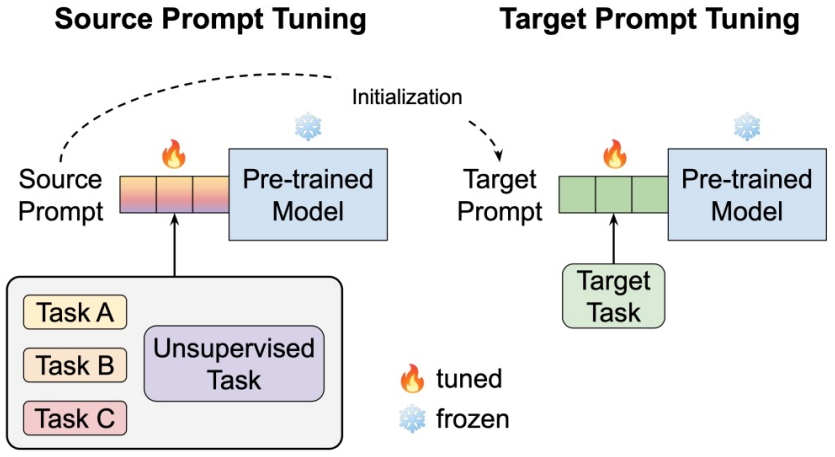


- Prefix prompts: continue a string prefix



Continuous prompts – an example

- Continuous prompts remove two constraints:
  - relax the constraint that the embeddings of template words be the embeddings of natural language (e.g., English) words.
  - Remove the restriction that the template is parameterized by the pre-trained LM's parameters.
- Prefix tuning [3] is a method that prepends a sequence of continuous task-specific vectors to the input, while keeping the LM parameters frozen.



We learn a single generic source prompt on one or more source tasks, which is then used to initialize the prompt for each target task.

One-shot Learning and few-shot learning

Similarly we have one-shot learning and few-shot learning

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

Few-shot

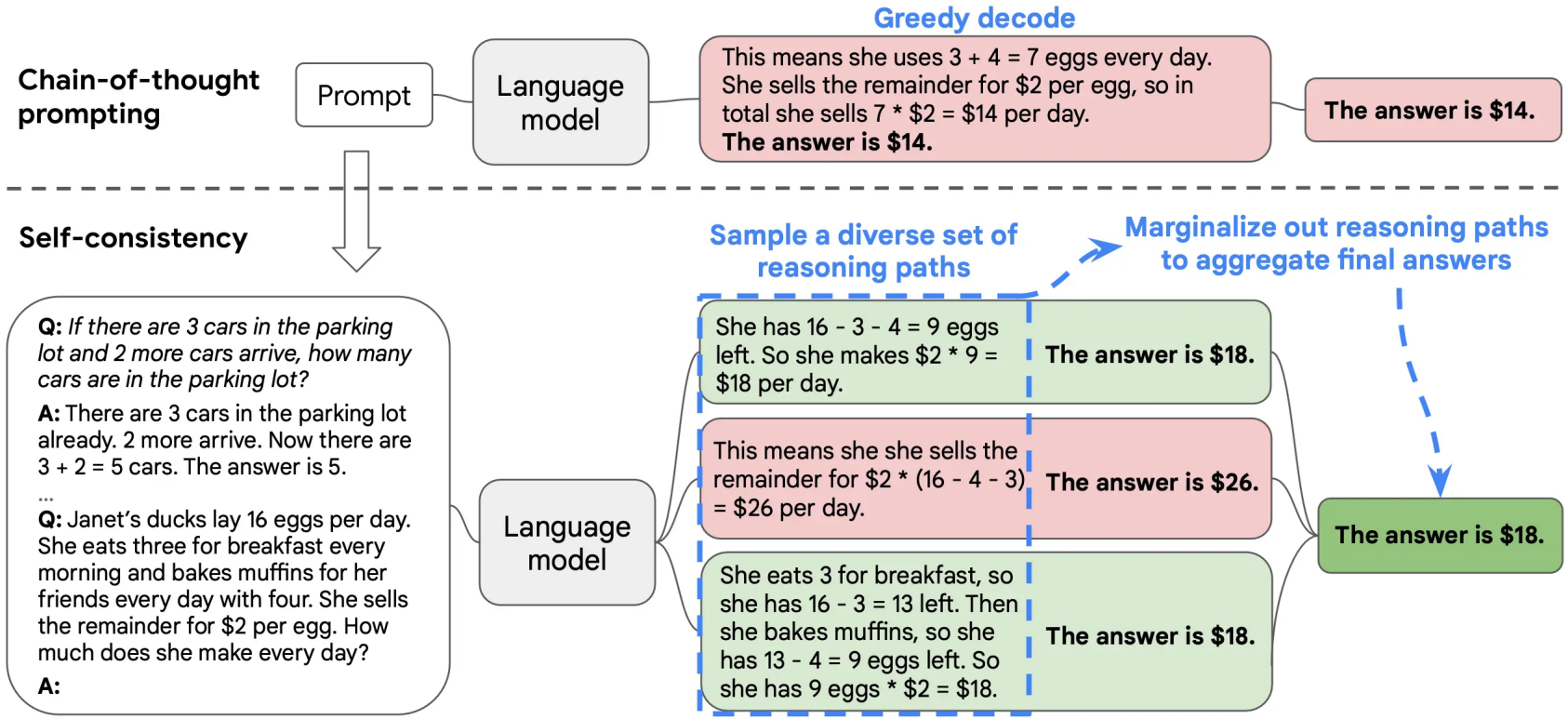
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée        ←
plush girafe => girafe peluche      ←
cheese =>                           ← prompt
```

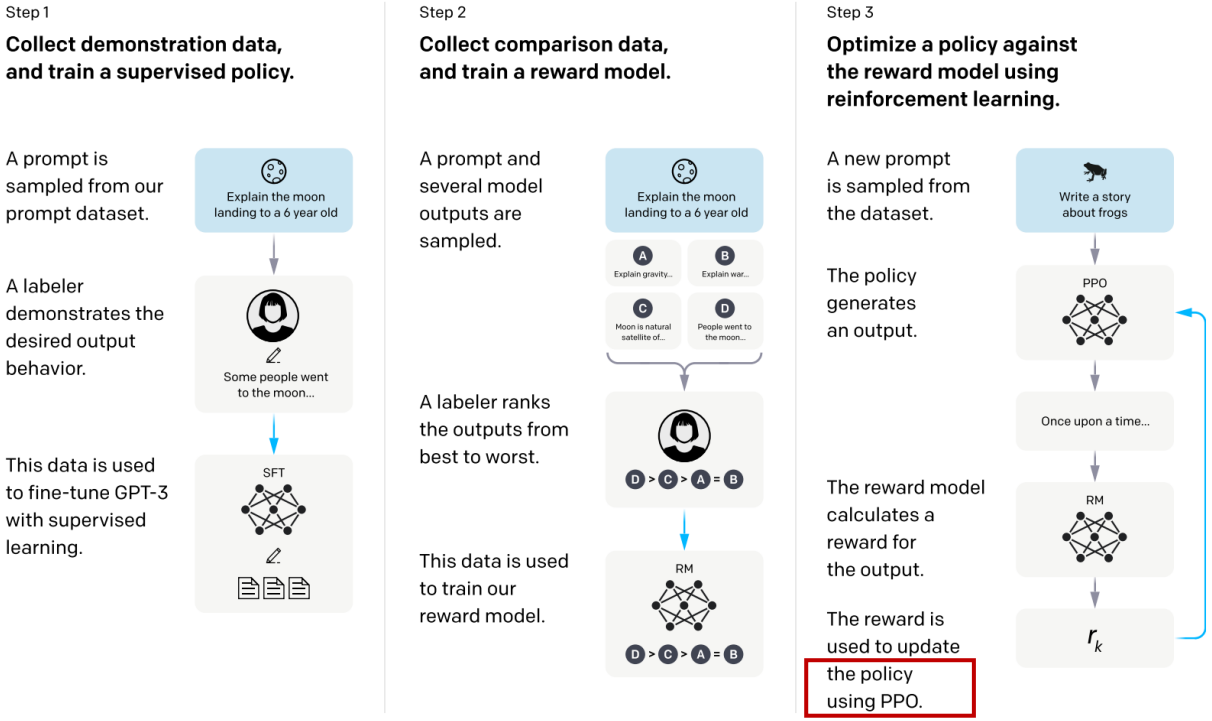
Chain of thought prompting – an example

Self-consistency Improves Chain Of Thought Reasoning in Language Models

A series of reasoning paths are sampled from the decoder of the large model. Each path can correspond to a final answer. We can select more paths that can obtain consistent answers as the reasoning paths obtained by our sampling. Based on this direct voting strategy, it is more in line with human intuition, that is, if many reasoning paths can get a corresponding answer, then the confidence of this answer will be relatively high.



Policy Gradient



The RL methods for LLM alignment which we will introduce later are basically derived from the policy gradient

Three RL Methods for LLM Alignment

- **PPO** (Proximal Policy Optimization): A RL algorithm used to train agents by optimizing their policies.
- **DPO** (Direct Preference Optimization): Align LLMs using human preferences without complex RL.
- **GRPO** (Group Relative Policy Optimization): A hybrid RL algorithm specifically designed to enhance reasoning capabilities in LLMs.

PPO

What is PPO?

- PPO (Proximal Policy Optimization): A popular reinforcement learning algorithm for training policies.
- Key Idea: Balances sample efficiency and stability during training.
- Core Mechanism: Uses a clipped objective to prevent large policy updates, ensuring smoother learning.

[Schulman et al., 2017]

DPO
How It Works

- Collect paired comparisons of model outputs (preferred vs. non-preferred)
- Directly optimize the LLM using a supervised loss function based on preferences.

