Architecture of ML Systems (AMLS)
07 LLM Training and Inference

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: Jun 05, 2025

Announcements / Org

- **#1 Hybrid & Video Recording**
  - Hybrid lectures (in-person, zoom) with optional attendance
    https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09
  - Zoom **video recordings**, links from website
    https://mboehm7.github.io/teaching/ss25_amls/index.htm

- **#2 Projects & Exercises**
  - **Submission deadline: Jul 15 EOD**
  - Get started, use the office hour (Tue 4pm-5.30), and mentor meetings

~3 ECTS → 90h

Categories of Execution Strategies

Batch
SIMD/ SPMD

Independent Tasks
MIMD

Mini-batch, Federated, RL

Self-supervised, supervised, RL

05_a Data-Parallel Execution

05_b Task-Parallel Execution

06 Parameter Servers
(data, model)

07 LLM Training
(dense, MoE)

08 Hybrid Execution and HW Accelerators

---

09 Caching, Partitioning, Indexing, and Compression

Agenda

- Basics and Terminology
- LLM Data Preprocessing
- LLM Pre-Training
- LLM Post-Training and Fine-Tuning
- LLM Inference

Basics and Terminology

Large Language Models (LLMs) Overview

- **Tokens/Words x_i**
  - Sequence of characters, n-grams, or words; **dictionary defined upfront**
  - The larger the tokens, the more efficient but the less effective

```
(AMLS, is, an, easy, course)
```

- **LLMs as Generate AI**
  - **Probability distribution** $p(x_1, \ldots, x_L)$
  - Generative models $X_{1:L} \sim p(x_1, \ldots, x_L)$

- **Autoregressive LLMs**
  - Sequential product of conditional token probabilities (by chain rule)
  - $p(x_1, \ldots, x_L) = p(x_1) * p(x_2|x_1) * p(x_3|x_2,x_1) * \ldots$

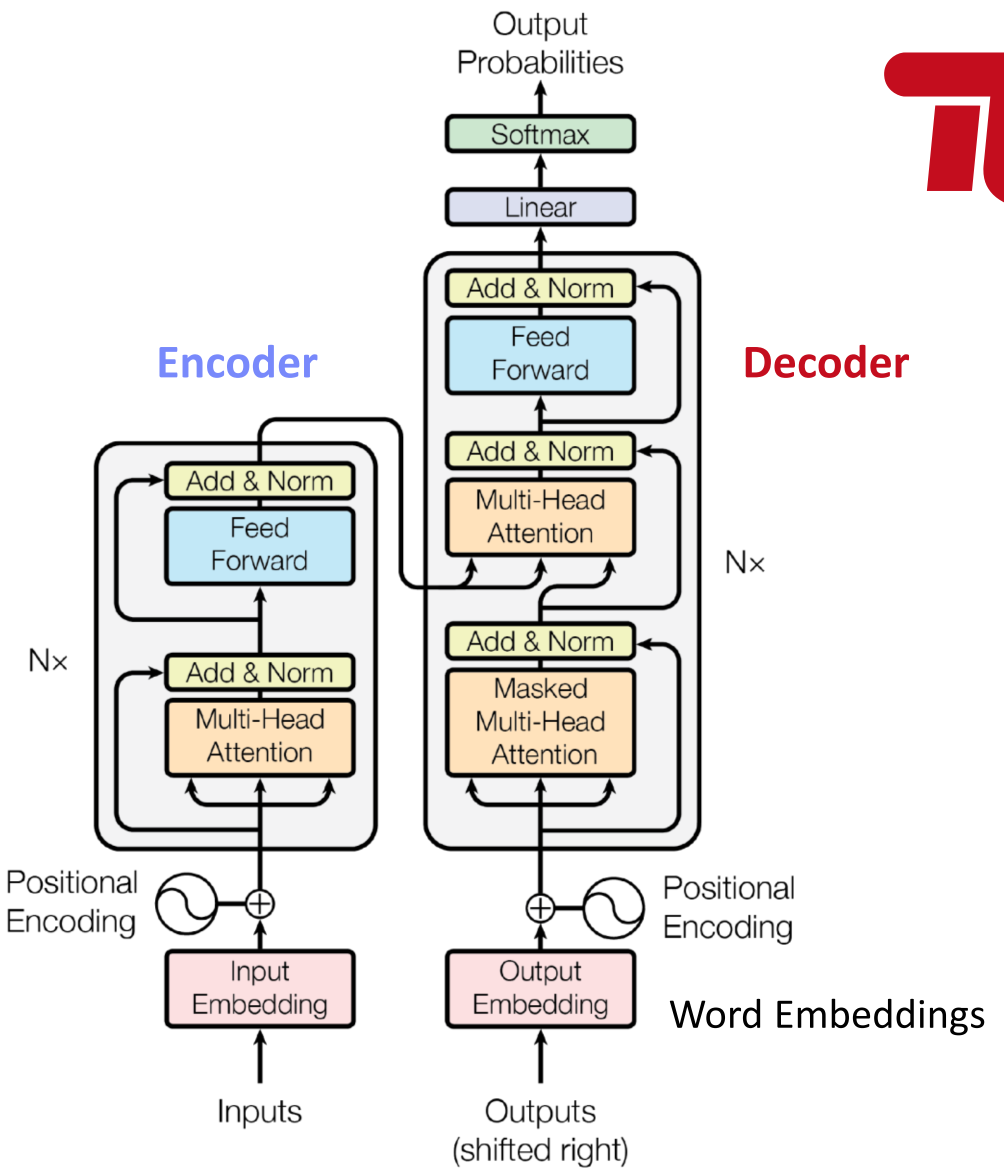
```
p(AMLS)
* p(is|AMLS)
* p(an|AMLS,is)
* p(easy|AMLS,is,an)
* p(course|AMLS,is,an,easy)
= 0.7
```

Transformer Model Architecture

- **Encoder-Decoder Structure**
  - **Encoder:** maps sequence of symbols (x_1, …, x_n) to continuous representation (z_1, …, z_n)
  - **Decoder:** maps sequence (z_1, …, z_n) sequentially to output symbol sequence (y_1, …, y_n)

- **Additional Features**
  - Stacked self-attention and fully-connected layers
  - **Residual connections** and layer normalization
  - **Masked self-attention** in decoder to ensure model uses only tokens before position i

[Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention is All you Need. **NeurIPS 2017**]

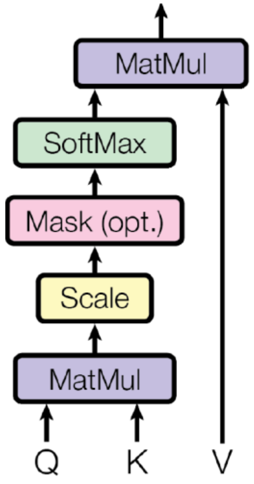


Transformer Model Architecture, cont.

[Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention is All you Need. **NeurIPS 2017**]

- **Basic Attention Block (Self-Attention)**
  - Mapping of a queries **Q** (search terms) and keys **K** (context words) / values **V** (modification of query) pairs to output
  - **Scaled dot-product attention** (divided by sqrt(d_k))

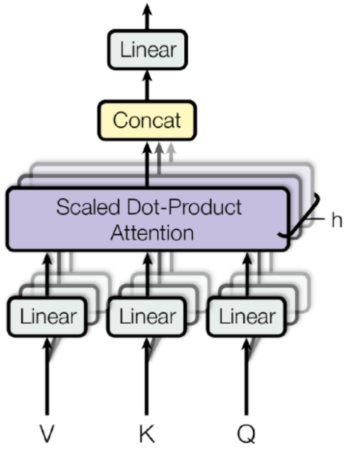
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



size d_k size d_v

- **Multi-head Attention Block**
  - **Project queries, keys, values h times** with different linear projections
  - Attend to information from different subspaces

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



- **Cross Attention**
  - Two sequences (in **Decoder**)
    - Key and value vectors from input sequence
    - Query vectors from output sequence

Dense vs Mixture of Experts (MoE) Models

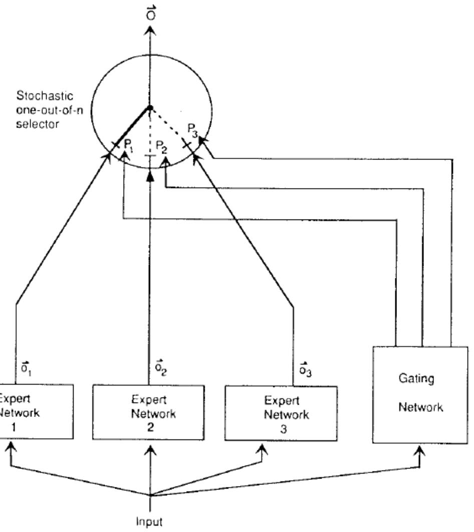
- **Dense Models**
  - Stack of transformer layers (e.g., 16 - 128), **all parameters active during inference**
  - E.g., GPT-2 and GPT-3 (sparse transformer)

[Tom B. Brown et al: Language Models are Few-Shot Learners. CoRR abs/2005.14165 (2020), GPT-3]

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

- **Mixture of Experts**
  - Ensemble of specialized models (experts)
  - Router / **gating network selects top-K experts**
  - E.g., DeepSeek-V3 (8+1 out of 256), [GPT-4]

[DeepSeek-AI: DeepSeek-V3 Technical Report, **2025**, https://arxiv.org/pdf/2412.19437]



[Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, Geoffrey E. Hinton: Adaptive Mixtures of Local Experts. **Neural Comput. 3(1), 1991**]

[Michael I. Jordan, Robert A. Jacobs: Hierarchies of Adaptive Experts. **NeurIPS 1991**]

LLM Training and Inference Workflow

Data Collection	Data Pre-processing (tokenize)	Pre-Training (multiple phases)	Post-Training / Fine-Tuning	LLM Inference
Crawl the Internet	Train Tokenizer (e.g., common subsequences)	Train LLM on very large data corpus	Fine-tune model on high-quality and/or user data	Test-time compute

LLM Data Preprocessing

Stanford ENGINEERING

[Stanford CS229 Machine Learning, Yann Dubois:
Building Large Language Models (LLMs),
https://www.youtube.com/watch?v=9vM4p9NN0Ts]

Data Sources

- **#1 Web Crawling**
  - Download websites from public internet
  - Common crawl **250 billion websites**, 2-5 billion new / month
  - Other sources: Project Gutenberg (books), Stack Exchange (code), Wikipedia, Newspapers (see **law suites**), LeetCode (code)
- **#2 Text Extraction**
  - From HTML and PDF (remove meta data)
- **#3 Filtering**
  - Remove undesirable content (e.g., outliers, backlist, low quality)
  - Remove duplicates (within and across documents)
- **#4 Composition**
  - Mix categories of data (e.g., source code, entertainment, text books)
  - Synthetic and multi-modal data

COMMON CRAWL

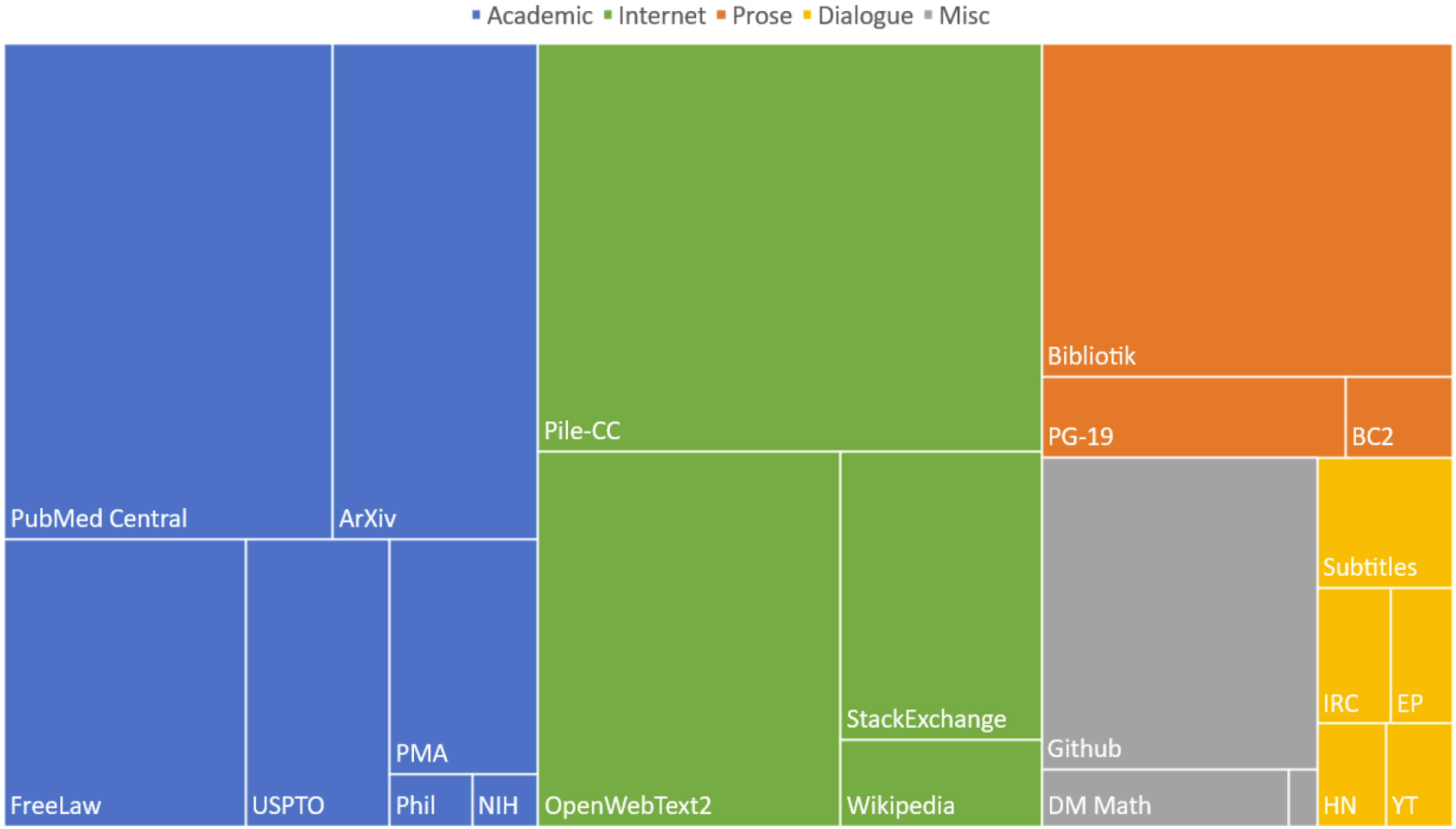
[https://commoncrawl.org/]

