LLM Storage Compression
&
LLM Systems – from Training to Serving

DS 5110: Big Data Systems

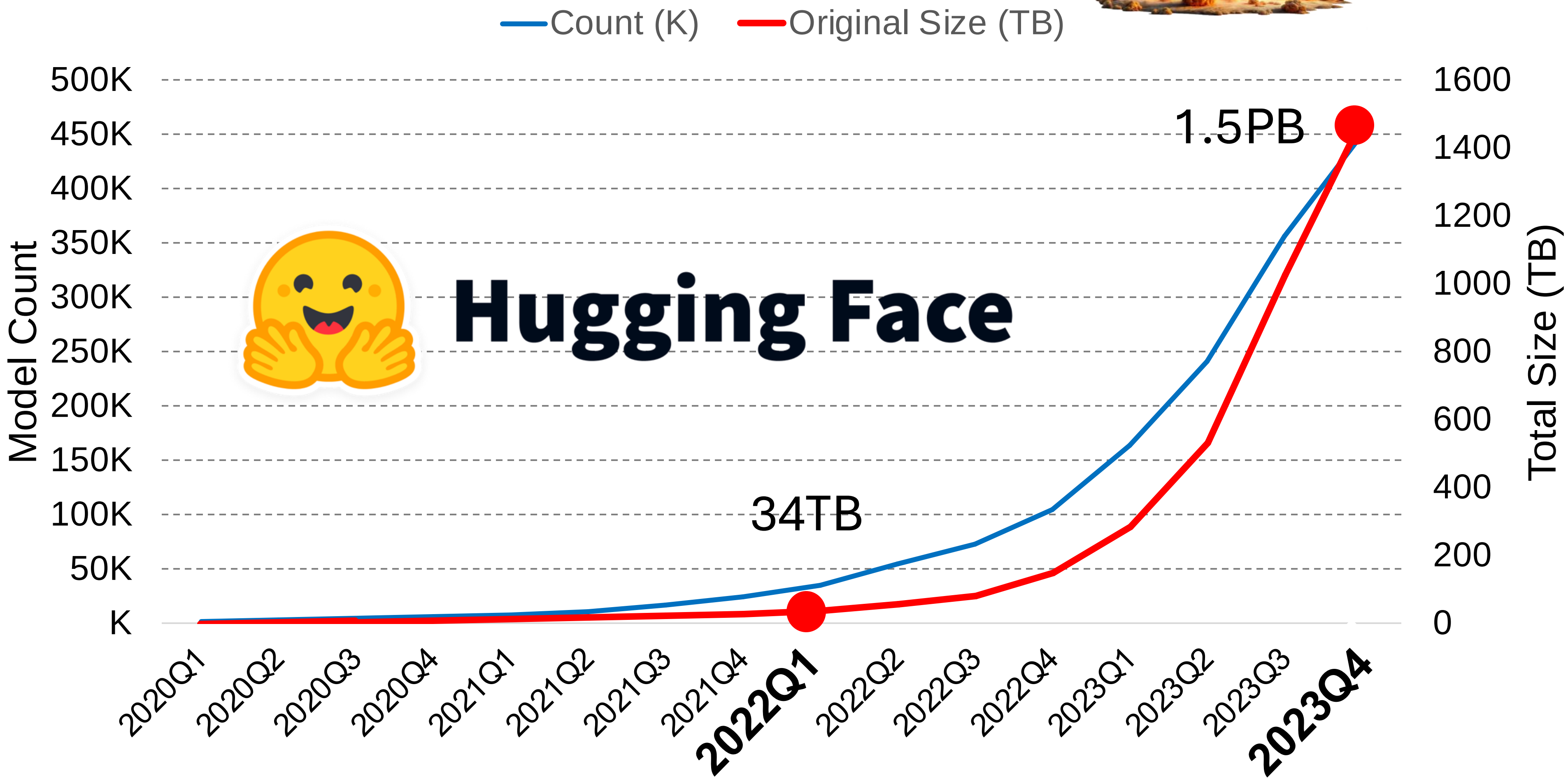
Spring 2025

Lecture 15

Zhaoyuan Su

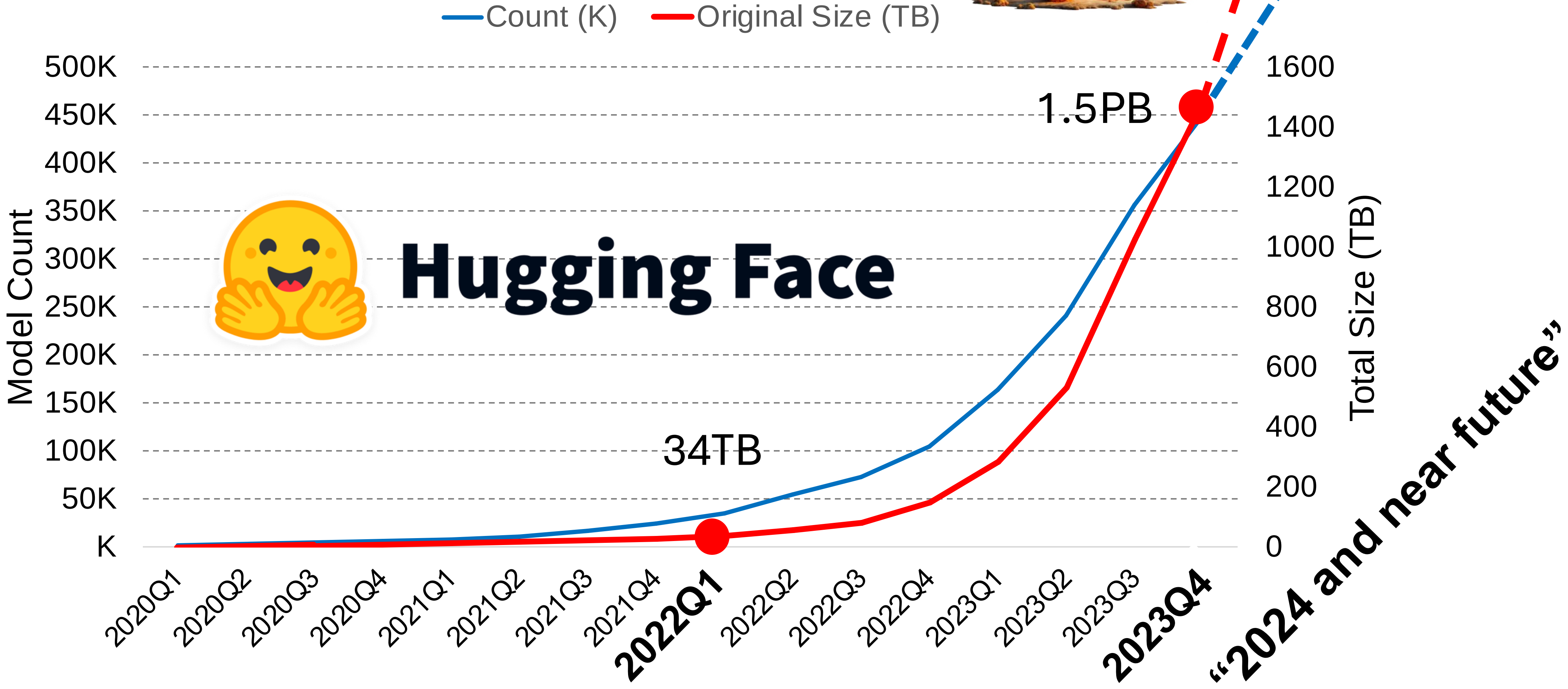
University of Virginia

ML Model Storage is  !



Su, Zhaoyuan, et al. "Everything You Always Wanted to Know About Storage Compressibility of Pre-Trained ML Models but Were Afraid to Ask." Proceedings of the VLDB Endowment 17.8 (2024): 2036-2049.

ML Model Storage is  !



HuggingFace’s pre-trained models (PTMs) are growing exponentially!

ML Model Storage is !

HuggingFace's pre-trained models (PTMs) are growing exponentially!

- **Q1:** What are the characteristics of PTM storage datasets?
- **Q2:** Are existing data reduction tools effective for reducing the sizes of PTMs?

Contribution 1:
Analysis of A Large-Scale PTM Storage

Category	Count (%)	Total Size in GB (%)
NLP	300 (33.33%)	170.85 (29.67%)
Audio	150 (16.67%)	154.30 (26.79%)
Multimodal	150 (16.67%)	97.81 (16.99%)
CV	150 (16.67%)	58.74 (10.20%)
Uninformed	150 (16.67%)	94.18 (16.35%)
Overall	900 (100%)	575.88 (100%)

We collected a PTM dataset from HuggingFace, which includes **900 PTMs** across multiple categories, in a total size of **575.88GB**.

Key Observations

- PTMs are growing exponentially and are generally large, with 90% > 100MB and 25% > 1GB

Demands effective data reduction methods

- PTMs are deep, with 75% having over 200 layers

- Float32 layers dominate, accounting for 96.87% of storage

Key Observations

- PTMs are growing exponentially and are generally large, with 90% > 100MB and 25% > 1GB → **Demands effective data reduction methods**
- PTMs are deep, with 75% having over 200 layers → Are there duplicates at chunk level?
- Float32 layers dominate, accounting for 96.87% of storage → Are there duplicates at parameter level?

Contribution 2:
Analysis of PTM Storage Compressibility

- Coarse-grained data chunks:
  - Storage deduplication
  - Delta compression

- Fine-grained parameters:
  - Distance encoding

Would Storage Dedup Help?

Data Type	Total Sz (GB)	Size of Duplicates in GB (%)		
		4 KB (FSC)	512 B (FSC)	CDC
float32	557.84	40.35 (7.23%)	42.92 (7.69%)	44.50 (8.16%)
float16	14.51	0.14 (0.96%)	0.14 (0.96%)	0.15 (1.03%)
float64	0.81	0 (0%)	0 (0%)	0 (0%)
uint8	1.75	1.74 (99.43%)	1.74 (99.43%)	1.74 (99.43%)
int64	0.97	0.94 (96.91%)	0.96 (98.97%)	0.96 (98.97%)
Overall	575.88	43.17 (7.50%)	45.76 (7.95%)	47.35 (8.22%)

Would Storage Dedup Help?