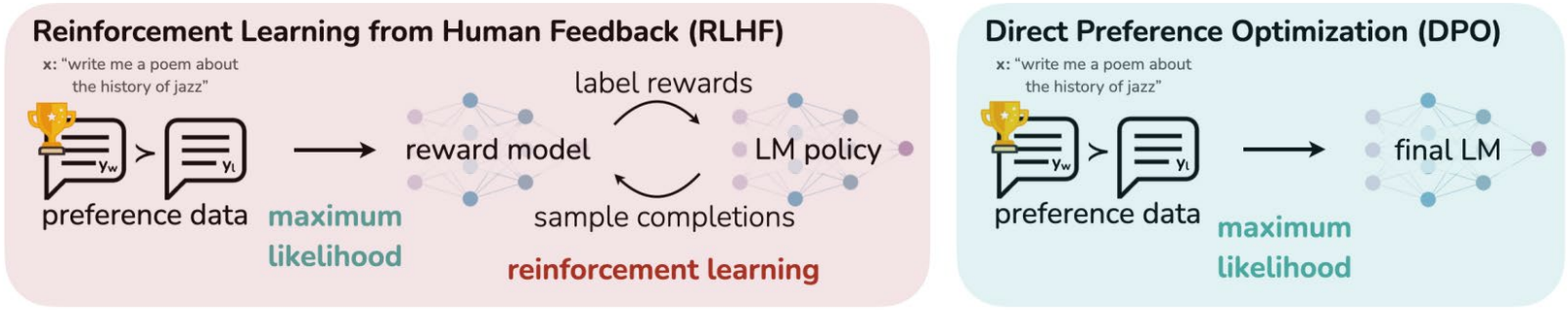


Figure: DPO optimizes for human preferences while avoiding reinforcement learning

GRPO vs PPO

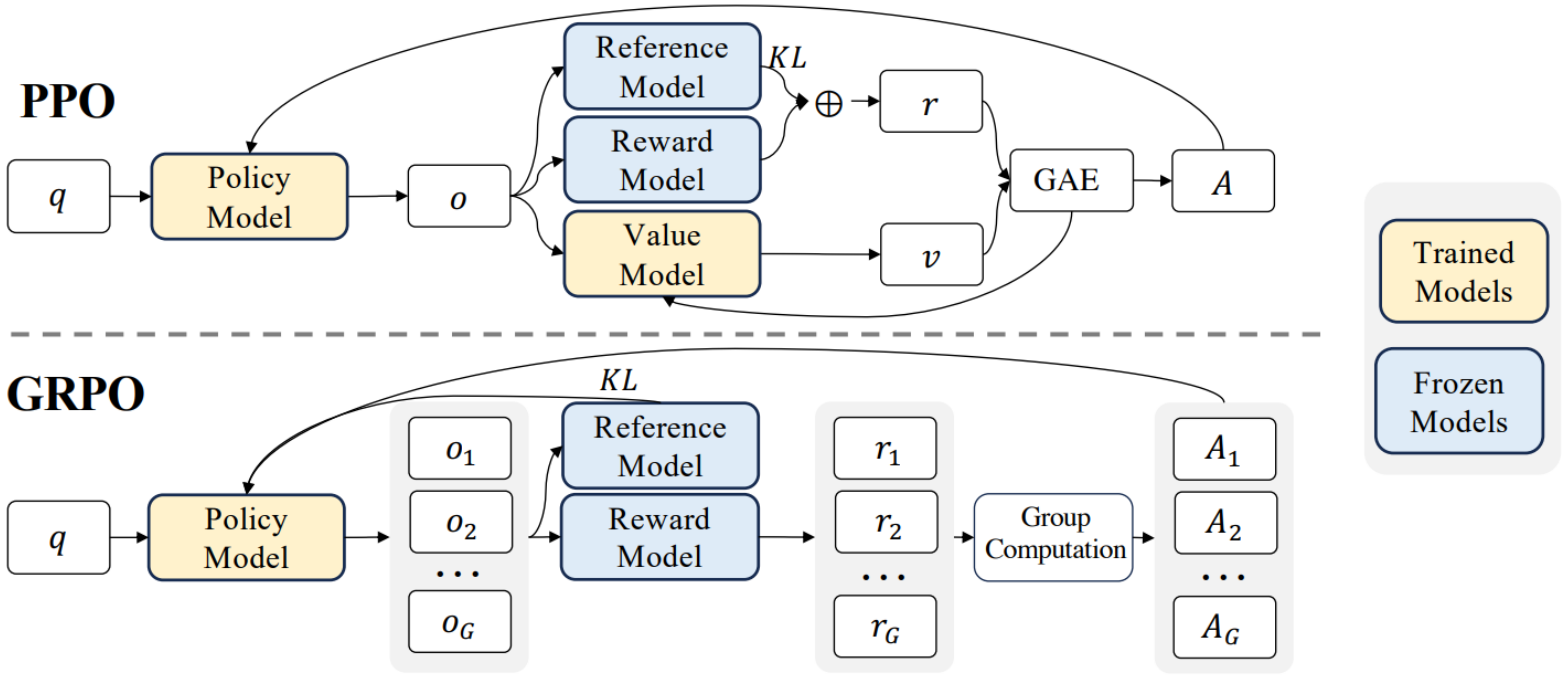
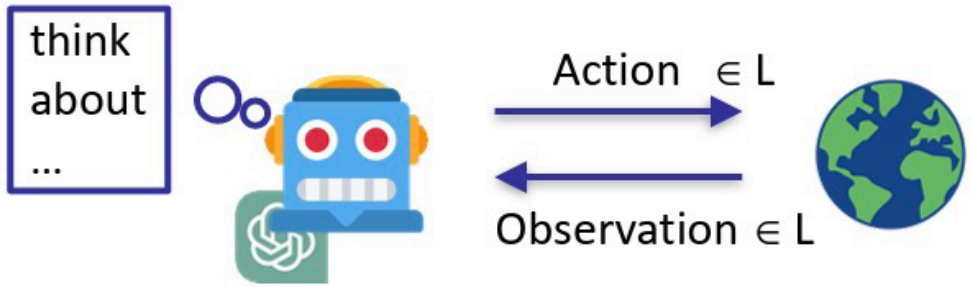


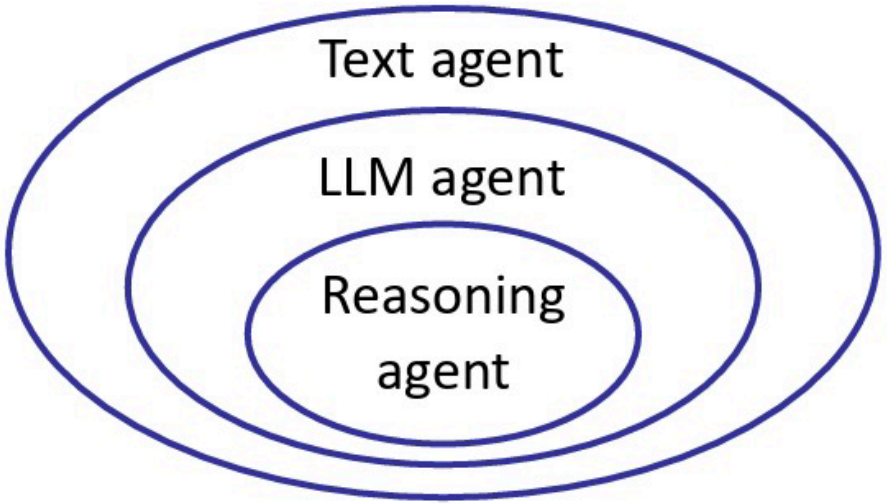
Figure: Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

L9 LLM Agents

- 1. LLM agents introduction
- 2. Reasoning
- 3. Tool learning
- 4. Knowledge incorporation-RAG

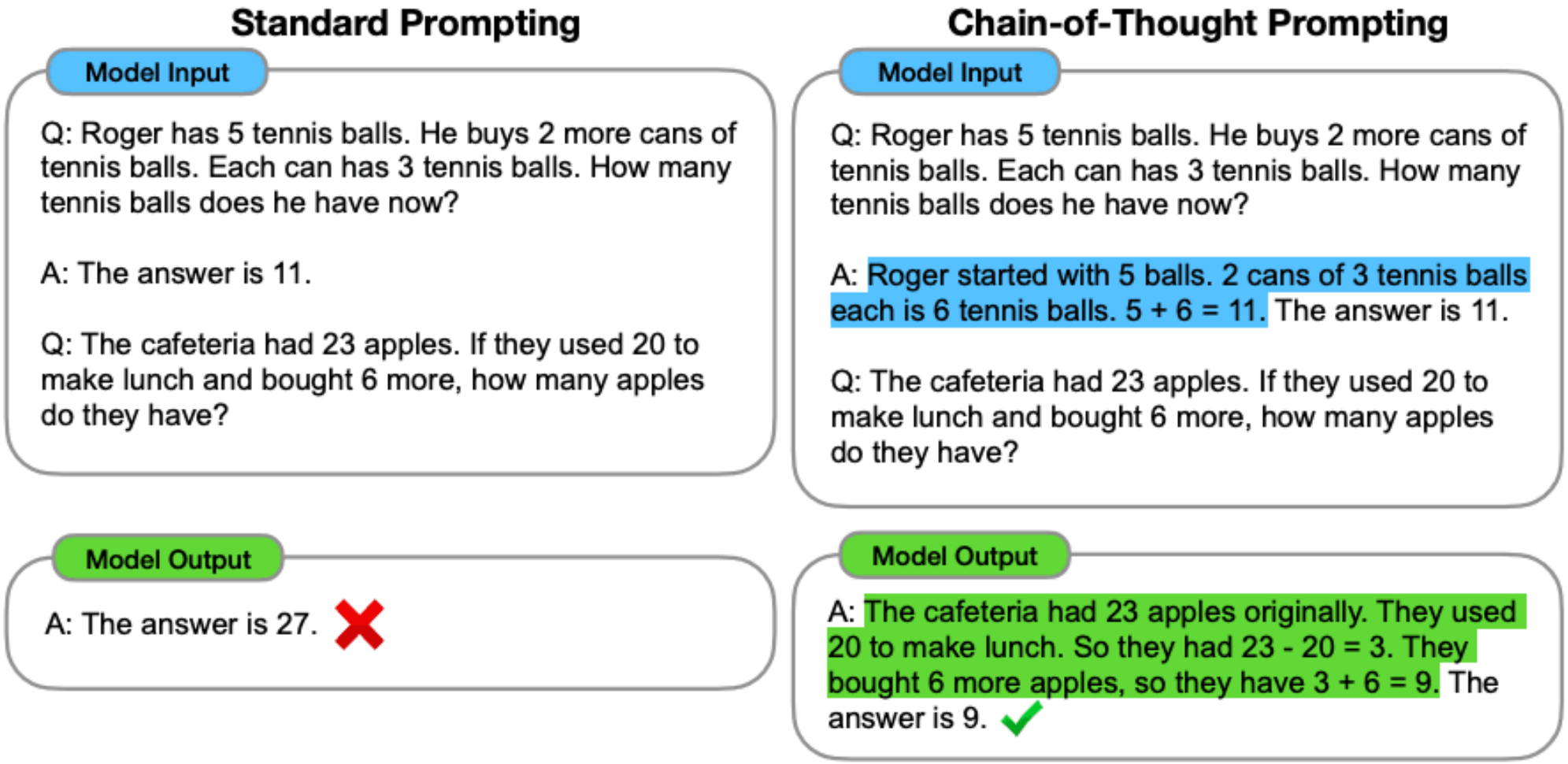
What are LLM agents?