[Dong Huang, Yuhao Qing, Weiyi Shang, Heming Cui, Jie Zhang: EffiBench: Benchmarking the Efficiency of Automatically Generated Code. **NeurIPS 2024**]

~15T Tokens
(usually 1 epoch)

Data Sources, cont.

- **Composition of the Pile by Category**



[Leo Gao et al: The Pile: An 800GB Dataset of Diverse Text for Language Modeling. CoRR abs/2101.00027, 2021, https://arxiv.org/pdf/2101.00027]

Tokenization

- **Purpose**
  - Split input strings into sequence of tokens
  - Robustness to typos and different languages (more general than words)

`(AML, Sis, ane, asy, cou, rse)`

- **Common Approach**
  - Often **Token := 3-4 characters** (n-grams), sometimes longer words or individual characters
  - **Special handling of numbers** (single digit vs full number) **and source code** (handling indentation)

- **Byte-Pair Encoding (BPE)**
  - Large text corpus, start with single characters
  - **Merge common pairs** of tokens into token
  - **Repeat** until num merges / dictionary size

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

[Rico Sennrich, Barry Haddow, Alexandra Birch: Neural Machine Translation of Rare Words with Subword Units. **ACL 2016**]

[Taku Kudo, John Richardson: SentencePiece: A simple and language independent subword tokenizer... **EMNLP 2018** (Gemini)]]

Positional Embeddings

[https://cyrilzakka.github.io/llm-playbook/nested/fixed-pos-embed.html]

- **Challenge**
  - Transformers/self-attention by themselves **invariant to permutations** of input sequences
  - Positional embeddings combine **word embeddings and token positions**
- **Fixed Positional Embeddings**
  - Absolute positional encoding, which requires fixed sequence length or scaling; token pos, embedding index i, embedding dim d
  - Add positional embedding to word embedding of input token
  - **Example: Sinusoidal Positional Encoding**

even i: $$PE_{(\mathrm{pos},\ 2\mathrm{i})} = \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right)$$

odd i: $$PE_{(\mathrm{pos},\ 2\mathrm{i}+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right)$$

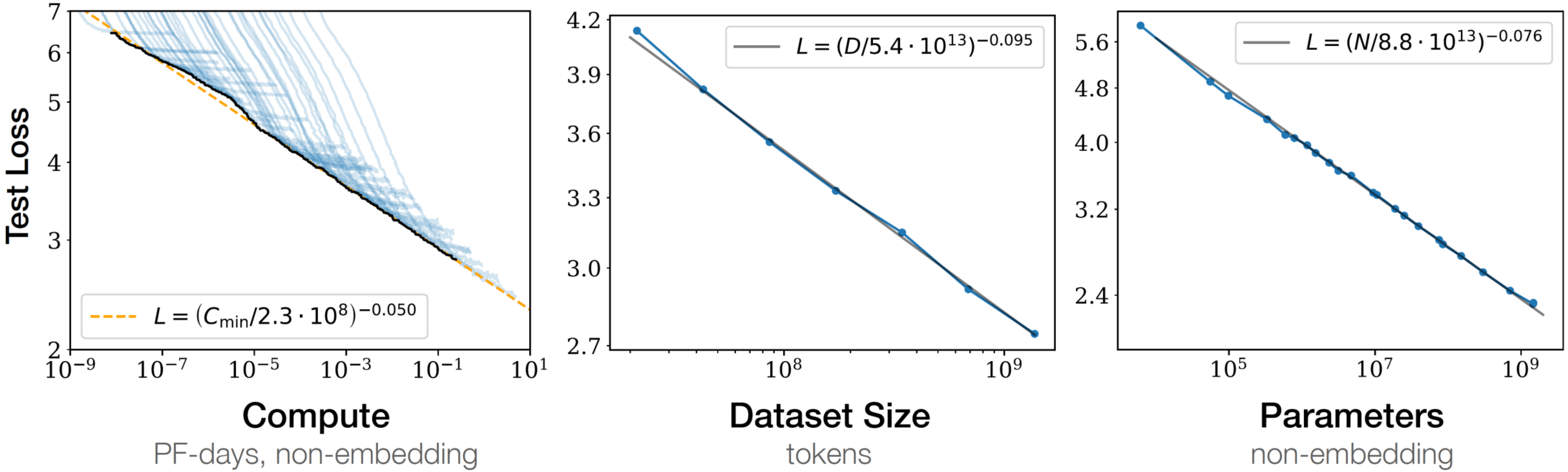
[https://medium.com/autonomous-agents/math-behind-positional-embeddings-in-transformer-models-921db18b0c28]

- **Rotary Positional Embedding (RoPE)**
  - Relative positional encoding
  - **Rotate queries and keys** with rotation relative to position in sequence