Data Type	Total Sz (GB)	Size of Duplicates in GB (%)		
		4 KB (FSC)	512 B (FSC)	CDC
float32	557.84	40.35 (7.23%)	42.92 (7.69%)	44.50 (8.16%)
float16	14.51	0.14 (0.96%)	0.14 (0.96%)	0.15 (1.03%)
float64	0.81	0 (0%)	0 (0%)	0 (0%)
uint8	1.75	1.74 (99.43%)	1.74 (99.43%)	1.74 (99.43%)
int64	0.97	0.94 (96.91%)	0.96 (98.97%)	0.96 (98.97%)
Overall	575.88	43.17 (7.50%)	45.76 (7.95%)	47.35 (8.22%)

Both fixed-sized chunking (FSC) and content-defined chunking (CDC) yield similarly negative results

Would Delta Compression Help?

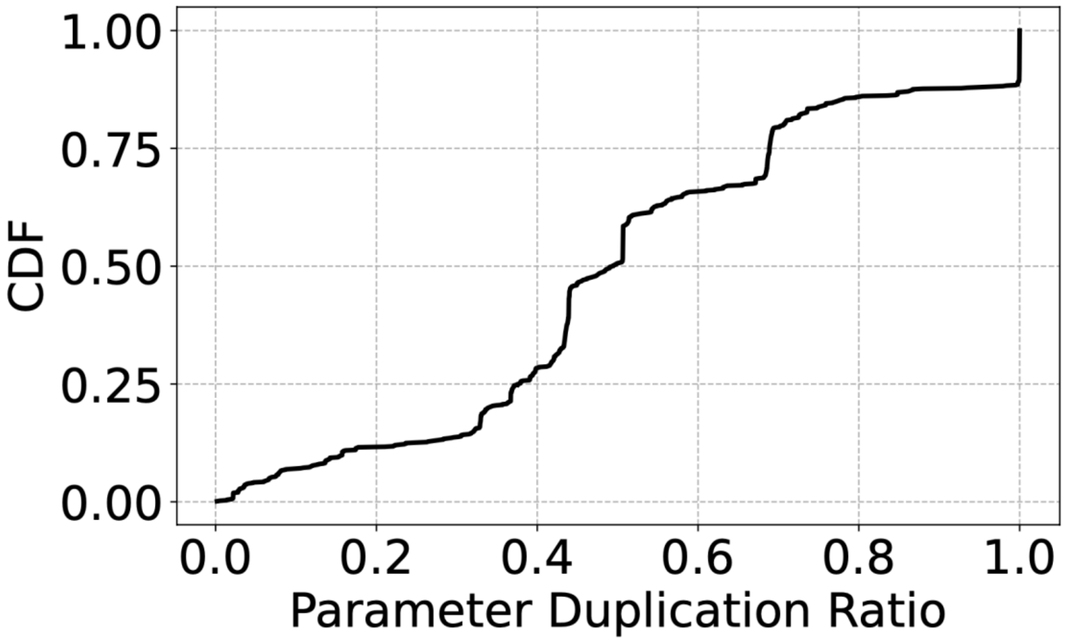
Data Type	Total Sz (GB)	Size of Similar Data in GB (%)		
		Layer	4 KB	512 B
float32	557.84	30.17 (5.41%)	40.39 (7.24%)	43.12 (7.73%)
float16	14.51	0.14 (0.96%)	0.14 (0.96%)	0.14 (0.96%)
float64	0.81	0 (0%)	0 (0%)	0 (0%)
uint8	1.75	1.74 (99.43%)	1.74 (99.43%)	1.74 (99.43%)
int64	0.97	0.94 (96.91%)	0.95 (97.94%)	0.96 (98.97%)
Overall	575.88	32.99 (5.73%)	43.22 (7.51%)	45.96 (7.98%)

Would Delta Compression Help?

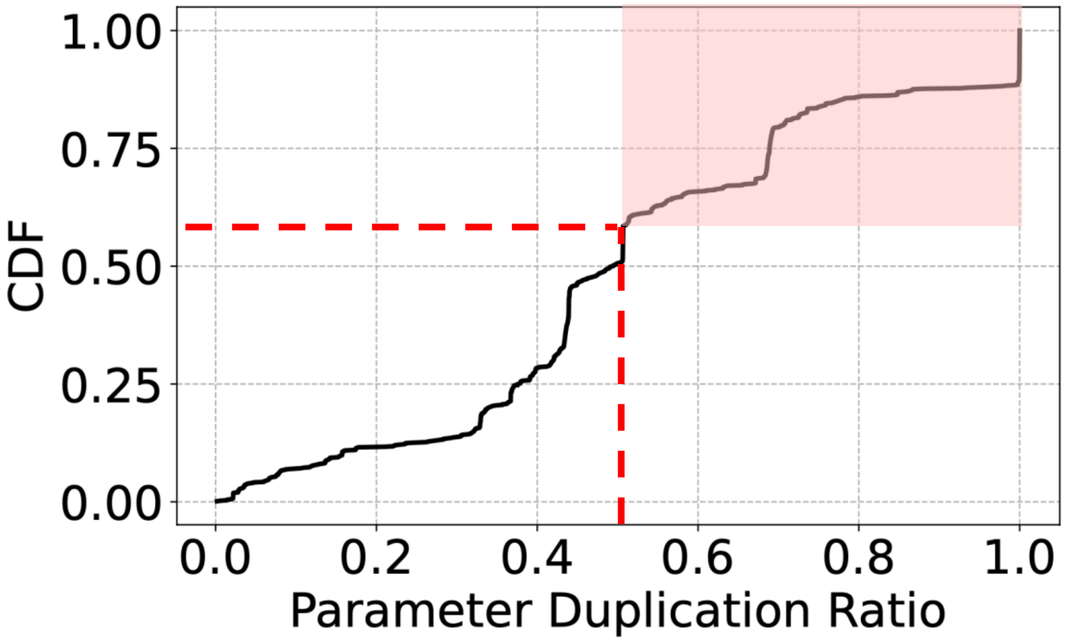
Data Type	Total Sz (GB)	Size of Similar Data in GB (%)		
		Layer	4 KB	512 B
float32	557.84	30.17 (5.41%)	40.39 (7.24%)	43.12 (7.73%)
float16	14.51	0.14 (0.96%)	0.14 (0.96%)	0.14 (0.96%)
float64	0.81	0 (0%)	0 (0%)	0 (0%)
uint8	1.75	1.74 (99.43%)	1.74 (99.43%)	1.74 (99.43%)
int64	0.97	0.94 (96.91%)	0.95 (97.94%)	0.96 (98.97%)
Overall	575.88	32.99 (5.73%)	43.22 (7.51%)	45.96 (7.98%)

Similarity-based delta compression is ineffective across various chunk granularities

Would Dictionary Coding Help?