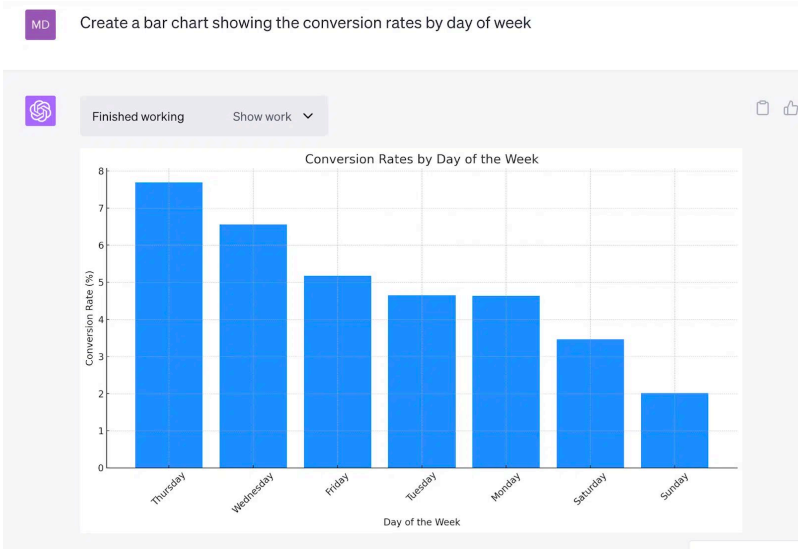
- Level 1: Text agent
  - Uses text action and observation
  - Examples: ELIZA, LSTM-DQN
- Level 2: LLM agent
  - Uses LLM to act
  - Examples: SayCan, Language Planner
- Level 3: Reasoning agent
  - Uses LLM to reason to act
  - Examples: ReAct, AutoGPT
  - **The key focus of the field and the talk**

LLM reasoning: recall of CoT



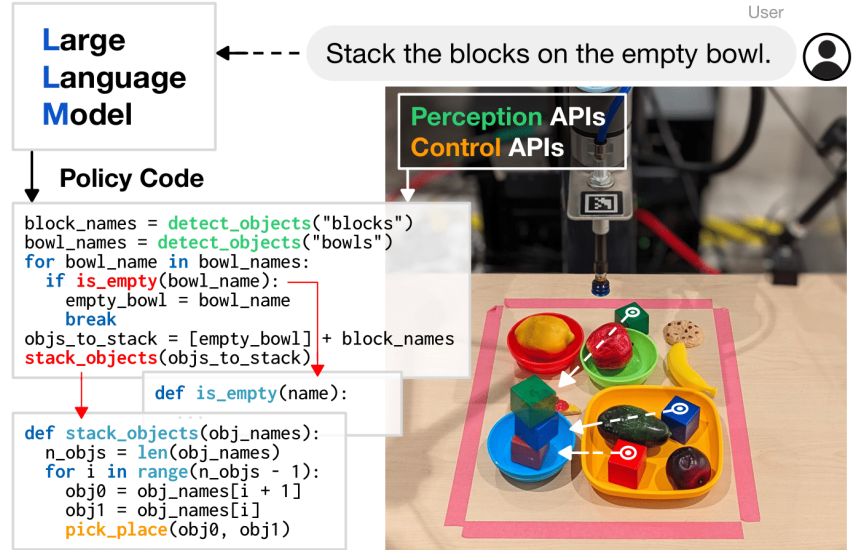
LLMs + tool use in perspective of executable language grounding

- Ground language models into executable actions
- Mapping natural language instructions into code or actions executable within various environments such as databases, web applications, and robotic physical world.
- LM (planning and reasoning) + actions



Data analysis

Web/Apps



Robotic physical world

LLMs + tool use in perspective of executable language grounding

- LLMs + tool use in executable language grounding tasks

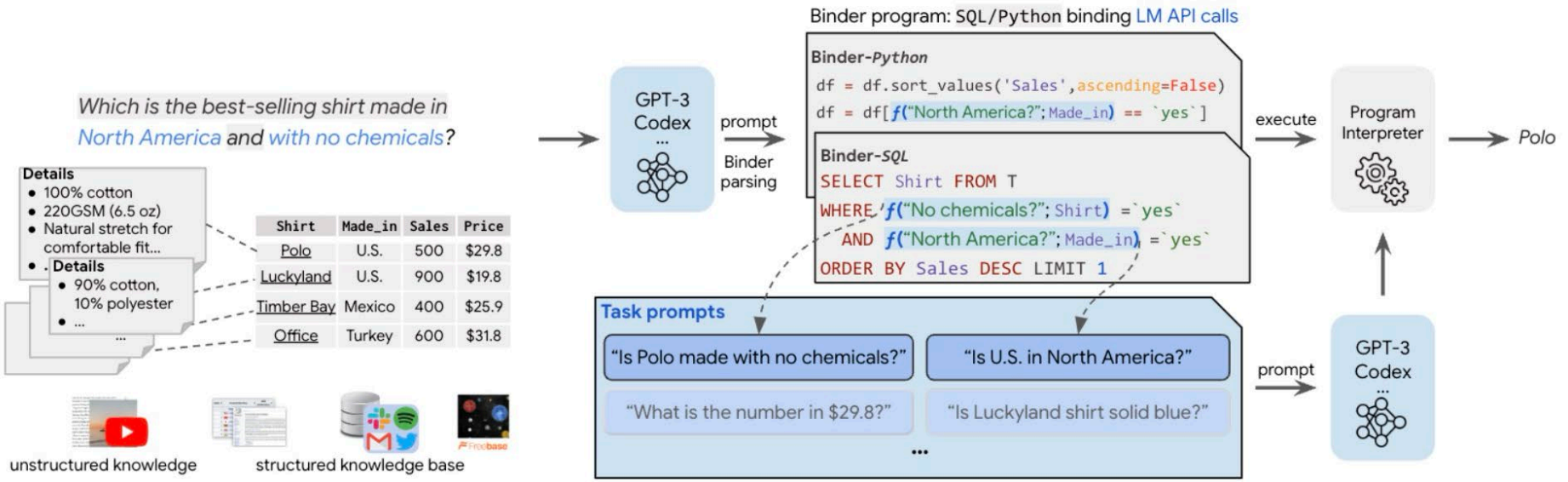
Inputs

- Language: user question/request
- Toolkit: code, APIs to search engines, self-defined functions, expert models…
- Environment: databases, IDE, web/apps, visual and robotic physical world…

Outputs

- Grounded reasoning code/action sequence that can be executed in the corresponding environment
  - What tools to select, when and how to use the selected tools

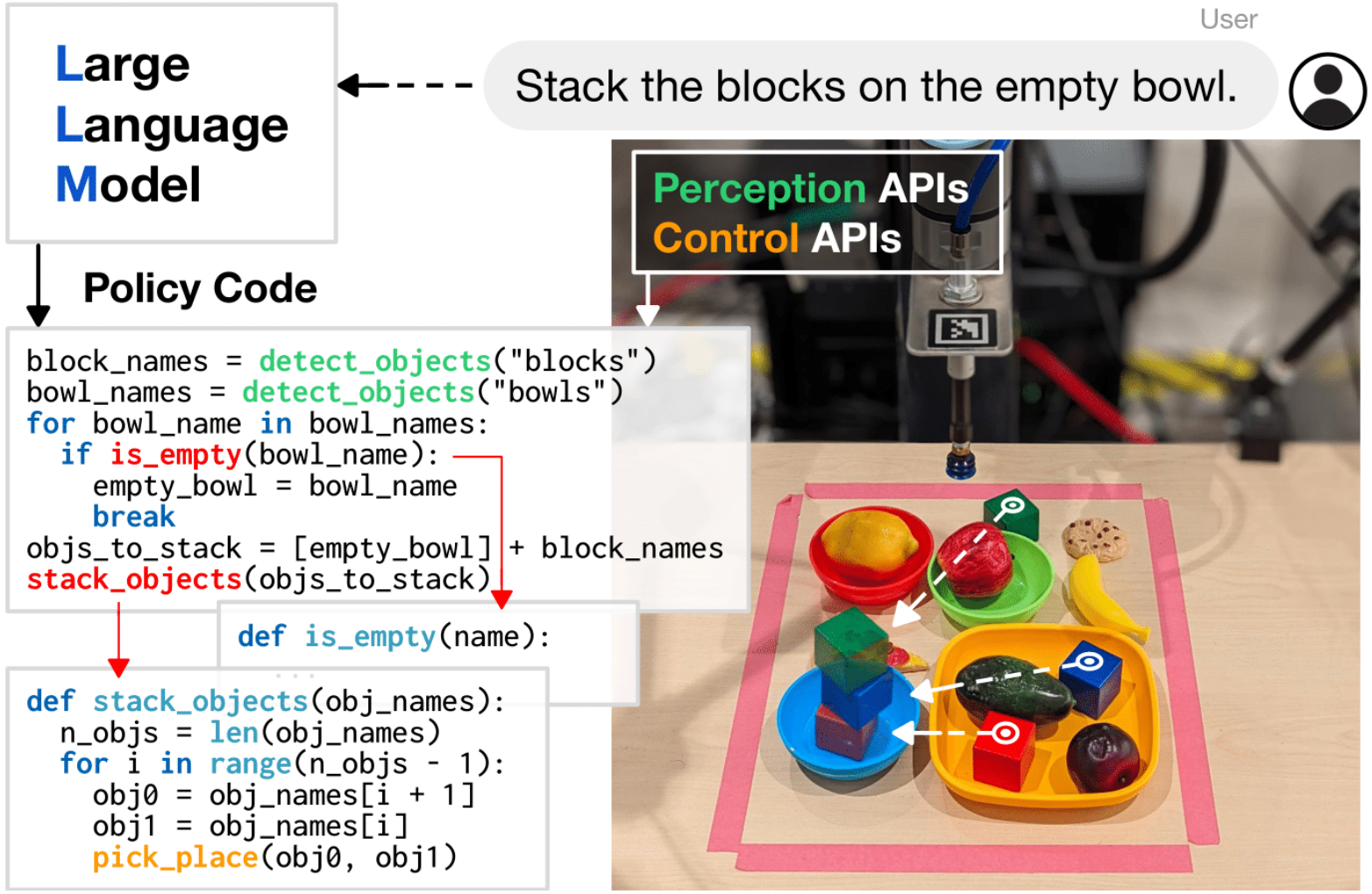
LLM + code and NLP expert function APIs



BINDER first prompts Codex to parse a question input into a BINDER program, in which Codex has to decide (1) which parts in the input can be converted to the target programming language (grey clause in figure), (2) the corresponding task API calls (blue clause in figure) to prompt Codex to resolve the other parts, and (3) where to insert the API calls in the BINDER program.
Next, BINDER prompts Codex again to generate answers to the task API calls (given the generated task prompts), integrates the generated results back to the programming language, and executes the resulting programming language expression to derive the final answer.

LLM + code, robotic arm, expert models: Code as Policies

Use LLM to write robot policy code given natural language commands.