LLM Pre-Training

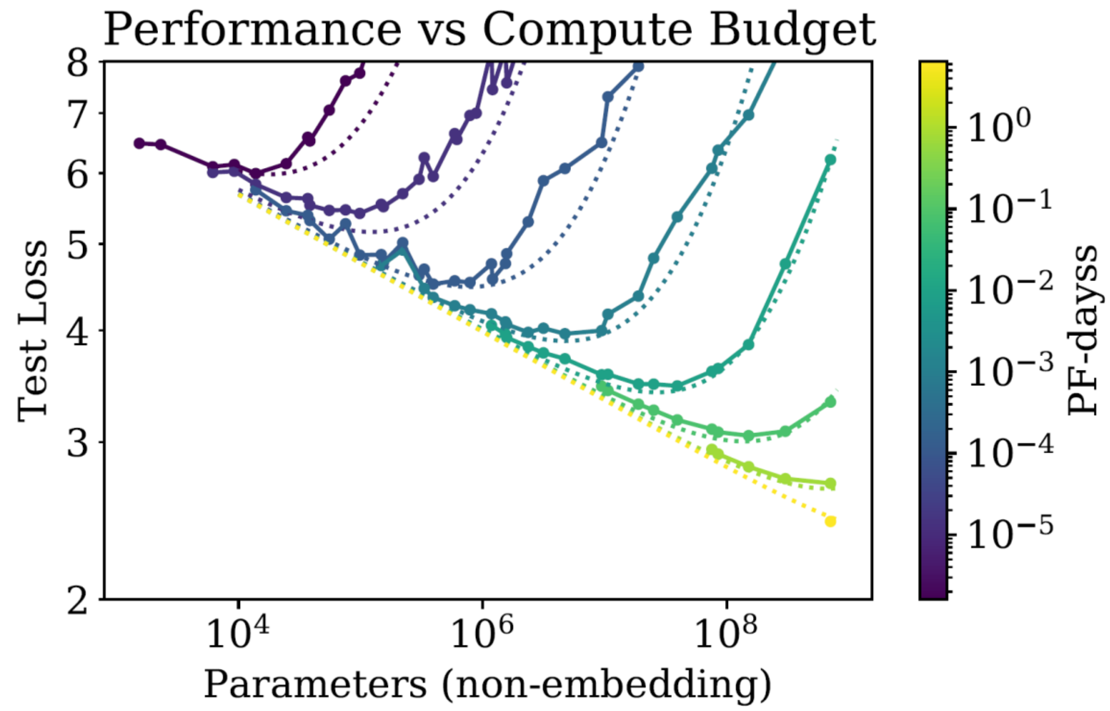
Scaling Laws

- **Predictable Performance w/ increasing Compute / Data Size / #Parameters**

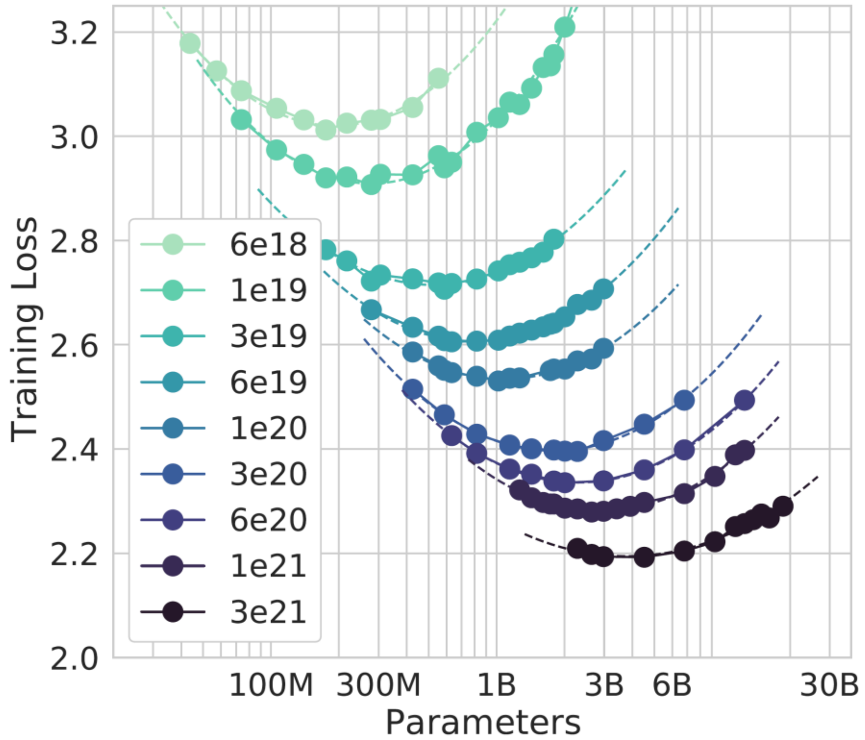


[Jared Kaplan et al: Scaling Laws for Neural Language Models. CoRR abs/2001.08361, **2020**, https://arxiv.org/pdf/2001.08361]

Scaling Laws, cont.



IsoFLOP Curves



[Jared Kaplan et al: Scaling Laws for Neural Language Models. CoRR abs/2001.08361, **2020**]

OpenAI

[Jordan Hoffmann et al.: Training Compute-Optimal Large Language Models. CoRR abs/2203.15556, 2022, https://arxiv.org/pdf/2203.15556]

DeepMind

Training Regimes

[DeepSeek-AI: DeepSeek-V3 Technical Report, **2025**, https://arxiv.org/pdf/2412.19437]

- **Data/Model Selection**
  - Given a fixed FLOPs budget, select model size & # training tokens
  - Training range of models → Fit curve for scaling
- **Pre-Training**
  - Train selected model in single-pass over data
  - **State-of-the-art: 15T tokens**; e.g., DeepSeek-V3: 14.8T tokens
- **Context-Length Extensions**
  - Phase 1: max context length extended to **32K tokens**
  - Phase 2: max context length extended to **128K tokens**
- **Fine-Tuning**
  - Overfitting on high-quality data, **Supervised Fine-Tuning** (SFT)
  - Reinforcement Learning (RL); **RL from Human Feedback** (RLFH)