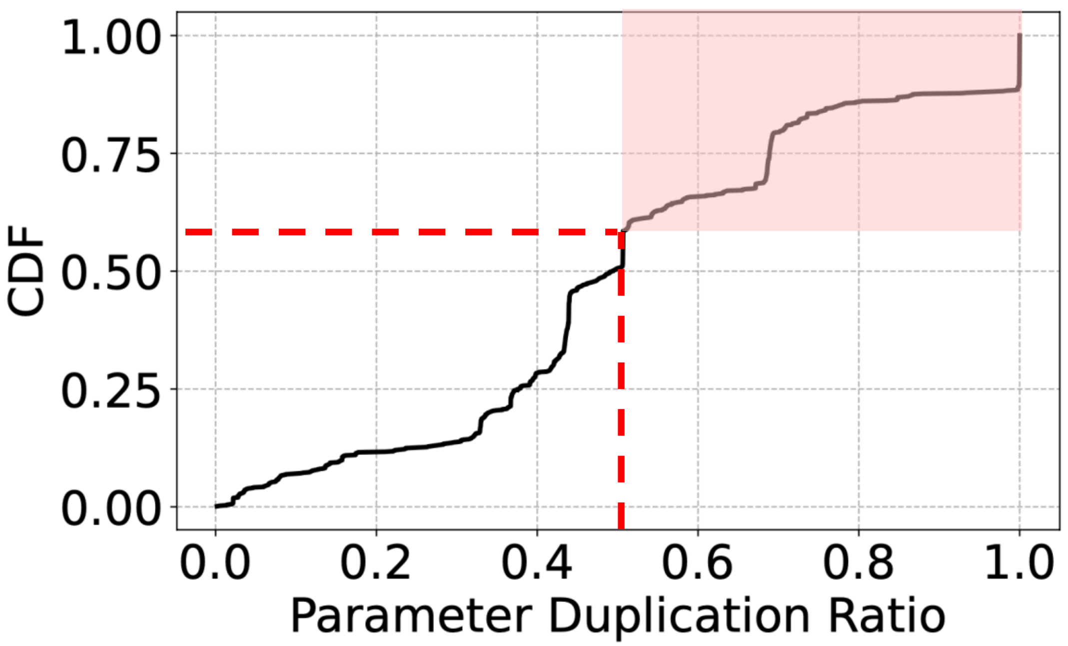


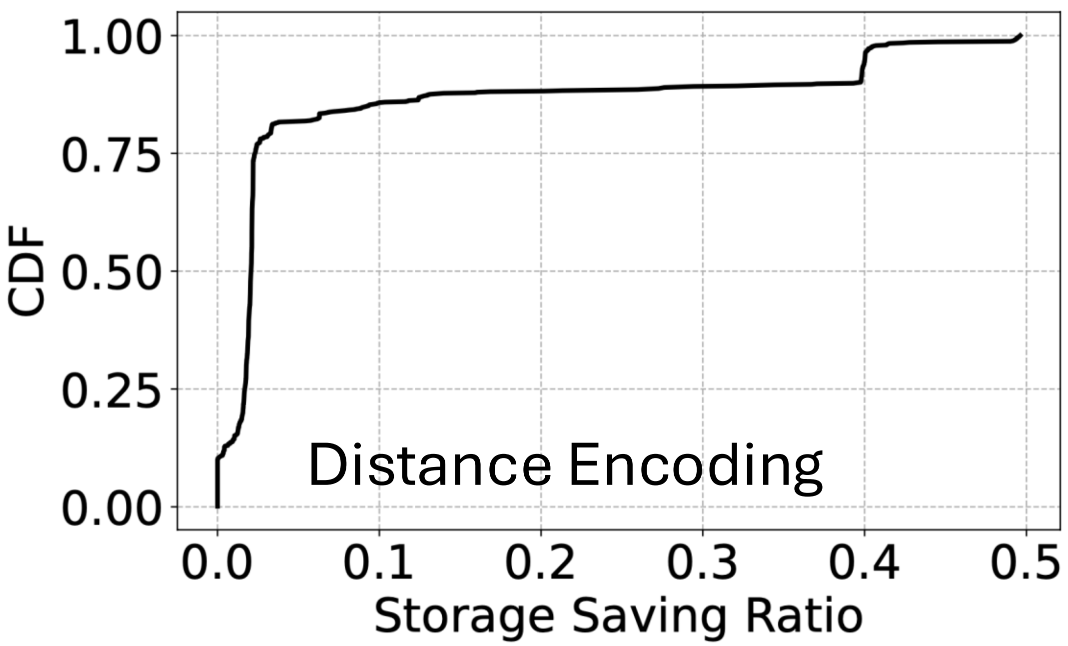
Would Dictionary Coding Help?



Most PTMs have duplicate parameters

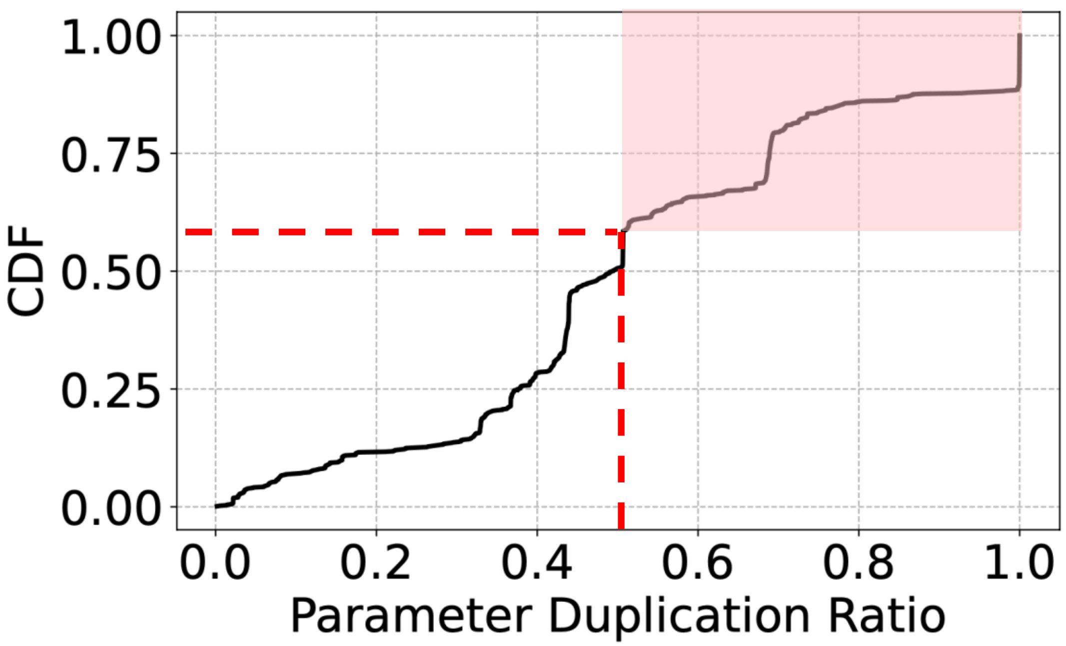
Would Dictionary Coding Help?

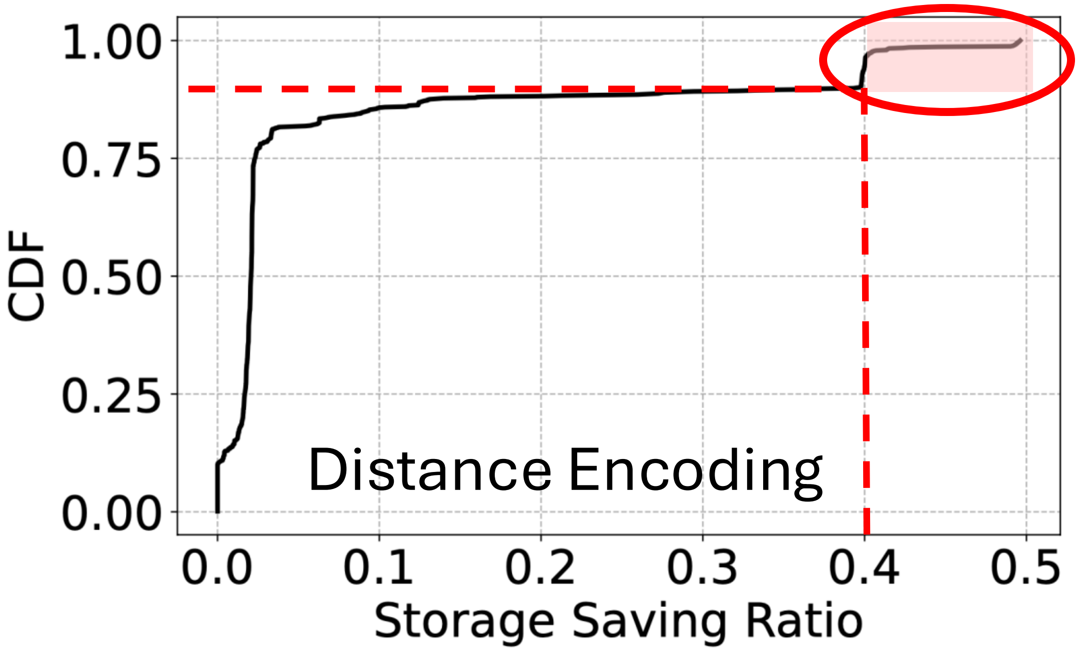




Most PTMs have duplicate parameters

Would Dictionary Coding Help?





Most PTMs have duplicate parameters

However, distance encoding only helps for ~10% of PTMs

Takeaway:
Analysis of PTM Storage Compressibility

- Duplication and resemblance pattern are minimal at data chunk level

- Parameter dedup only helps for only a small fraction of PTMs with extremely high parameter repetition

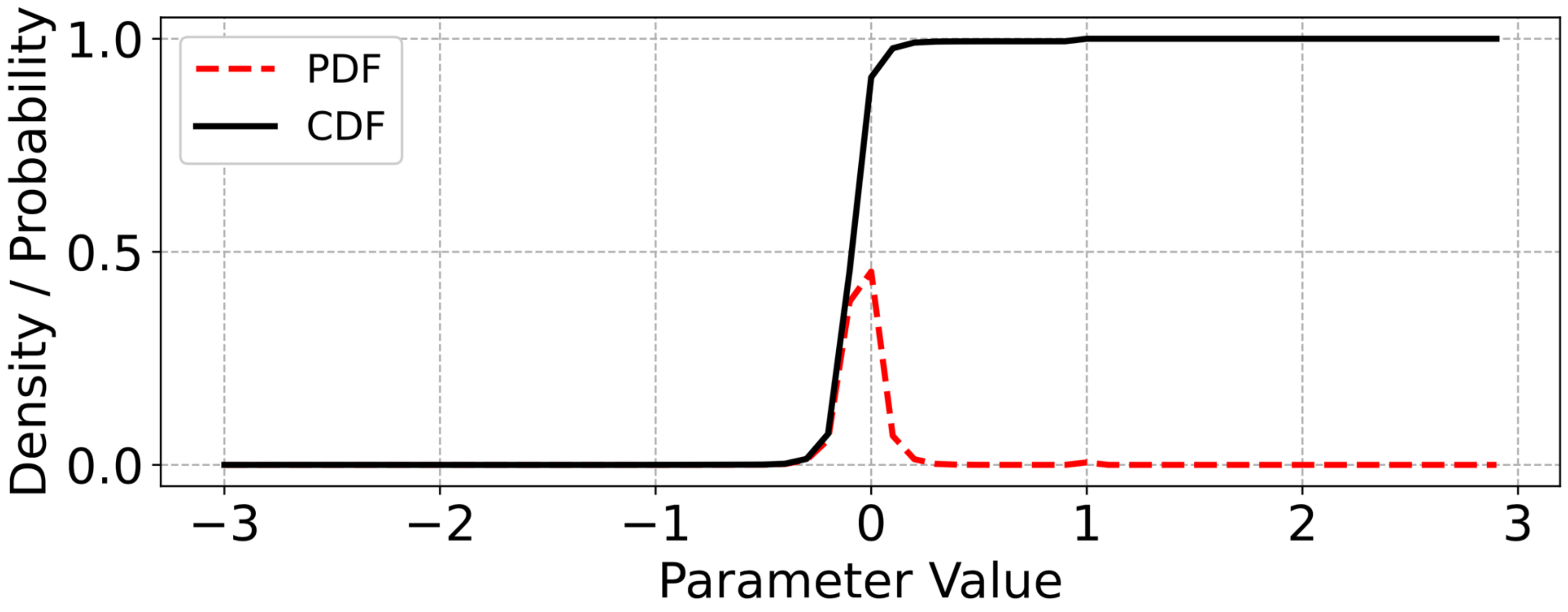
- Parameter randomness makes PTM storage compression challenging