https://code-as-policies.github.io/

LLM finetuning/pretraining for tool use: Toolformer

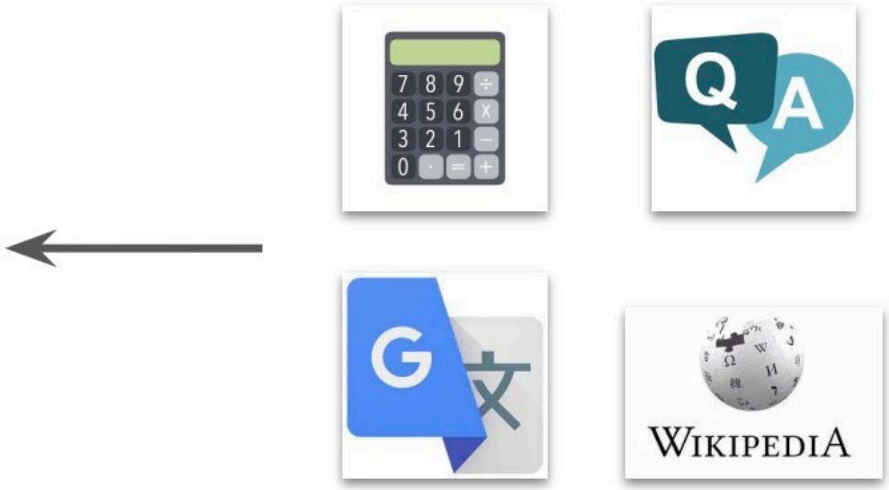
Toolformer is a model trained to decide which APIs to call, when to call them, what arguments to pass, and how to best incorporate the results into future token prediction. This is done in a self-supervised way, requiring nothing more than a handful of demonstrations for each API.

The New England Journal of Medicine is a registered trademark of **[QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society]** the MMS.

Out of 1400 participants, 400 (or **[Calculator(400 / 1400) → 0.29]** 29%) passed the test.

The name derives from "la tortuga", the Spanish word for **[MT("tortuga") → turtle]** turtle.

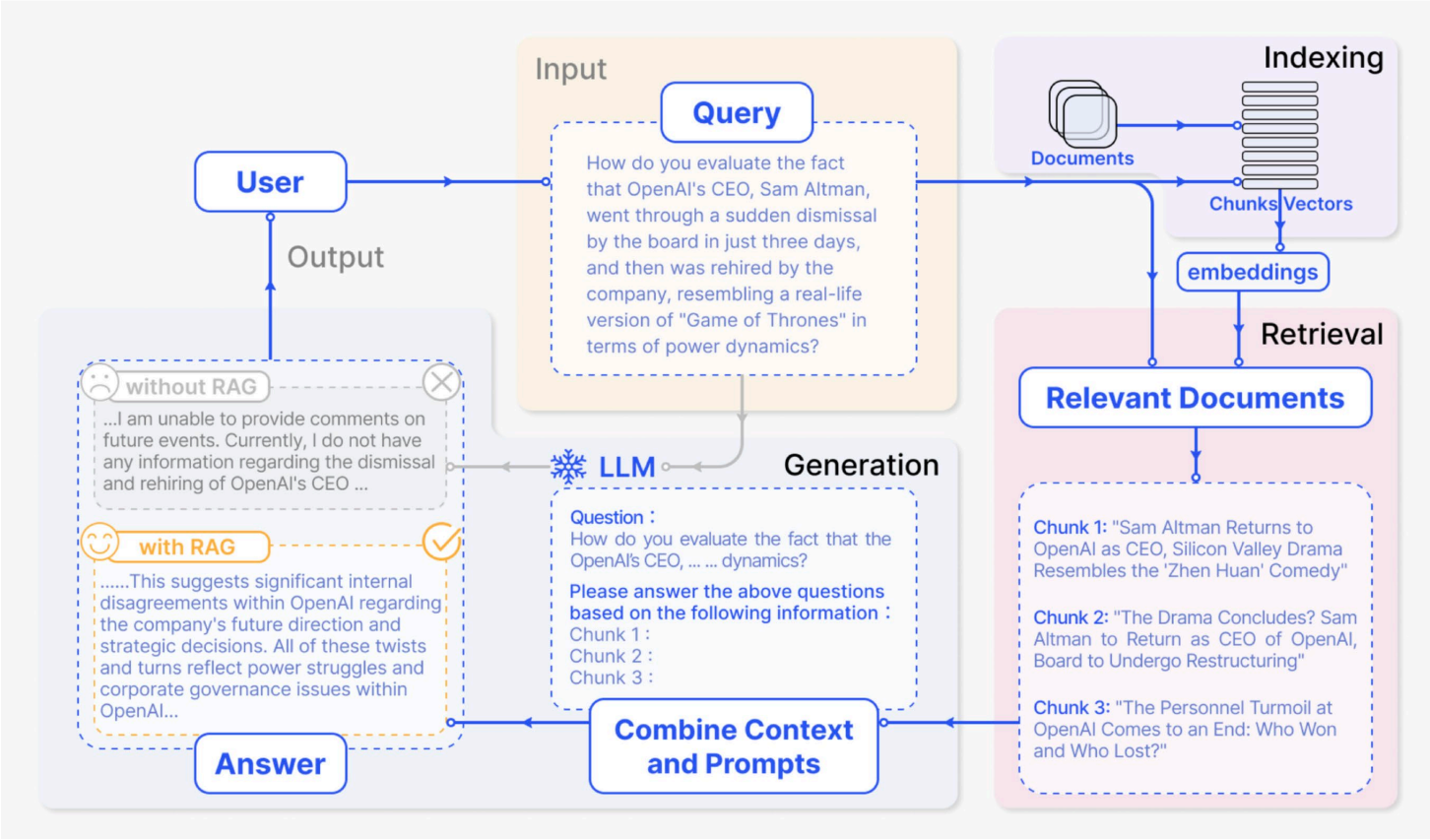
The Brown Act is California's law **[WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.]** that requires legislative bodies, like city councils, to hold their meetings open to the public.



Toolformer: Language Models Can Teach Themselves to Use Tools, Meta AI

Incorporating external knowledge-RAG

- Vector Database
  - Embedding
  - Indexing
  - Querying (Retrieve)
  - Post-process
- Generator
  - LLM (like GPT, DeepSeek)

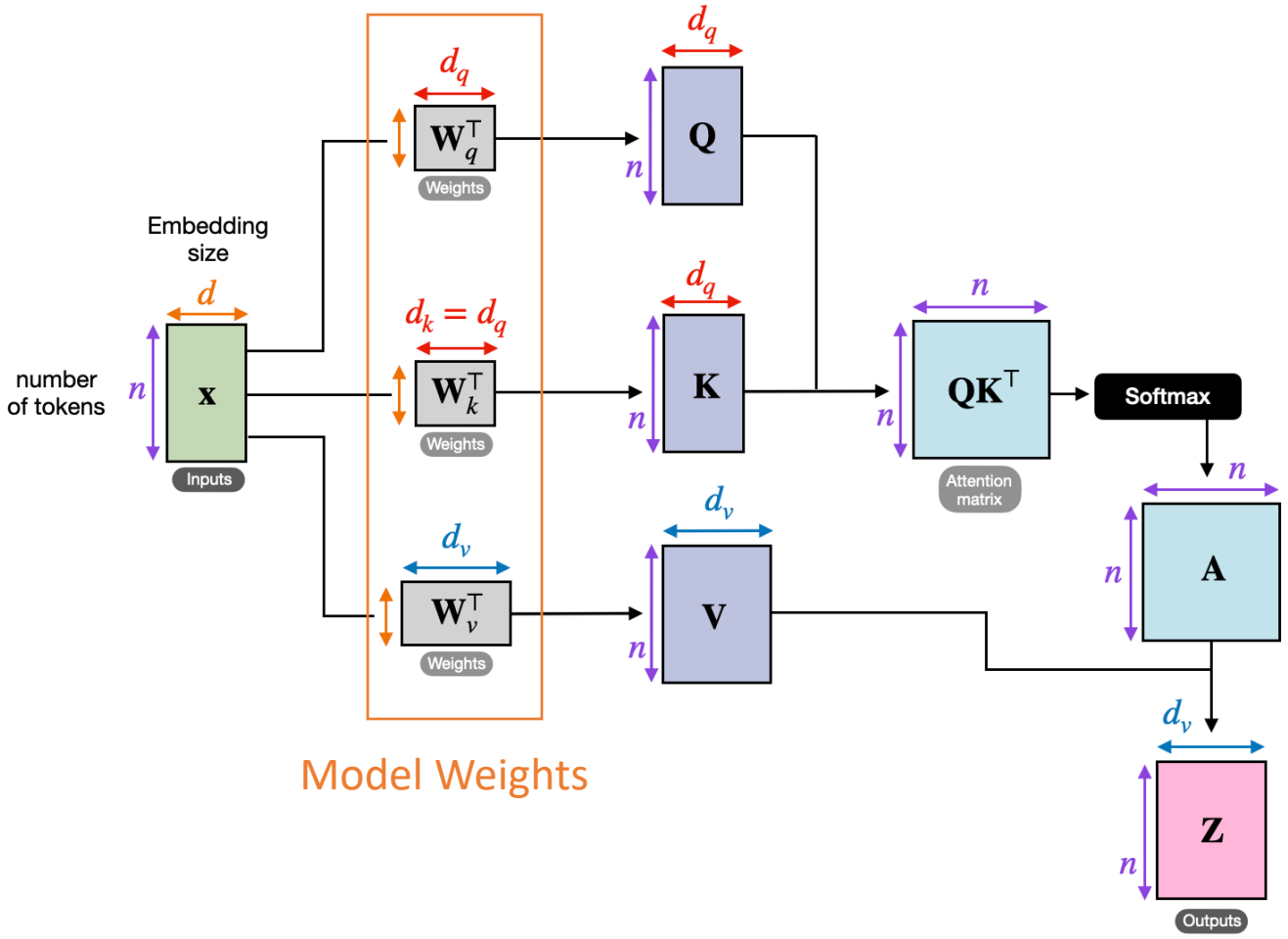


L10 Efficient training

- 1. Background
- 2. Model Parallelism
- 3. Data Parallelism
- 4. Parameter-Efficient Fine-Tuning

Background- Model Weights

- Weights are the parameters of the model that are learned during training.
- They are adjusted by the optimizer based on the gradients to minimize the loss function.
- In transformers, weights include the parameters in self-attention, position-wise feed-forward networks, and layer normalization.



Memory consumption

- Consider a model with Ψ parameters.
- During fp16 training, model parameters need 2Ψbytes to store the model weights, 2Ψbytes for the corresponding gradients.
- Adam's Optimizer States:
  - fp32 copy of parameters: 4Ψ
  - fp32 copy of momentum:4Ψ
  - fp32 copy of variance: 4Ψ
  - In total: $4\Psi + 4\Psi + 4\Psi = 12\Psi$
- Total consumption: $2\Psi + 2\Psi + 12\Psi = 16\Psi$

Background – Activation (intermediate results)

- The intermediate results stored from forward pass to perform backward pass.