Models and Training Cost

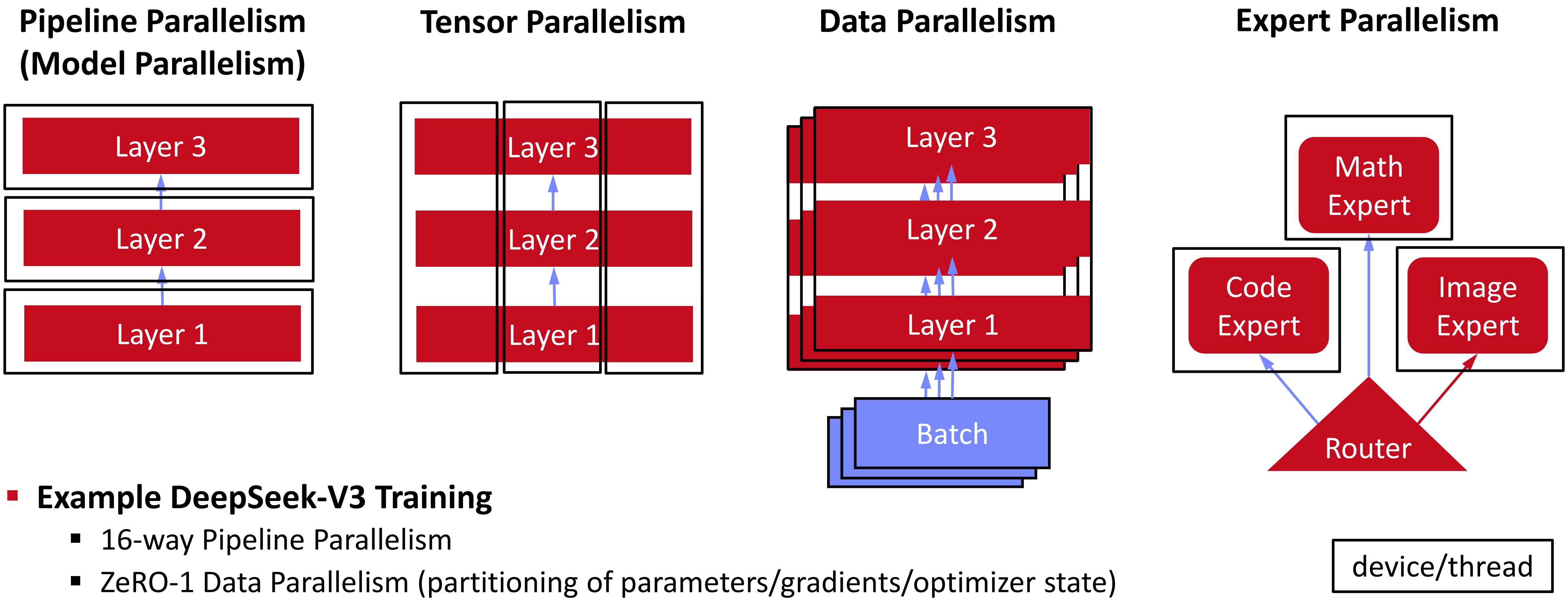
- **OpenAI GPT**
  - GPT-3: **4.5M USD**, GPT-4: **100M USD**
- **Google Gemini**
  - Gemini 1.0 Ultra: **192M USD**
- **Meta Llama**
  - Llama-3.1: **170M USD**
- **DeepSeek**
  - DeepSeek-V3: **5.3M USD**
- **Mistral**
  - Mistral Large: 41M USD
- **X Grok**
  - Grok-2: **107M USD**

[https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/]

[https://www.visualcapitalist.com/the-surging-cost-of-training-ai-models/]

Training Costs	Pre-Training	Context Extension	Post-Training	Total
in H800 GPU Hours	2664K	119K	5K	2788K
in USD	$5.328M	$0.238M	$0.01M	$5.576M

Employed Types of Parallelism



- **Example DeepSeek-V3 Training**
  - 16-way Pipeline Parallelism
  - ZeRO-1 Data Parallelism (partitioning of parameters/gradients/optimizer state)
  - 64-way Expert Parallelism on 8 nodes

Sparse Transformers (e.g., GPT-3)

[Rewon Child, Scott Gray, Alec Radford, Ilya Sutskever: Generating Long Sequences with Sparse Transformers, CoRR 2019, https://arxiv.org/pdf/1904.10509]

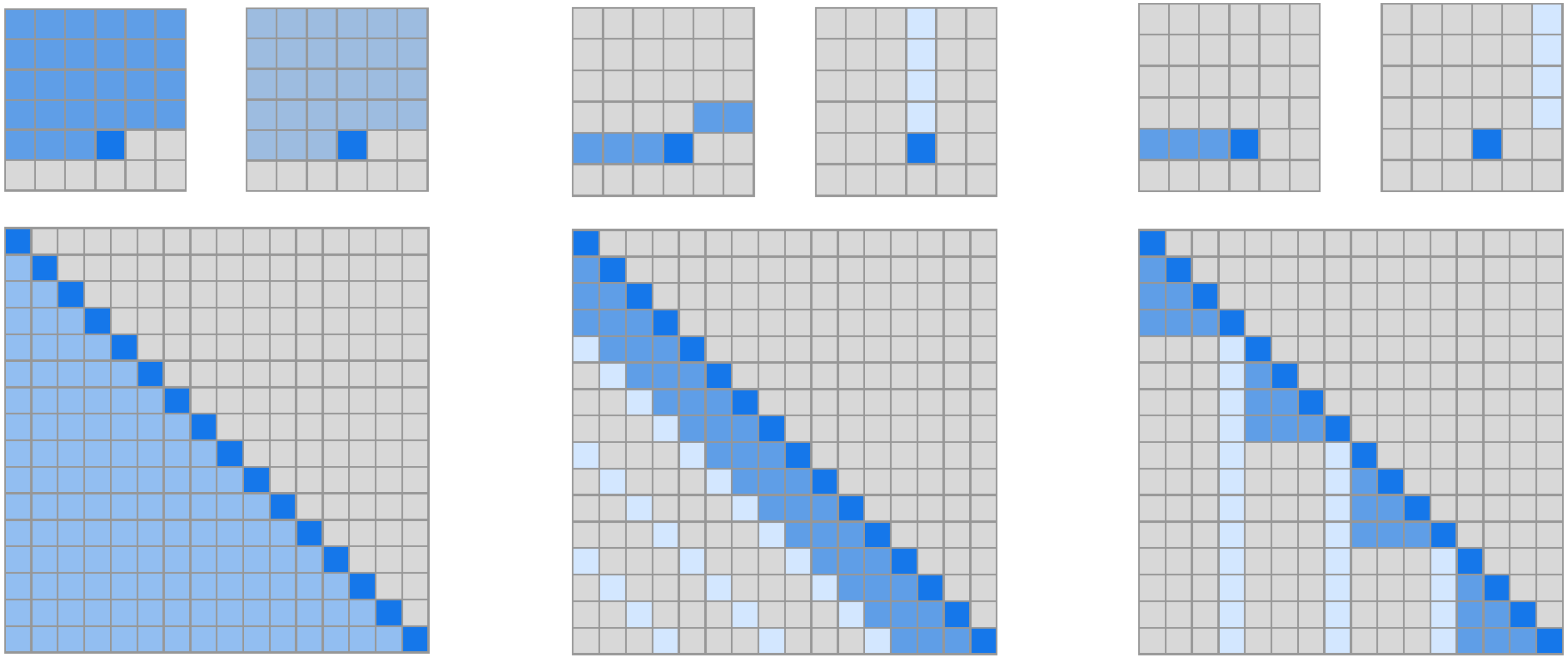
Transformer
O(n^2 * d)

Sparse Transformer
Strided

O(n * √n * d)

Sparse Transformer
Fixed/Blocked

Current Position (Query)