Contribution 3:
Exponent-Less Floating-Point Compression (ELF)

- Exploits PTMs’ data distribution and floating-point arithmetic properties
- ELF compression: Align the parameter magnitude to [1, 2) in order to eliminate common exponent

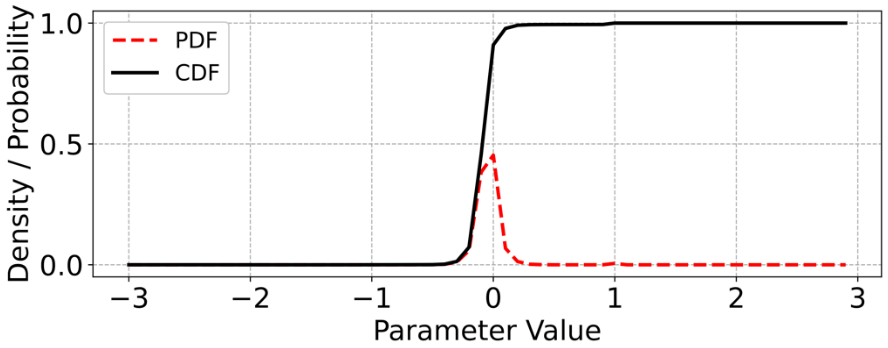
ELF: Key Observations

- **Observation 1:** Around **99%** of all parameters fall within **(-1, 1)**



ELF: Key Observations

- Observation 1: Around **99%** of all parameters fall within **(-1, 1)**



- Observation 2: floats falling within **[1, 2)** share same exponent

IEEE 754 float32

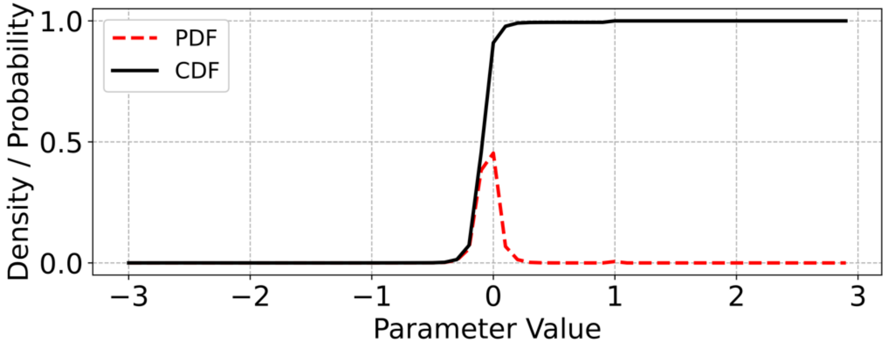
Sign	Exponent	Mantissa
1 bit	8 bits	23 bits

IEEE 754 float16

Sign	Exponent	Mantissa
1 bit	5 bits	10 bits

ELF: Key Observations

- **Observation 1:** Around **99%** of all parameters fall within **(-1, 1)**



- **Observation 2:** floats falling within **[1, 2)** share same exponent

P_1_bin:

0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**

P_1_dec: $(-1)^s \times \mathbf{2^{e-127}} \times (1.m_1m_2...m_{23})_2 = (-1)^0 \times \mathbf{2^0} \times (1.001...0101)_2 = \mathbf{1.1415926218}$

P_2_bin:

0	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**

P_2_dec: $(-1)^s \times \mathbf{2^{e-127}} \times (1.m_1m_2...m_{23})_2 = (-1)^0 \times \mathbf{2^0} \times (1.000...0101)_2 = \mathbf{1.0987650156}$

ELF Compression

For parameters that fall within (-1, 1)

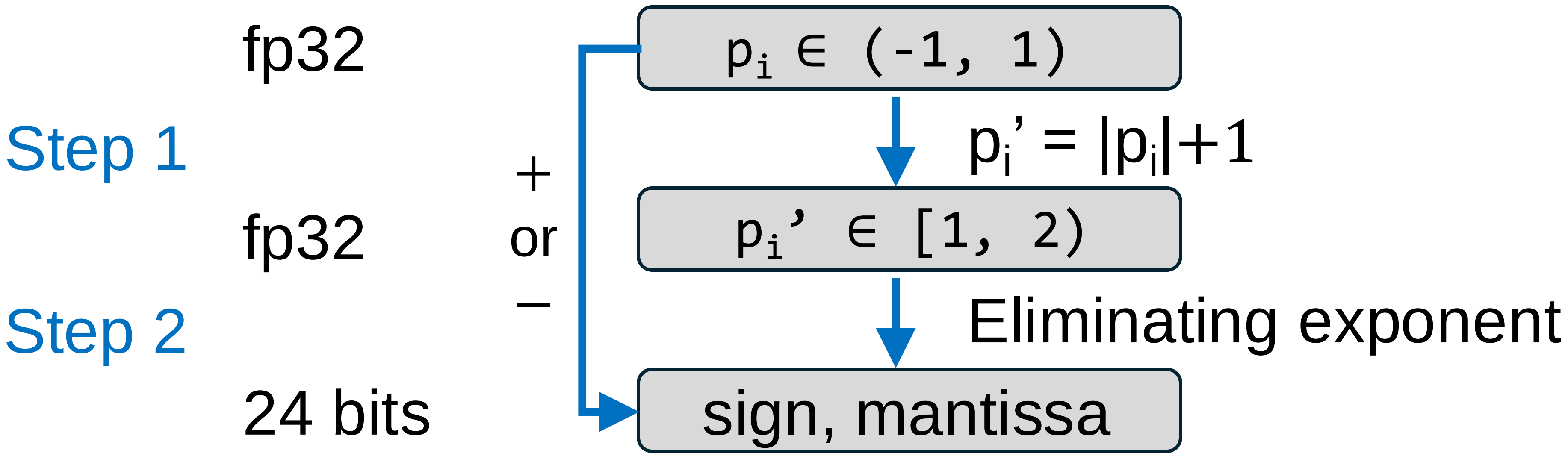
fp32 $p_i \in (-1, 1)$

Step 1 $p_i' = |p_i|+1$

fp32 $p_i' \in [1, 2)$

ELF Compression

For parameters that fall within (-1, 1)



ELF Compression

For parameters that fall within (-1, 1)

fp32 — $p_i \in (-1, 1)$

Step 1 — $p_i' = |p_i|+1$

fp32 — $p_i' \in [1, 2)$

Step 2 — Eliminating exponent

+ or −

24 bits — sign, mantissa

Step 3 — Concatenating and converting

3 `uint8` — $\mathrm{uint}_0, \mathrm{uint}_1, \mathrm{uint}_2$

Appending

`uint8` array — $[\mathrm{ui}_0, \mathrm{ui}_1, \mathrm{ui}_2, \ldots, \mathrm{ui}_m]$

ELF Decompression

Perform decompression to restore p_i

Step 1

24 bits — sign, mantissa

Extracting sign and mantissa

3 `uint8` — $\mathtt{uint}_0, \mathtt{uint}_1, \mathtt{uint}_2$

Reading

`uint8` array — $[\mathtt{ui}_0, \mathtt{ui}_1, \mathtt{ui}_2, \ldots, \mathtt{ui}_m]$

ELF Decompression

Perform decompression to restore p_i

fp32 — $p_i' \in [1, 2)$

Step 2 — ↑ Appending exponent `01111111`

24 bits — sign, mantissa

Step 1 — ↑ Extracting sign and mantissa

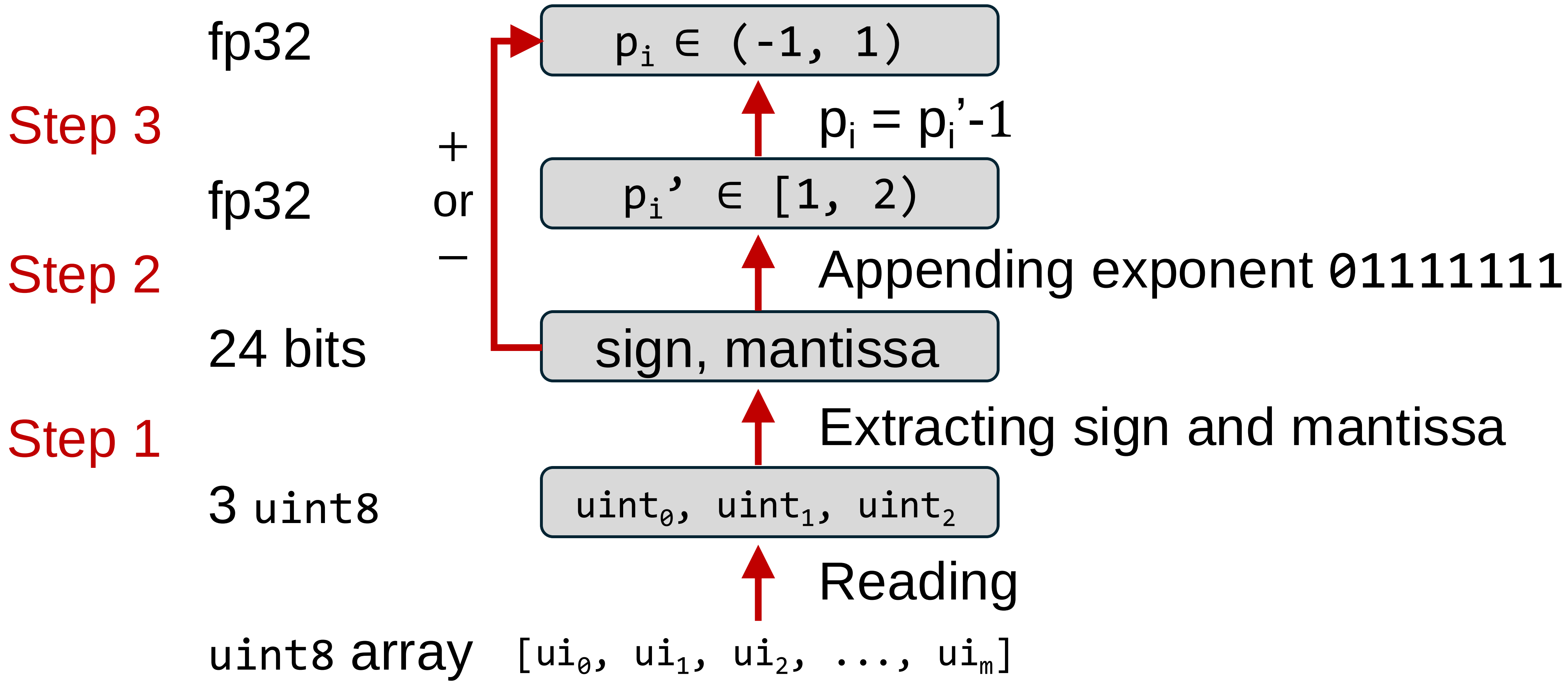
3 `uint8` — $uint_0$, $uint_1$, $uint_2$