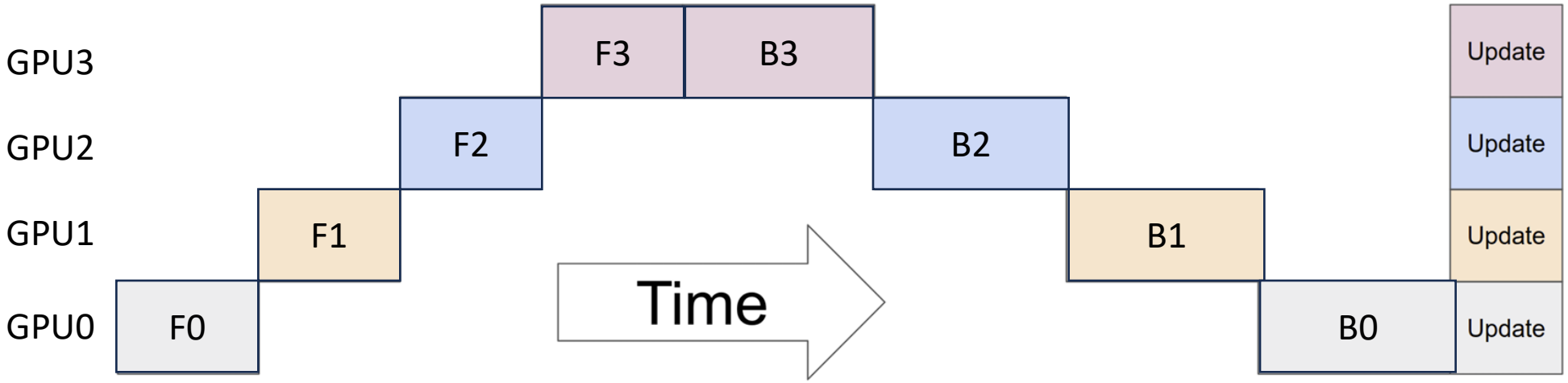
- The activation memory of a transformer-based model is proportional to the number of

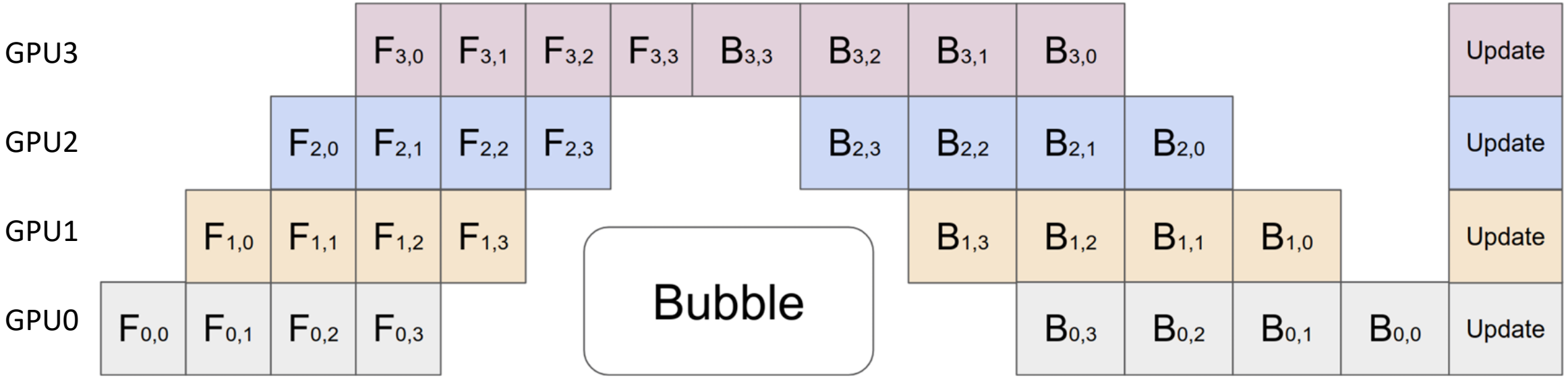
$$transformer\ layers \times\ hidden\ dimensions \times\ sequence\ length \times\ batch\ size.$$

- As a concrete example, the 1.5B parameter GPT-2 model trained with sequence length of 1K and batch size of 32 requires about 60GB of memory.

Naïve MP

- Advantage:
  - fit very large models onto limited hardware
- Disadvantage:
  - GPU idle (Bubbles).



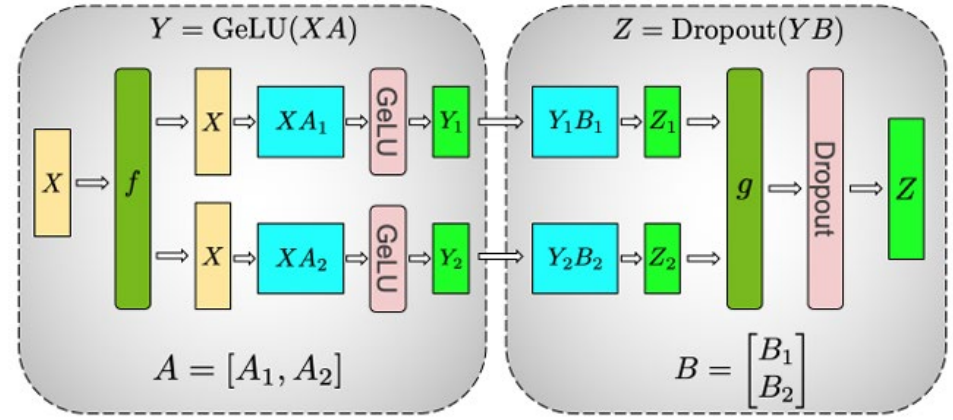


GPipe

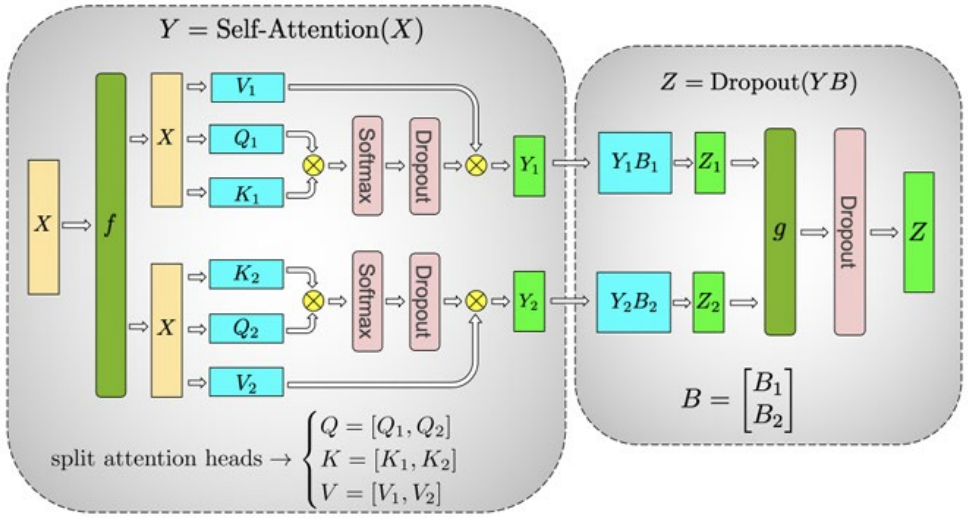
- Advantage:
  - Less bubble, make the most use of GPUs.
- Disadvantage:
  - Need to experiment to find the size of micro-batch that leads to the highest efficient utilization of the GPUs.

Modern MP – MegatronLM (Tensor Parallelism)

- In Tensor Parallelism each GPU processes only a slice of a tensor and only aggregates the full tensor for operations that require the whole thing.
- It parallize the tensor in column- and row-wise methods.



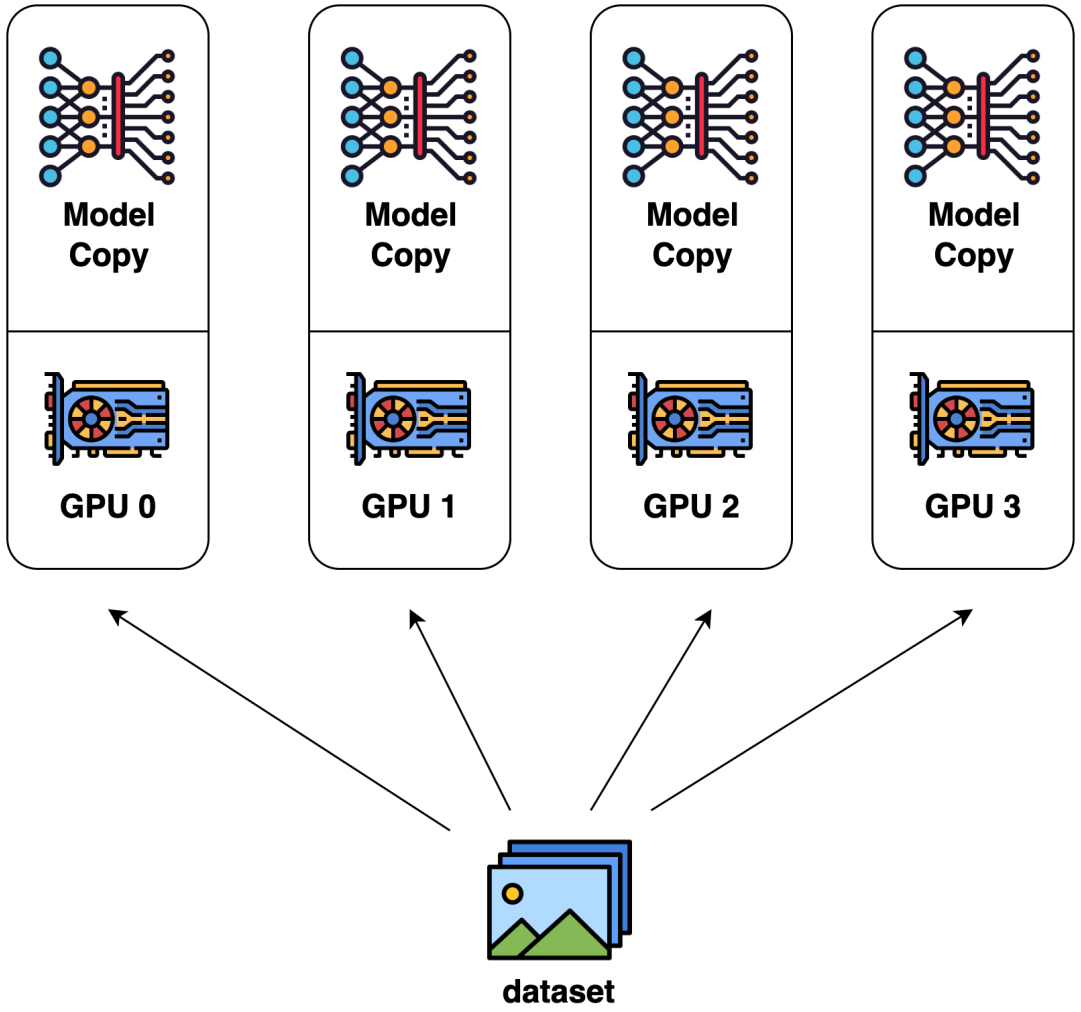
(a) MLP



(b) Self-Attention

Data parallelism (DP) – Naïve DP

- In data parallel training, the dataset is split into several shards, each shard is allocated to a device.