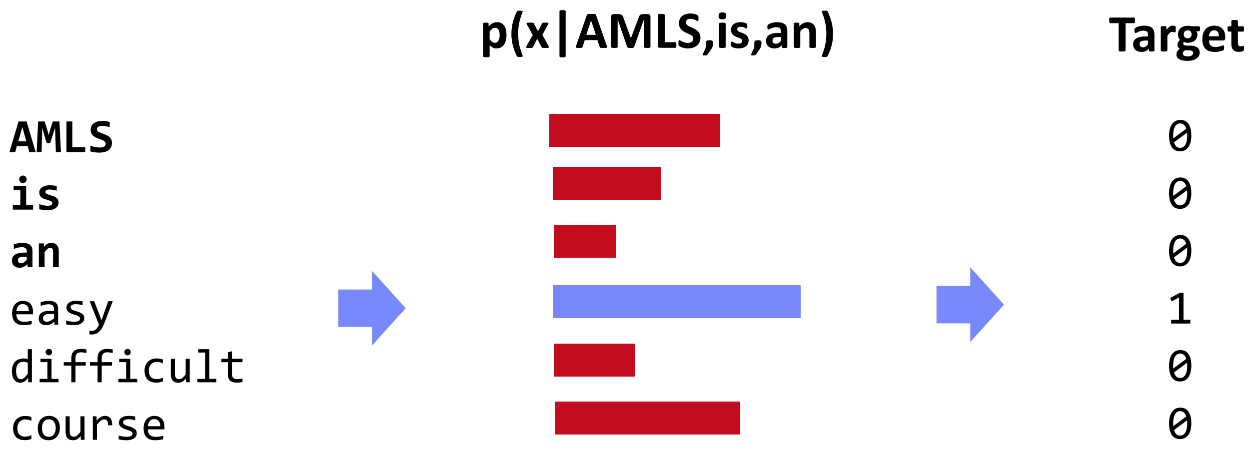


Attention Positions

LLM Post-Training and Fine-Tuning

Training Objective and Losses

- **Objective: Classify next token (multi-class classification)**
  - Cross-entropy loss
  - **Min –log(p(x))**



Types of Post-Training

- **#1 Foundation Model Fine-Tuning**
  - Fine-tuning for specific **high-quality data** (e.g., Wikipedia), and **tasks** (e.g., coding) through feedback
  - **Overfitting** to high-quality data

- **#2 Model Distillation and Quantization**
  - Transferring knowledge from **large teacher model** to **small student model**
  - **Goal:** reduce inference costs / resource requirements (# layers, model dim, attention heads)
  - **Example:** Llama 3.2 (09/2024): 90B, 11B, 3B, 1B parameters + FP32 / FP16 / UINT8 quantized variants

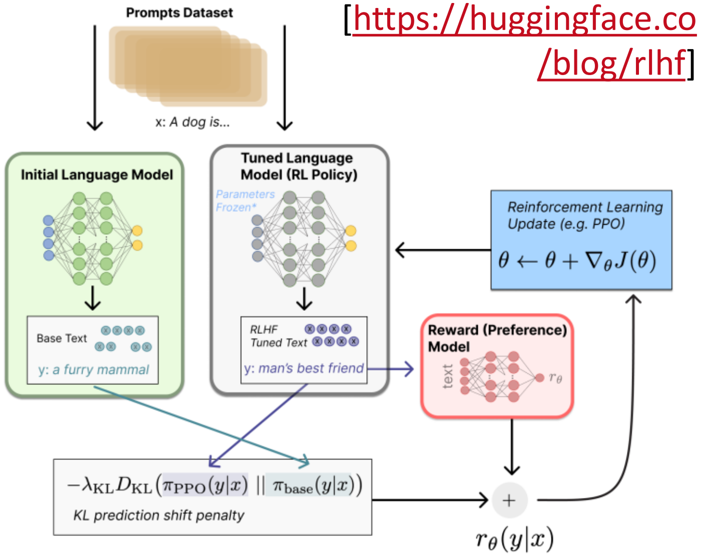
- **#3 User-level Fine-Tuning**
  - Specialize the LLM through user- and domain-specific data
  - Create model variants (**pro:** effectiveness, **con:** number of models)

SFT and RLHF

- **Supervised Fine-Tuning (SFT)**
  - Fine-tune LLM with language modeling (next word prediction) and desired output (human feedback)
  - Example prompts and expected answers (free form text) → **limited by human ability / costs**
  - **Example:** GPT-3 → ChatGPT

- **Reinforcement Learning from Human Feedback (RLHF)**
  - Aks fine-tuned LLM to generate two different answers
    → **humans ranks the answers**
  - #1 Train Reward Model **RM** for classifying preferences
  - #2 Proximal Policy Optimization (PPO) with **RM**
  - Freeze model parameters and **LoRA Low-Rank Adaptation** for LLMs

[https://huggingface.co/blog/rlhf]



[Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen: **LoRA:** Low-Rank Adaptation of Large Language Models. **ICLR 2022**]

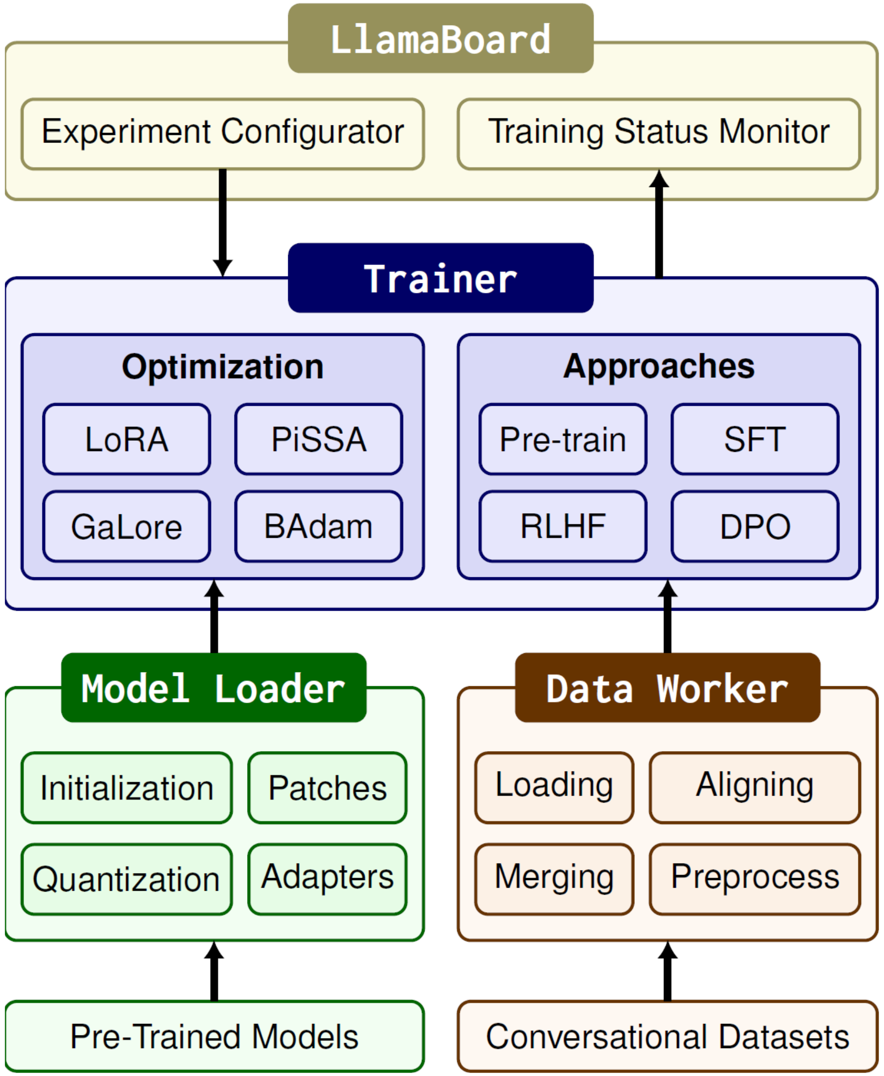
$$h = W_0 x + \Delta W x = W_0 x + BAx$$