↑ Reading

`uint8` array — $[ui_0, ui_1, ui_2, \ldots, ui_m]$

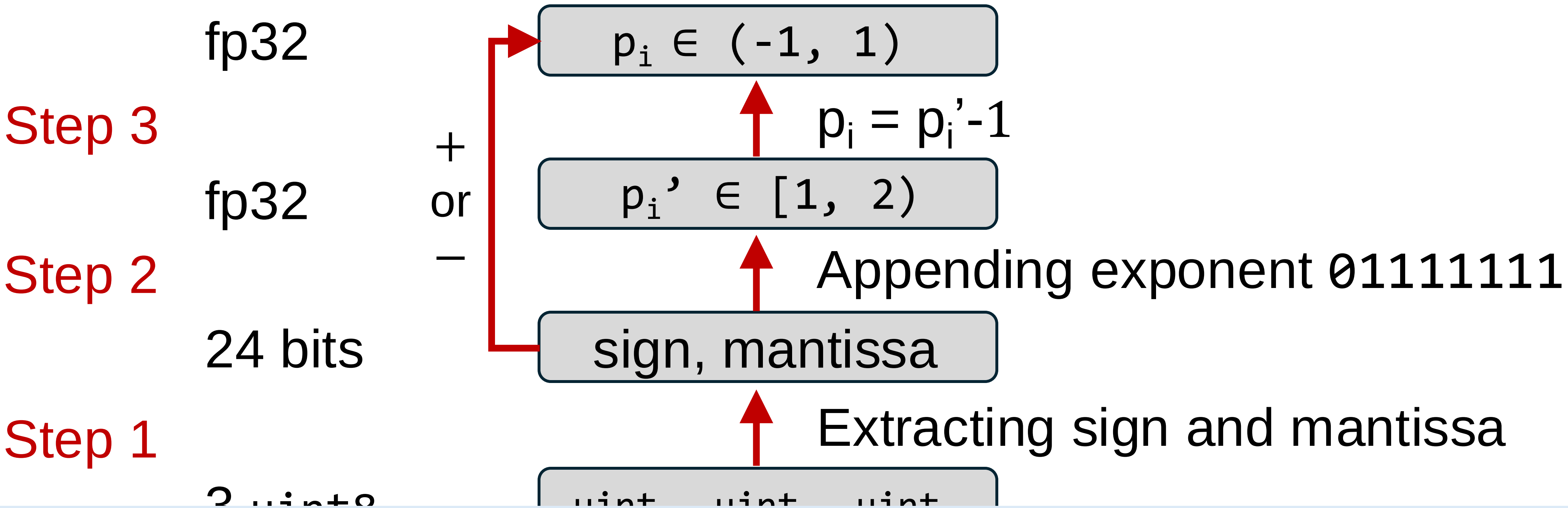
ELF Decompression

Perform decompression to restore p_i



ELF Decompression

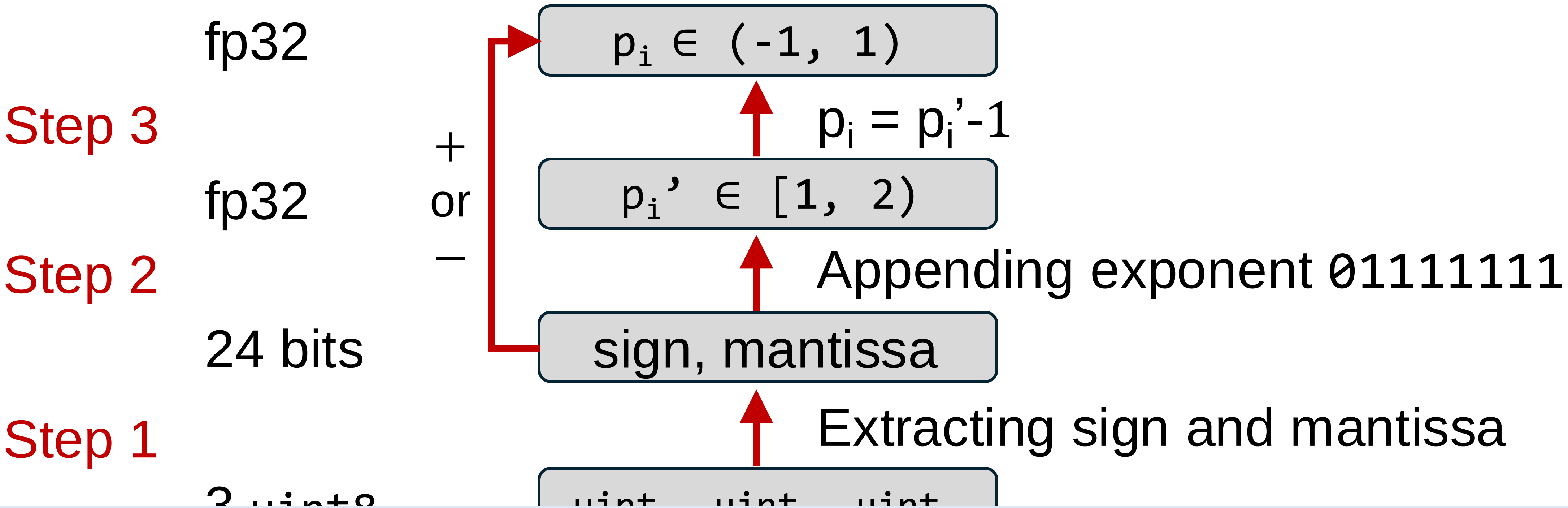
Perform decompression to restore p_i



ELF is **lossy** – It introduces bounded errors due to exponent alignment and mantissa shift performed during floating-point add.

ELF Decompression

Perform decompression to restore p_i

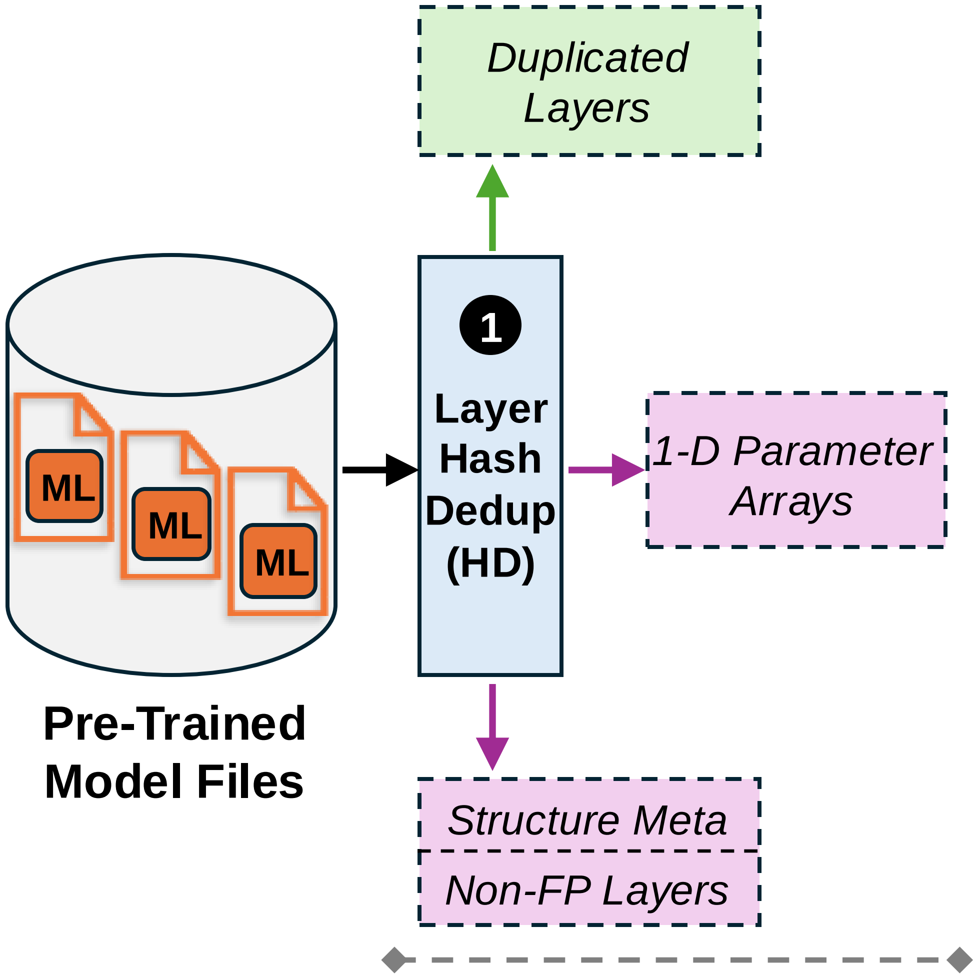


ELF is **lossy** – It introduces bounded errors due to exponent alignment and mantissa shift performed during floating-point add. The errors are **bounded to 2^{-24}** for float32.