- Each device will hold a full copy of the model replica and trains on the dataset shard allocated.



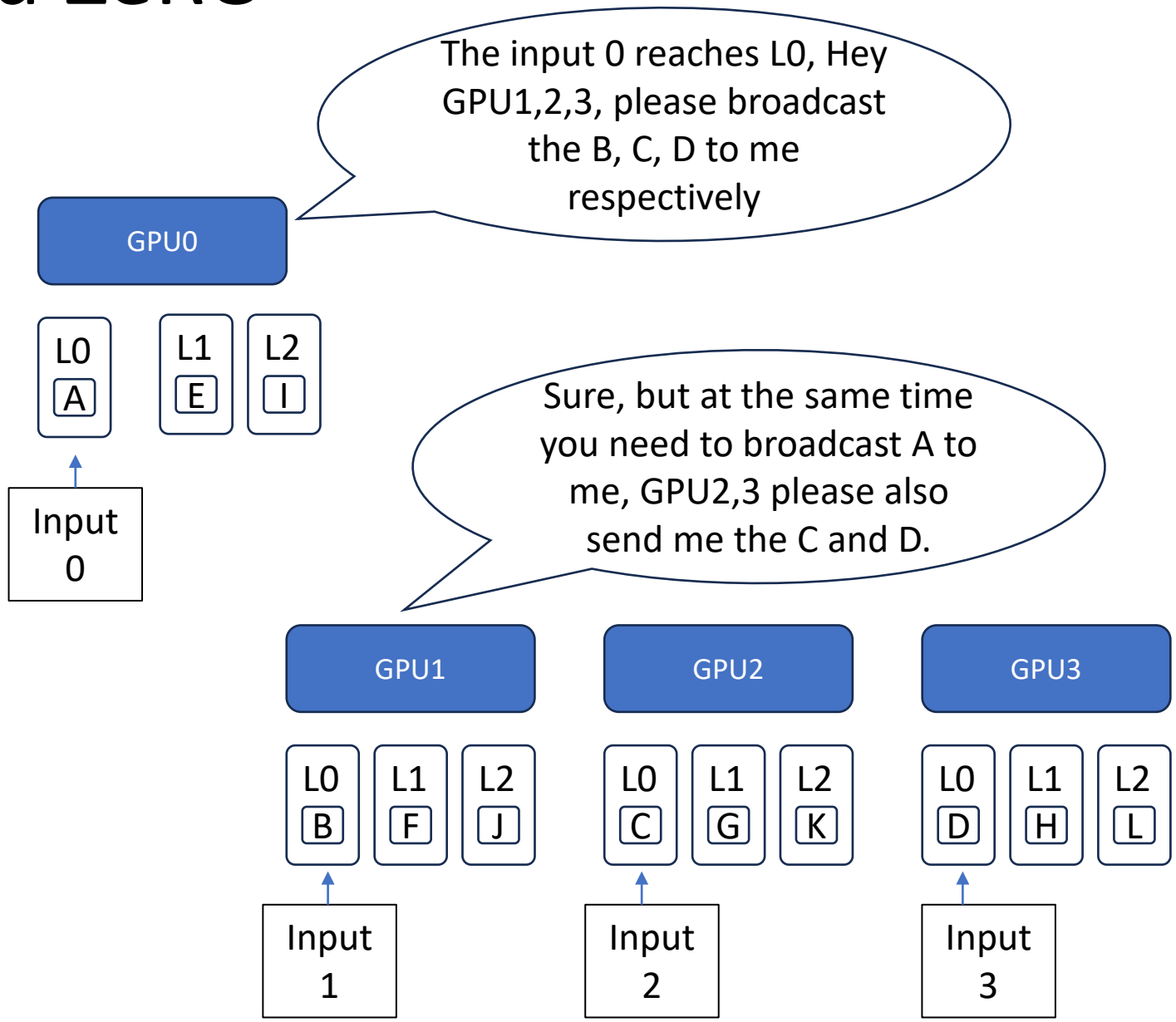
Naïve DP vs DDP

- DistributedDataParallel is multi-process and works for both single- and multi- machine training. (multiprocess-based)
- In DDP, each GPU consumes each own mini-batch of data directly
- During backward, once the local gradients are ready, they are then averaged across all processes.
- DDP is more “balance” than DP, which is also faster and memory-friendly.

DP vs DDP

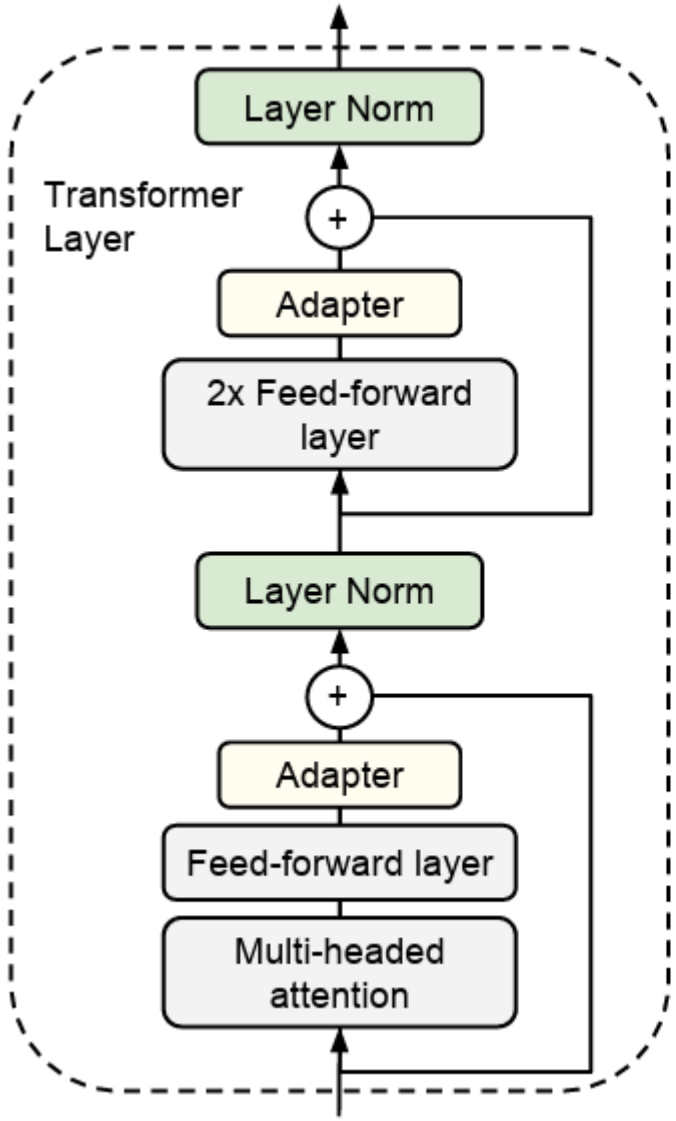
Solution: DeepSpeed ZeRO

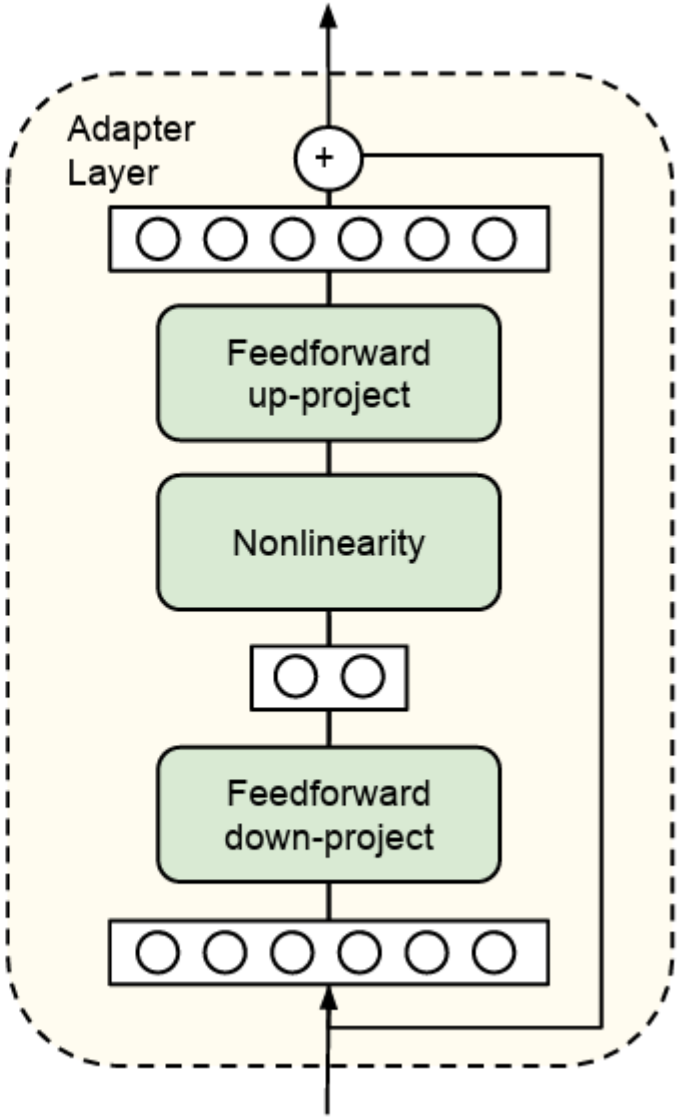
- However, different from the original DP, each GPU in DeepSpeed ZeRO only holds a small fraction of the model parameters instead of the replicated entire model.
- To finish the forward pass, each GPU need to ‘borrow’ parameters from other GPUs



Parameter-efficient fine-tuning (PEFT) - Adapter

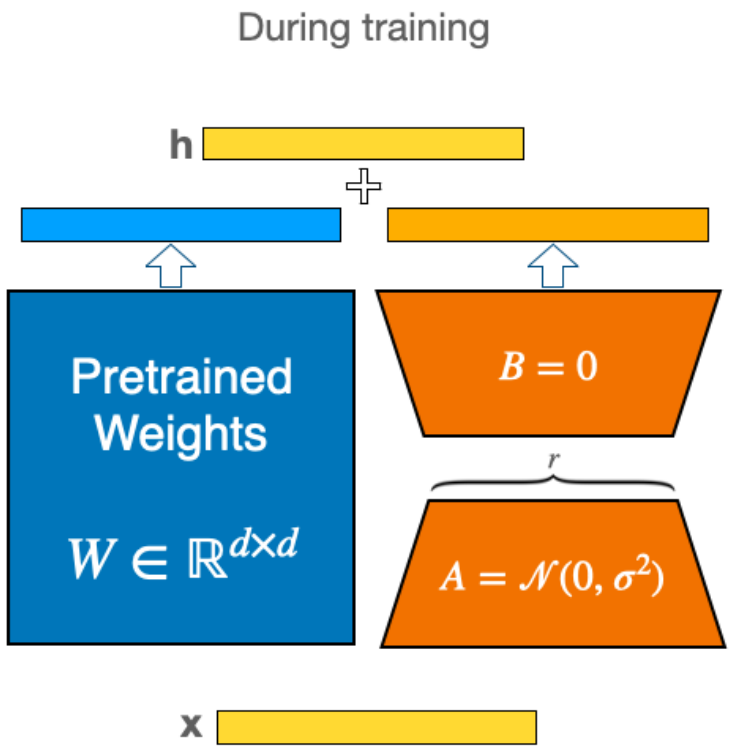
- Adapter-based methods add extra trainable parameters after the attention and fully-connected layers of a frozen pretrained model to reduce memory-usage and speed up training.