User-Level Fine-Tuning Tools

- **Example LlamaFactory**

	LlamaFactory	FastChat	LitGPT	LMFlow	Open-Instruct
LoRA	✓	✓	✓	✓	✓
QLoRA	✓	✓	✓	✓	✓
DoRA	✓				
LoRA+	✓				
PiSSA	✓				
GaLore	✓	✓		✓	✓
BAdam	✓				
Flash attention	✓	✓	✓	✓	✓
S^2 attention	✓				
Unsloth	✓		✓		
DeepSpeed	✓	✓	✓	✓	✓
SFT	✓	✓	✓	✓	✓
RLHF	✓			✓	
DPO	✓				✓
KTO	✓				
ORPO	✓				

[Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Yongqiang Ma: LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. CoRR abs/2403.13372 (2024), https://arxiv.org/pdf/2403.13372]



LLM Inference

Prompting Strategies

- **#1 Zero-shot**
  - General question answering and task processing, often via **prompt templates**
  - **No provided examples**, best effort results
- **#2 Few-shot**
  - Give **general instruction and few examples**; one-shot as an extreme case (“human instruction”)
  - LLM executes task while considering the pattern of examples
- **DSPy**
  - Programmatic, compositional interaction with LLMs

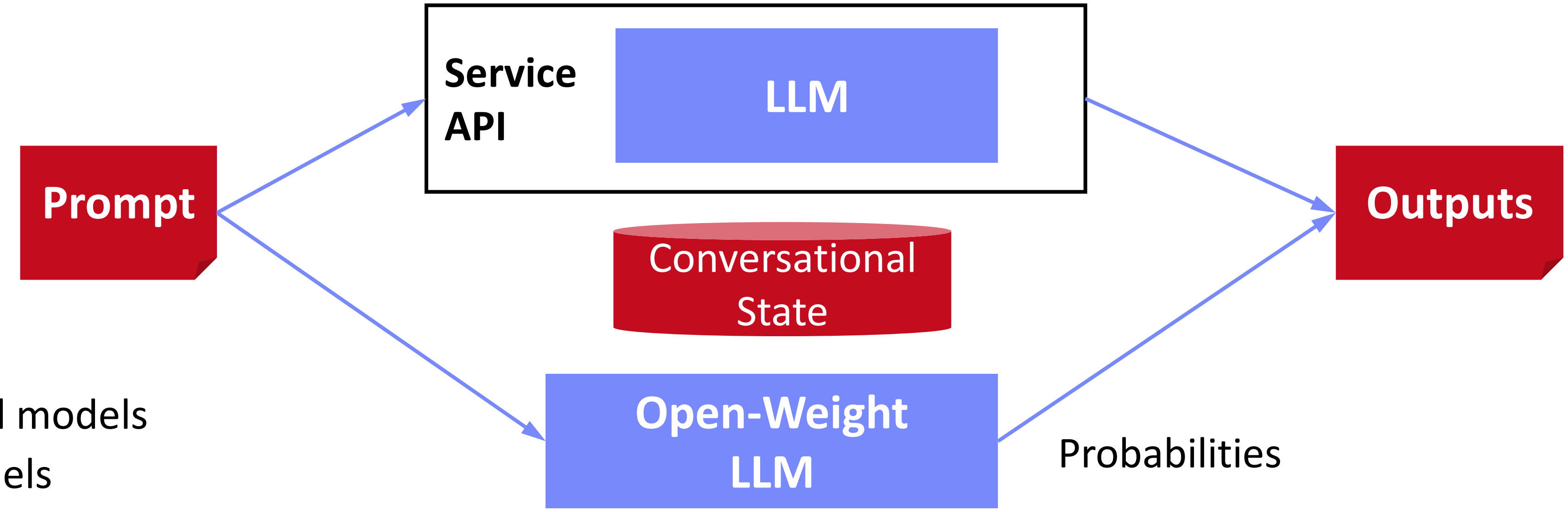
[https://github.com/stanfordnlp/dspy]

- **Jail Breaking**
  - Append special prefixed/suffixes for getting refused answers from LLM
  - **Example:** optimized **suffix search for probability of “Sure”**

[Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion: Jailbreaking Leading Safety-Aligned LLMs with Simple Adaptive Attacks. **CoRR 2024** https://arxiv.org/pdf/2404.02151v4]

LLM Inference Overview

- **Overview**



- **Model Variants**
  - Large and small models
  - Quantized models

- **Service API Costs**
  - Billed by input/output tokens
  - GPT-4.1: 2/8$ per 1M input/output tokens; **0.5$ / 1M cached input tokens**
  - GPT-4.1-mini: 0.4/1.6$ per 1M input/output tokens; **0.1$ / 1M cached input tokens**

LLM Offline and Online Inference

Easy, fast, and cheap LLM serving for everyone

[https://docs.vllm.ai/en/stable/getting_started/examples/basic.html#]

- **vLLM: LLM Inference and Serving**
  - Management of attention key/value cache
  - Batching of incoming requests, eviction mechanisms
  - HuggingFace models and hardware accelerators