Contribution 4:
ELVES: A PTM Compression Framework built on ELF

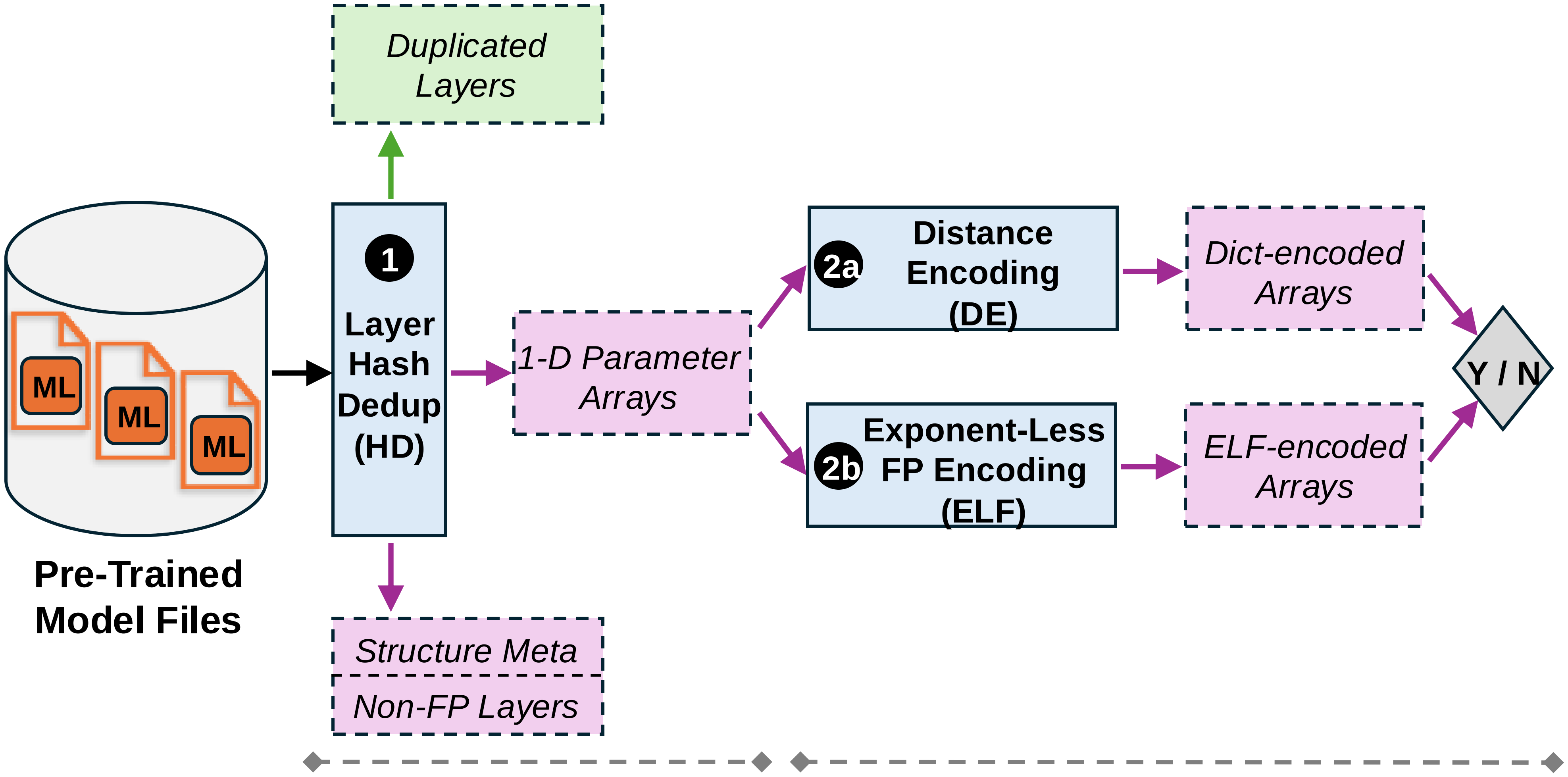
- ELVES combines the best of both worlds between ELF and existing data reduction methods that we've explored
  - including layer-based hash dedup (HD)
  - and parameter-level distance encoding (DE)