- The adapters are typically small but demonstrate comparable performance to a fully finetuned model and enable training larger models with fewer resources.





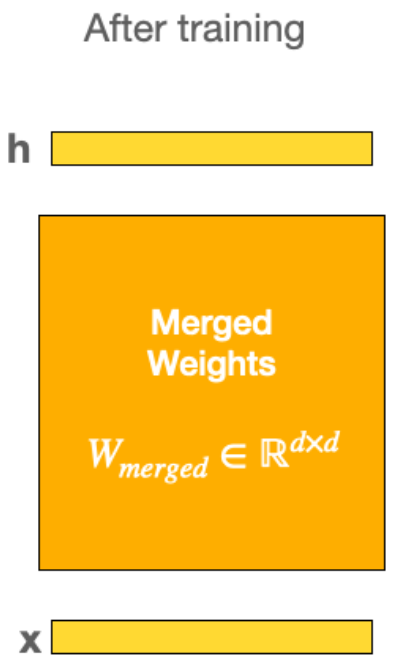
LoRA

- Different from the adapter, instead of sequentially added to the transformer layers, LoRA is added parallel to the pretrained weights.
- After training, LoRA weights can be merged into the model weights, so there is no extra inference latency.



$$h = Wx + BAx$$

$$h = \underbrace{(W + BA)}_{W_{merged}}x$$

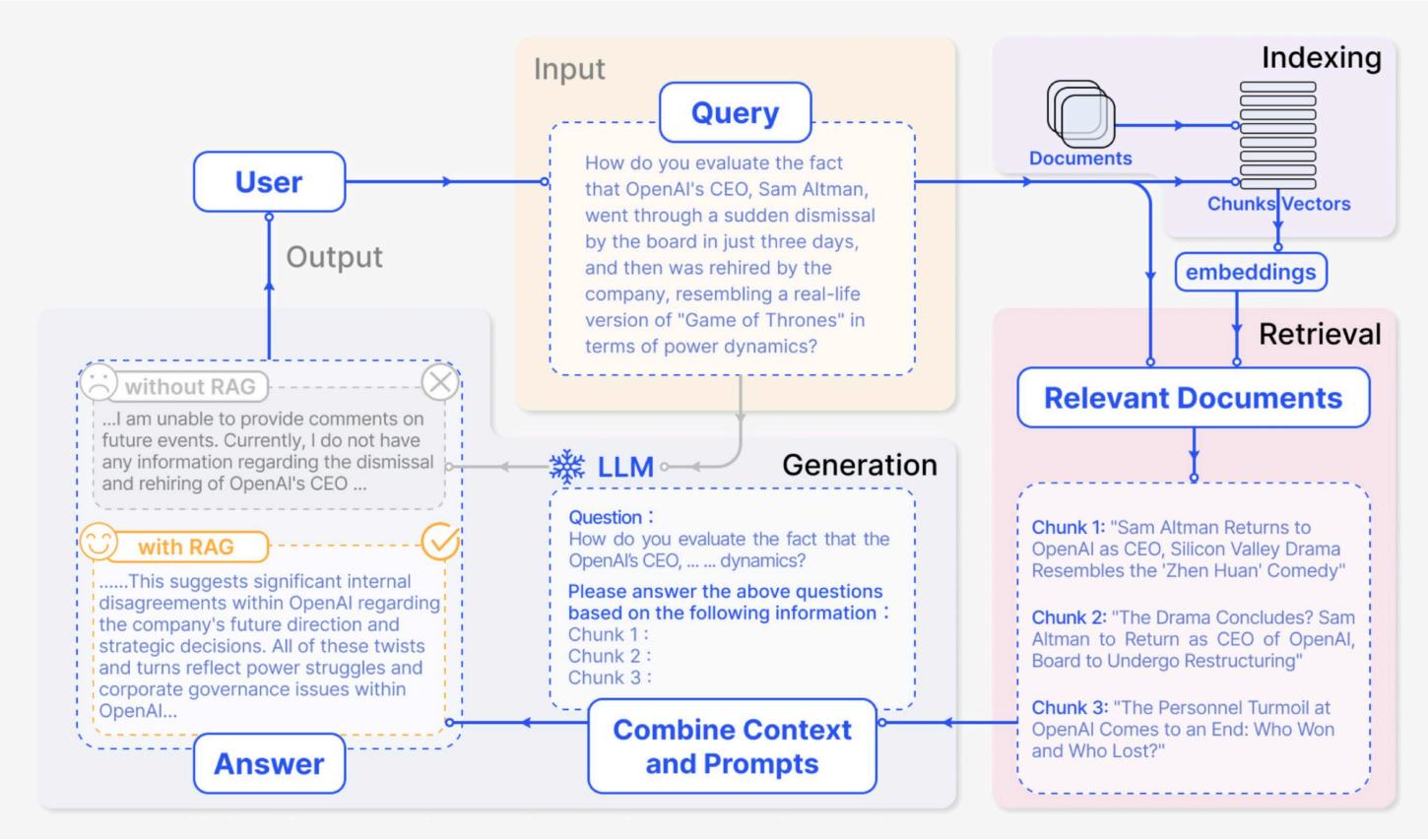


L11 RAG

- 1. What’s RAG?
- 2. What / When / How to retrieve?

RAG architecture

- Vector Database
  - Embedding
  - Indexing
  - Querying (Retrieve)
  - Post-process
- Generator
  - LLM (like GPT, BART, T5)

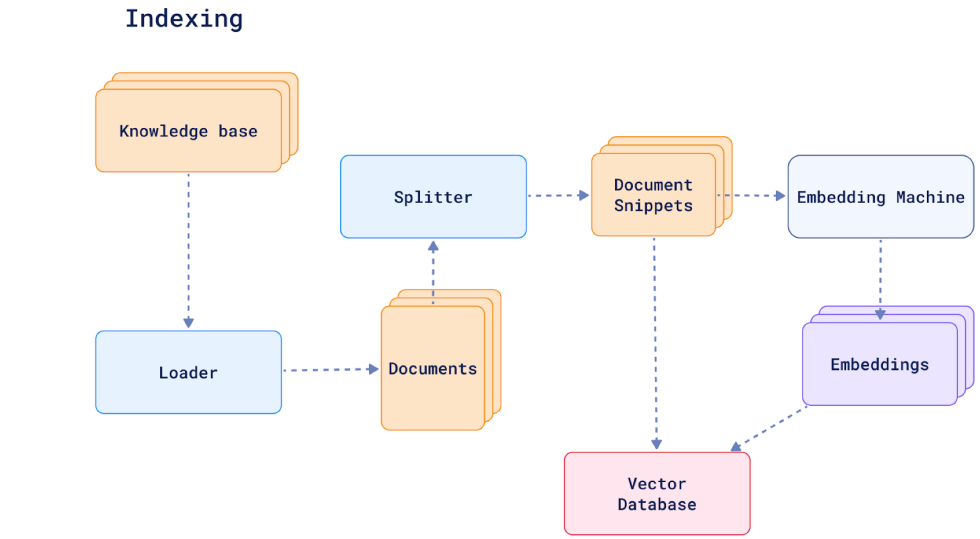


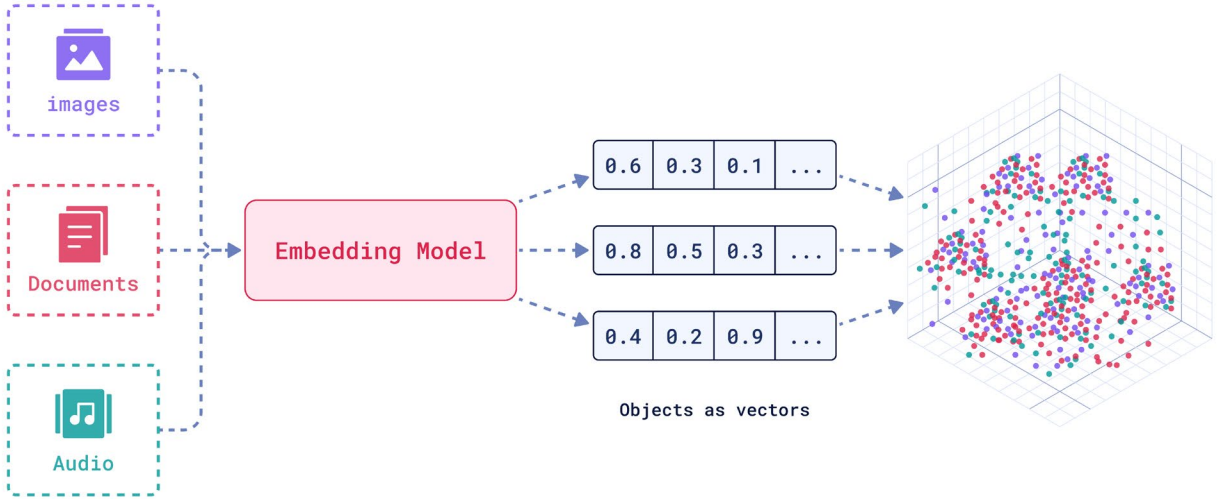
Vector database-embedding

Start with a loader that gathers documents containing your data. These documents could be anything from articles and books to web pages and social media posts.

Next, a splitter divides the documents into smaller chunks, typically sentences or paragraphs. This is because RAG models work better with smaller pieces of text. In the diagram, these are document snippets.

Each text chunk is then fed into an embedding machine (TF-IDF, BM25, BERT, GPT). This machine uses complex algorithms to convert the text into vector embeddings.