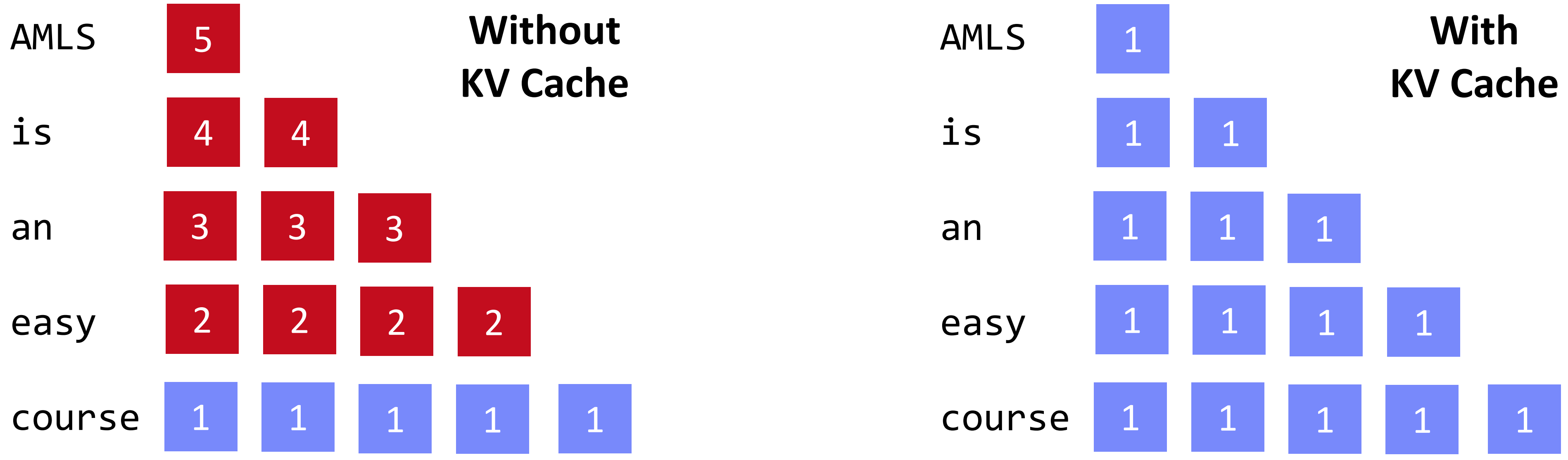
```python
from vllm import LLM, SamplingParams

# Sample prompts.
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]
# Create a sampling params object.
sampling_params = SamplingParams(
  temperature=0.8, top_p=0.95)
...
```

```python
# Create an LLM.
llm = LLM(model="facebook/opt-125m")
# Generate texts from the prompts (RequestOutput)
outputs = llm.generate(prompts, sampling_params)
# Print the outputs.
print("\nGenerated Outputs:\n" + "-" * 60)
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt:    {prompt!r}")
    print(f"Output:    {generated_text!r}")
    print("-" * 60)
```

Key-Value (KV) Caching

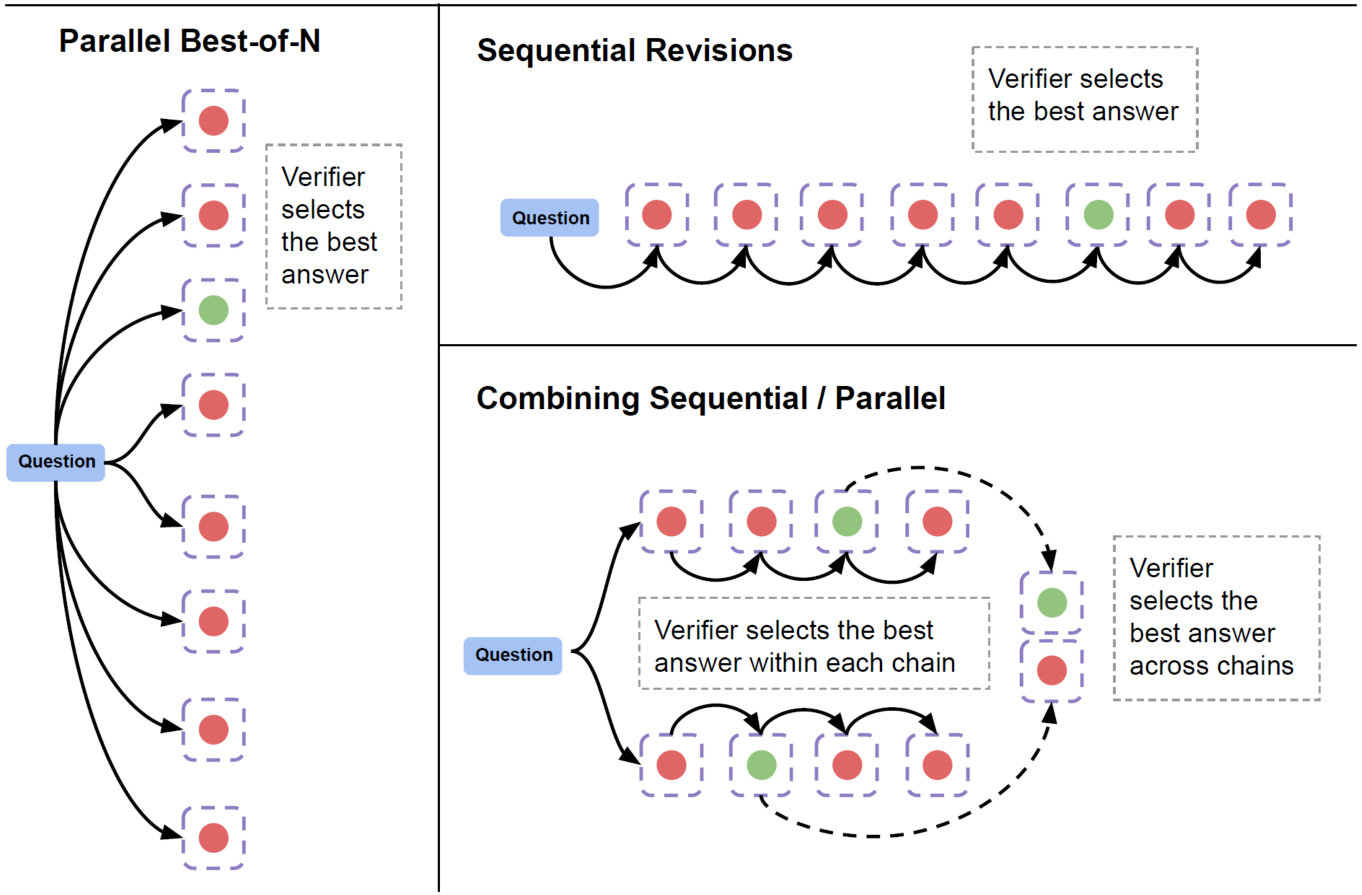
- **Basic Idea**
  - For every token, compute squared attention to all previous tokens → **large overlap**
  - Establish KV cache to **compute rows of keys/values just ones**



Test-Time Compute

- **Overview: Leverage multiple LLM invocations for improved accuracy at expense of compute costs**

[Charlie Snell, Jaehoon Lee, Kelvin Xu, Aviral Kumar: Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. CoRR abs/2408.03314 (2024) https://arxiv.org/pdf/2408.03314]



Summary & QA

- **Basics and Terminology**
- **LLM Data Preprocessing**
- **LLM Pre-Training**
- **LLM Post-Training and Fine-Tuning**
- **LLM Inference**

- **Next Lectures** (Part A)
  - **08 Hybrid Execution and HW Accelerators** [Jun 12]
  - **09 Caching, Partitioning, Indexing and Compression** [Jun 19]