ELVES Workflow



Stage 1:
Eliminating duplicate layers

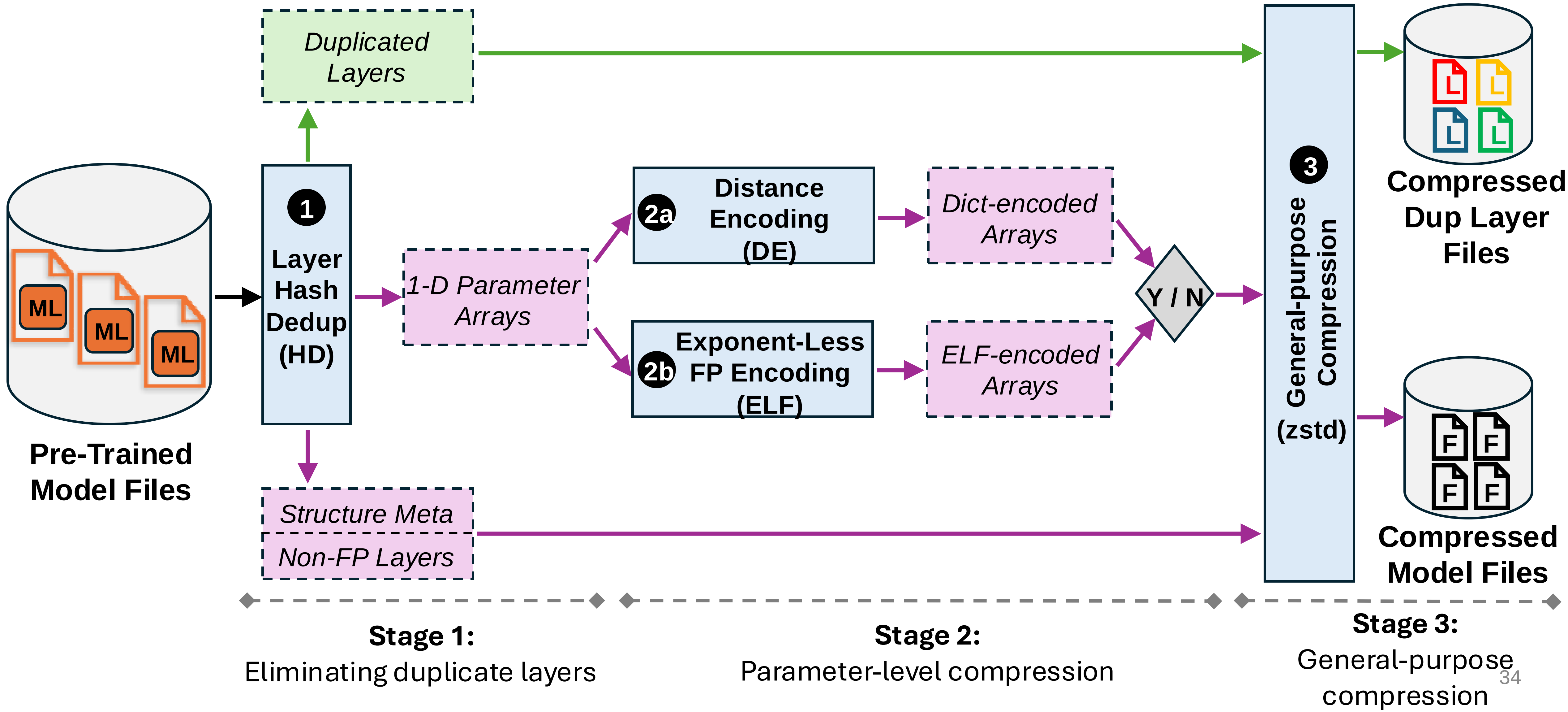
ELVES Workflow



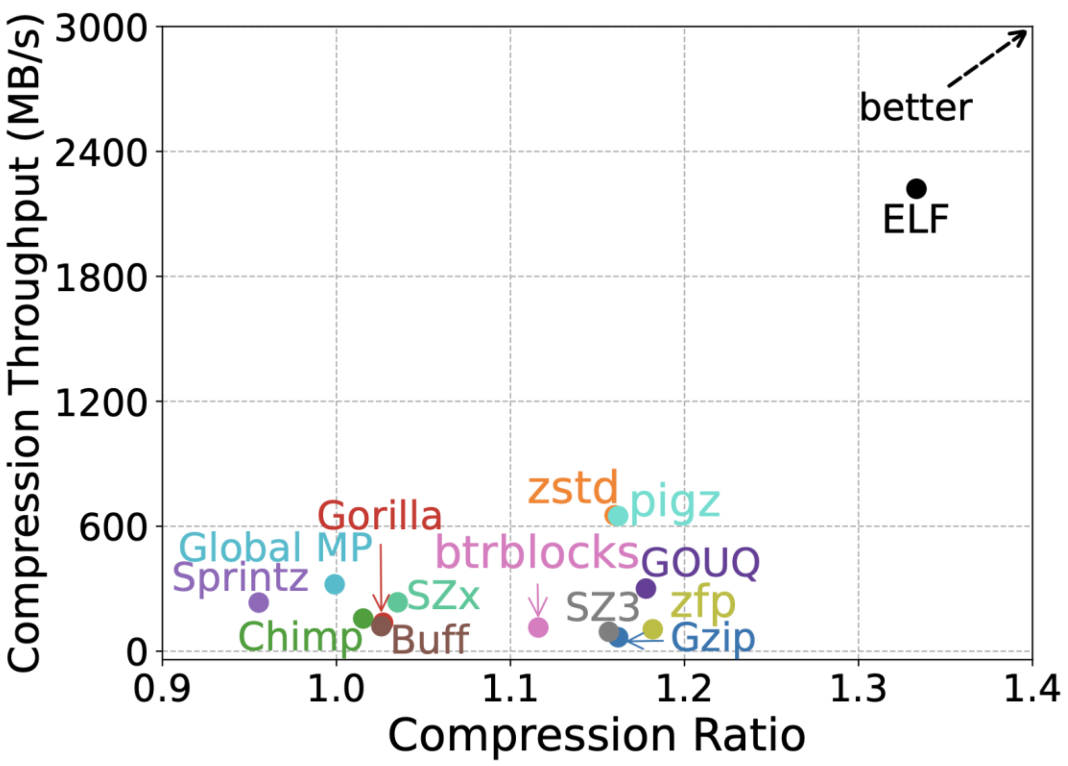
Stage 1:
Eliminating duplicate layers

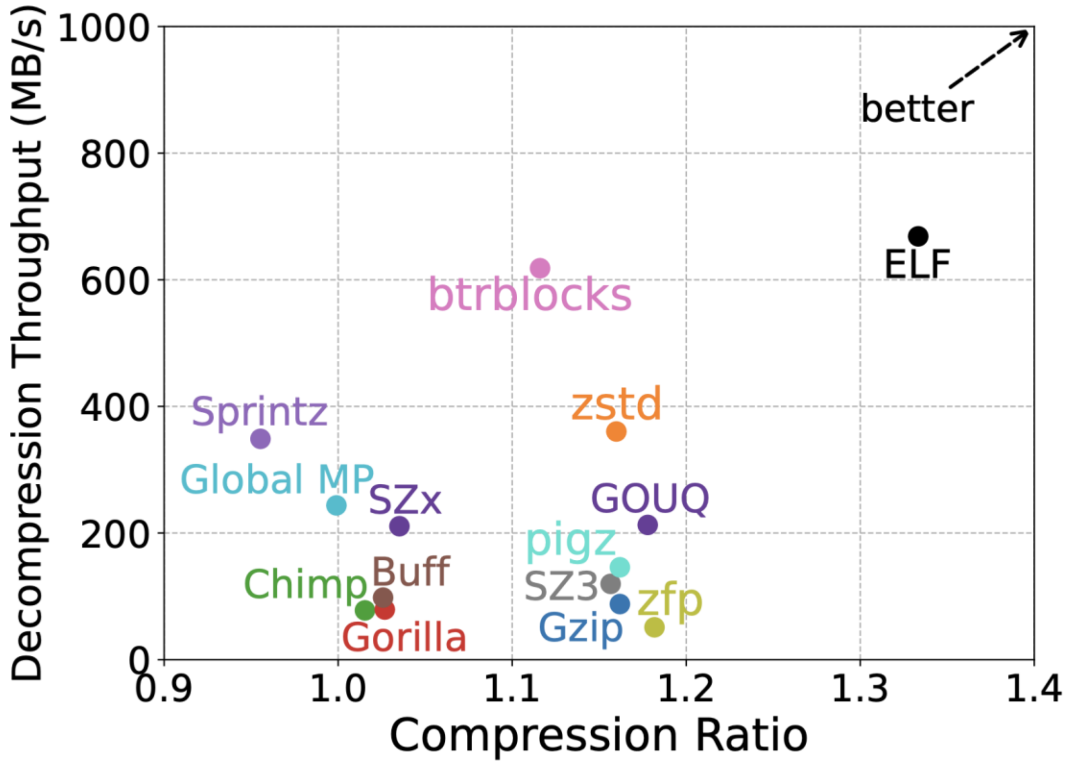
Stage 2:
Parameter-level compression

ELVES Workflow

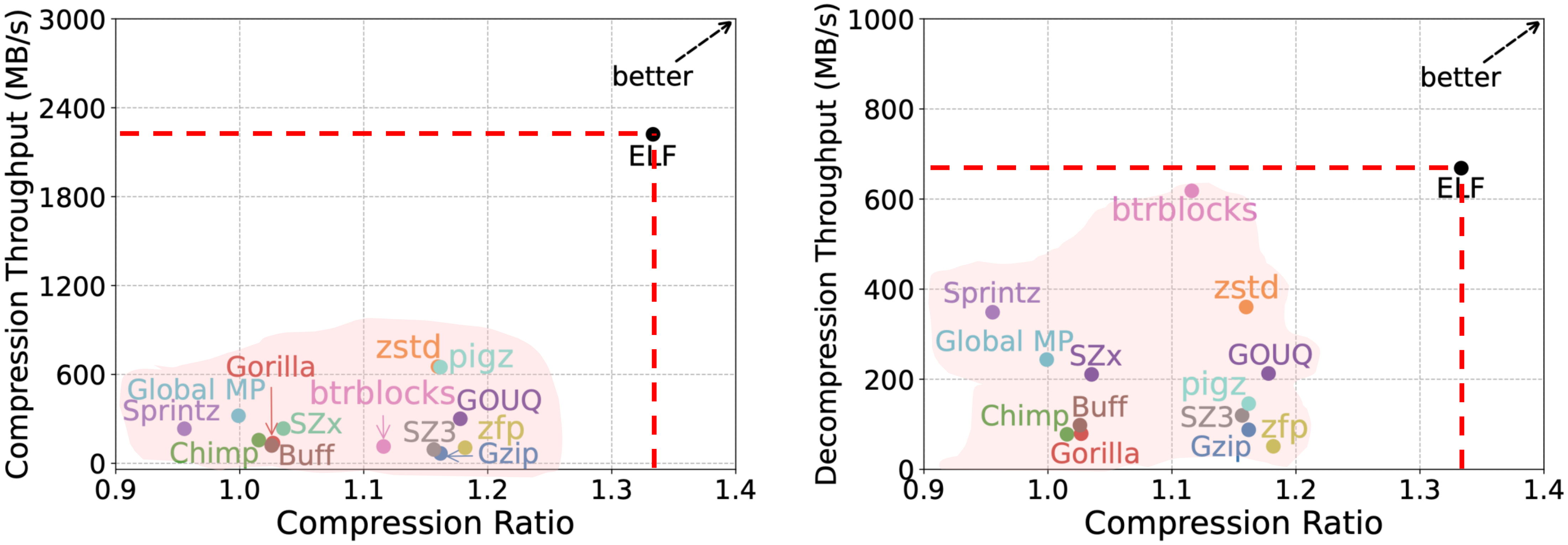


Compression and Decompression Speed



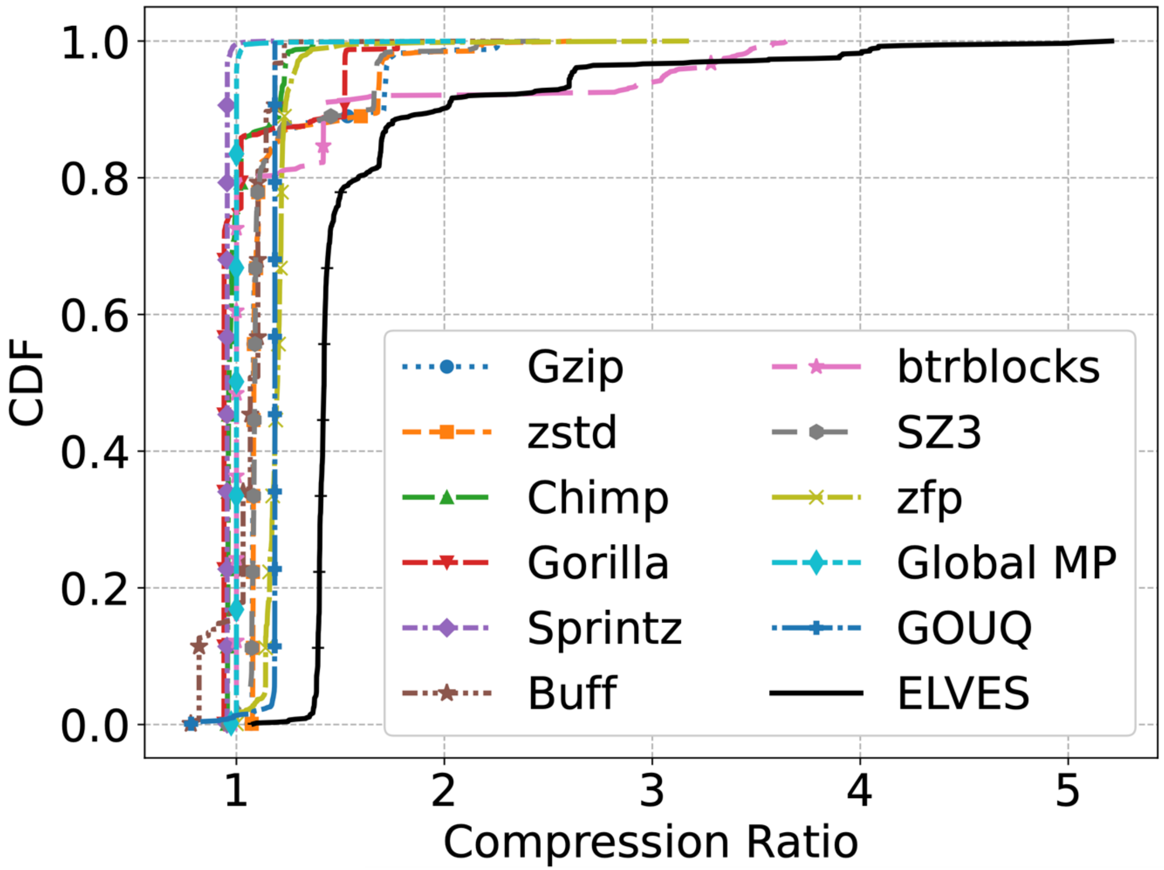


Compression and Decompression Speed



ELF is the fastest compressor, outperforming all other 14 baselines, while achieving highest compression ratio