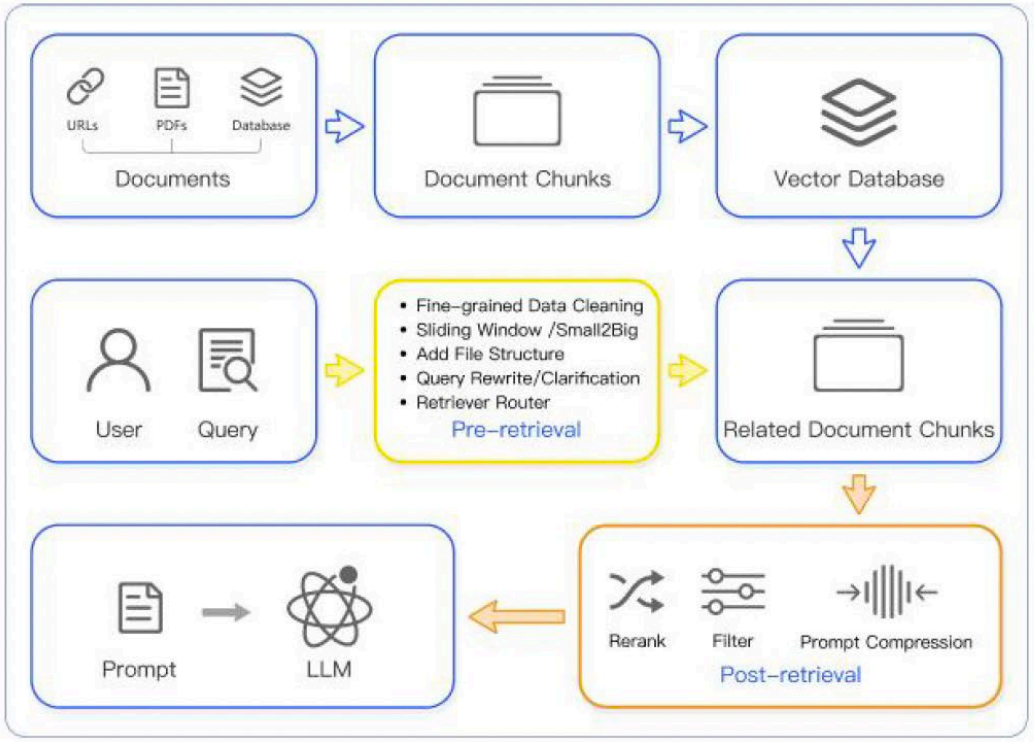




Advanced RAG

Index Optimization → Pre-Retrieval Process → Retrieval → Post-Retrieval Process→ Genaration

- **Optimizing Data Indexing:** sliding window, fine-grained segmentation、adding metadata
- **Pre-Retrieval Process:** retrieve routes, summaries, rewriting, and confidence judgment
- **Post-Retrieval Process:** reorder, filter content retrieval



Main issues in RAG – what/when/how

What to retrieve ?

- Token
- Phrase
- Chunk
- Paragraph
- Entity
- Knowledge graph

When to retrive ?

- Single search
- Each token
- Every N tokens
- Adaptive search

How to use the retrieved information ?

- Input/Data Layer
- Model/Intermediate Layer
- Output/Prediction Layer

Other Issues

Augmentation stage:
- Pre-training
- Fine-tuning
- Inference

Retrieval choice:
- BERT
- Roberta
- BGE
- .....

Model Collaboration

Scale selectionz

Generation choice:
- GPT
- Llama
- T5
- .....

What to retrieve?

coarse

Retrieval granularity

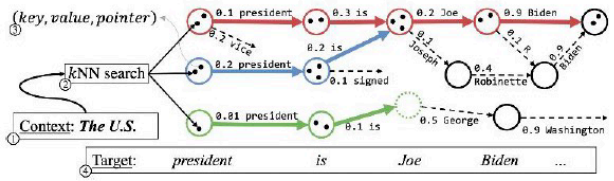
meticulous

low

level of structuration

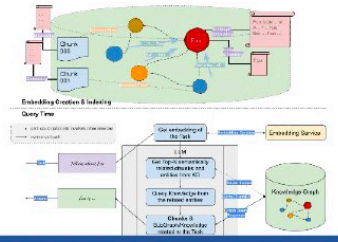
High

Chunk | In-Context RAG 2023



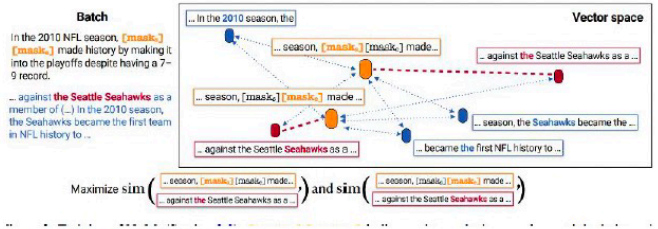
The search is broad, recalling a large amount of information, but with low accuracy, high coverage but includes much redundant information.

Knowledge Graph | 2023

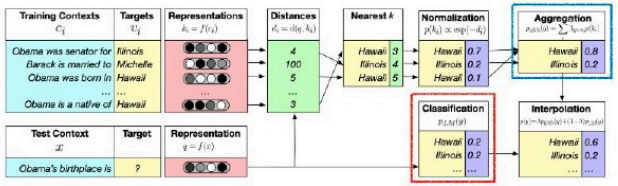


Richer semantic and structured information, but the retrieval efficiency is lower and is limited by the quality of KG.

Phrase | NPM 2023

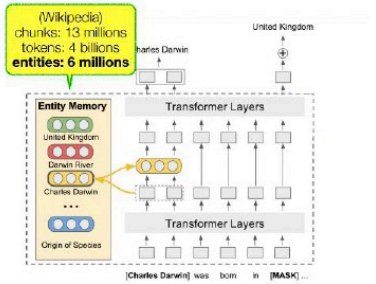


Token | KNN-LMM 2019



It excels in handling long-tail and cross-domain issues with high computational efficiency, but it requires significant storage.

Entity | EasE 2022



When to retrieve?

High efficiency, but low relevance of the retrieved documents

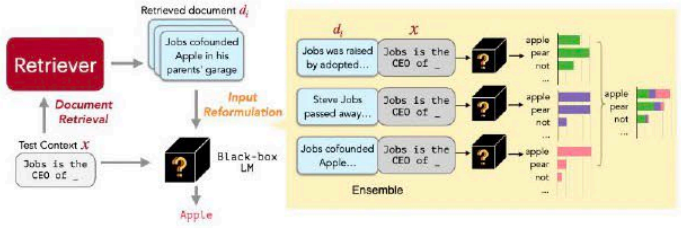
Balancing efficiency and information might not yield the optimal solution

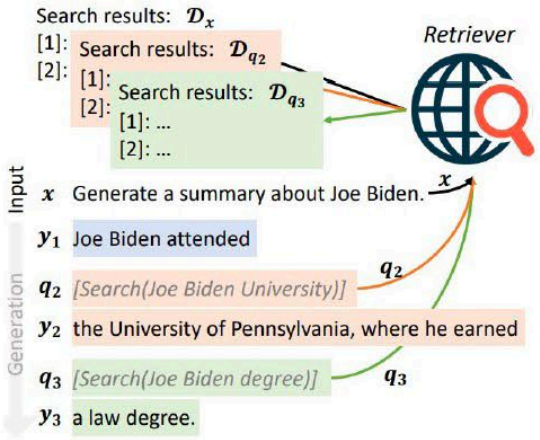
A large amount of information with low efficiency and redundant information.

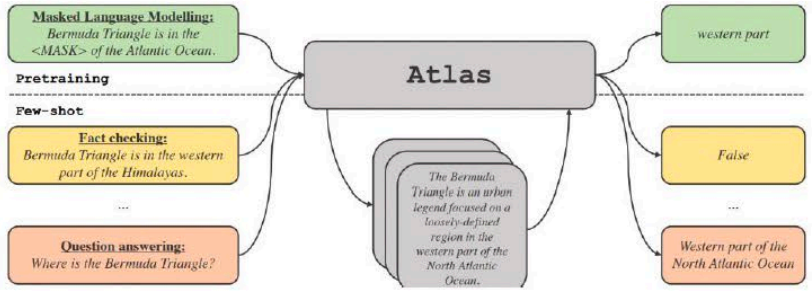
Once | Replug 2023

Adaptive | Flare 2023

Every N Tokens | Atlas 2023







Conducting once search during the reasoning process.

Adaptively conduct the search.

Retrieve once for every N tokens generated.

Low

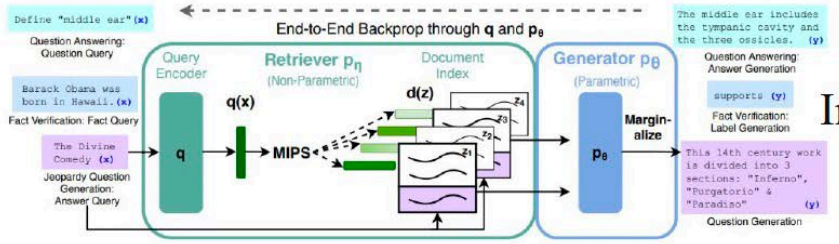
Retrieval frequency

High

How to use?

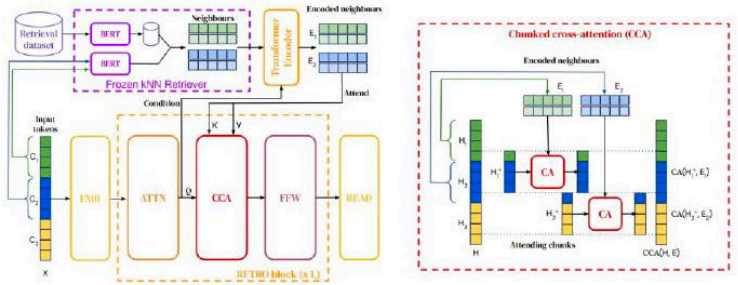
Integrating the retrieved information into different layers of the generation model,during inference process.

Integrate retrieval positions.



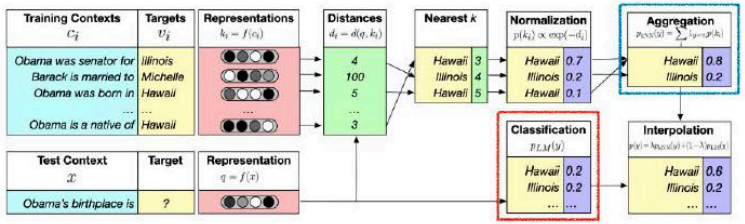
Input / Date layer

Using simple, but unable to support the retrieval of more knowledge blocks, and the optimization space is limited.



Model / Interlayer

Supports the retrieval of more knowledge blocks, but introduces additional complexity and must be trained.



Output /Prediction layer

Ensuring the output results are highly relevant to the retrieval content, but the efficiency is low.

Learning outcome questionnaire (LOQ)

Final exam

Assessment

- Continuous assessment (60%)
  - 2 individual homework assignments (each 15%)
  - 1 group project with presentations (30%)
- Final exam (40%)

Final exam (1)

- It will be a closed-book exam. No materials and electronics are allowed; no calculators.

- The scope of the exam includes all lectures, tutorials, and homework assignments.
- Less focus will be put on memorization. The key focus will be put on understanding basic concepts and principles, understanding fundamentals techniques, analyzing advantages and disadvantages of different techniques, and proposing solutions for specific problems.

Final exam (2)

- There will be in total 5 questions.
  - Short questions to ask your understanding of some concepts, principles, reasons, key features of some techniques, etc. (around 50%)
  - A little bit design, calculation, program codes analysis, etc. (around 50%)

- We will hold an office hour via Zoom (to be announced later) to answer questions approximately one week before the final exam. Welcome to join if you have questions.

Q&A