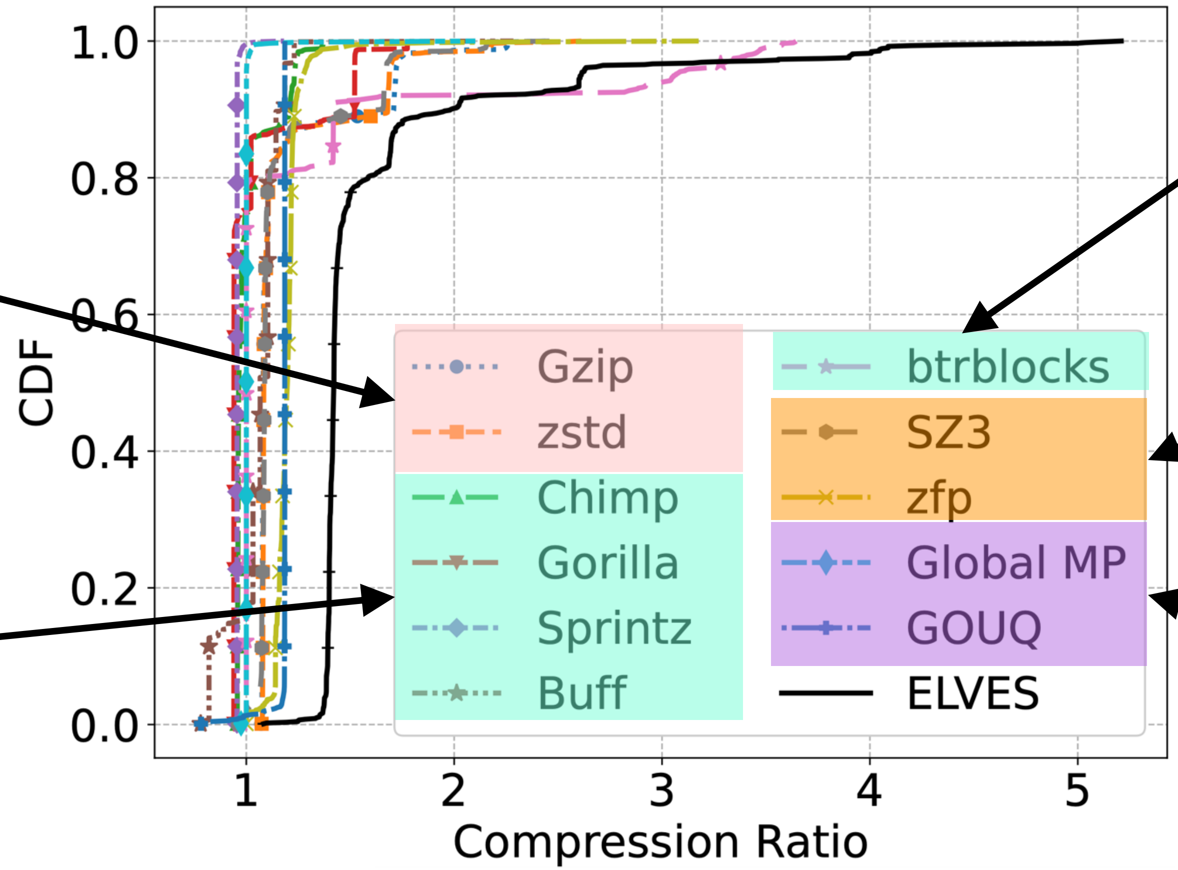
Compression Ratio



Compression Ratio

General-purpose compressors

Time series compressors

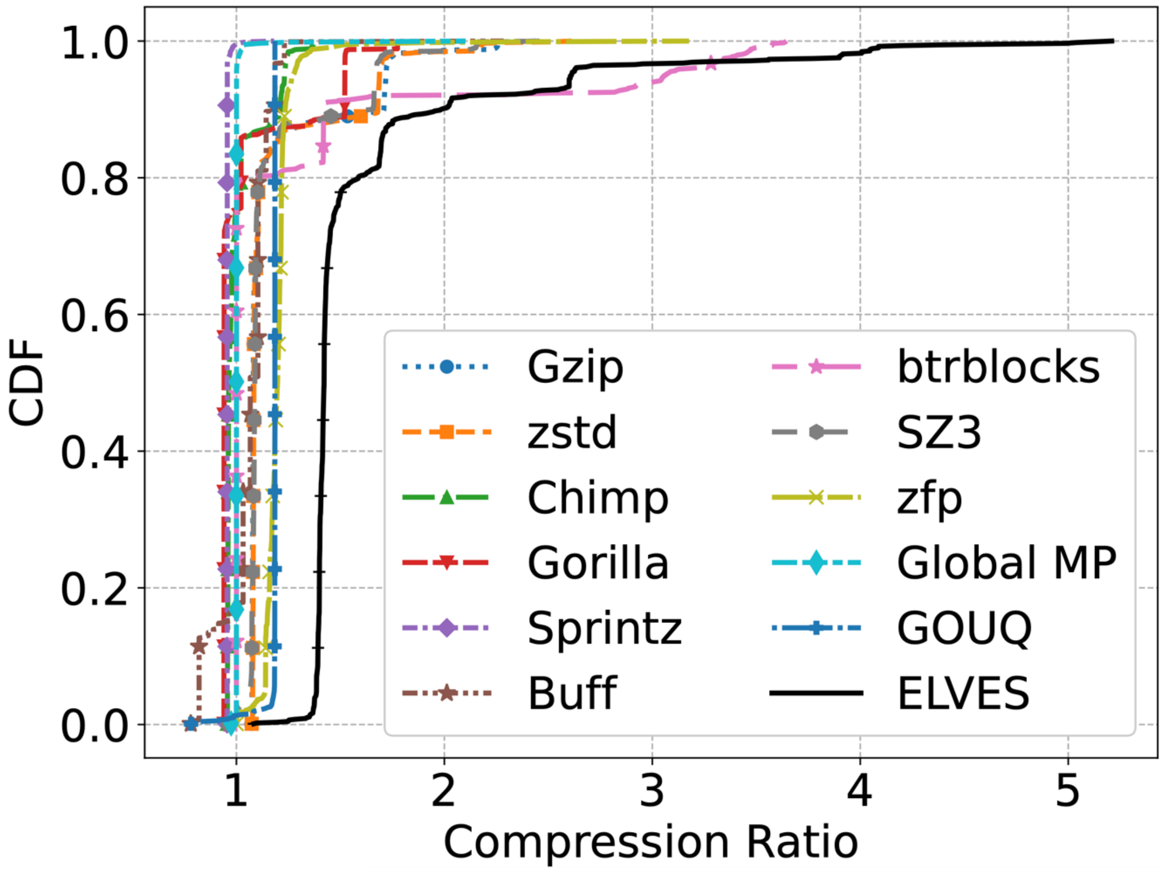


Data lake compressor

Error-bounded lossy compressors

Model pruning and quantization methods

Compression Ratio



ELVES outperforms all 11 baselines in five categories

Conclusion

- Existing and SOTA data reduction methods are generally ineffective for pre-trained models
- ELF exploits PTMs' data distribution and floating-point arithmetic properties
  - **Simple yet effective:** higher compression ratio than SOTA baselines
  - **Highly parallelizable:** superior compression and decompression speed
- ELVES integrates ELF and other data reduction methods for offline PTM storage compression

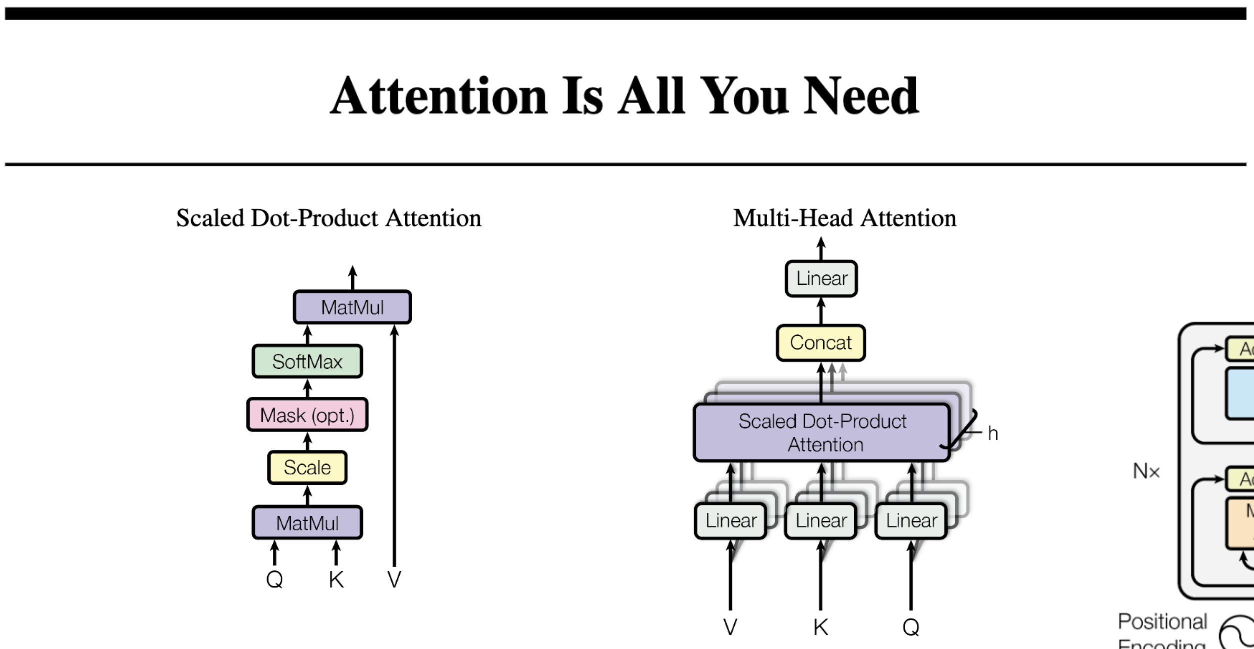
LLM Systems – from Training to Serving

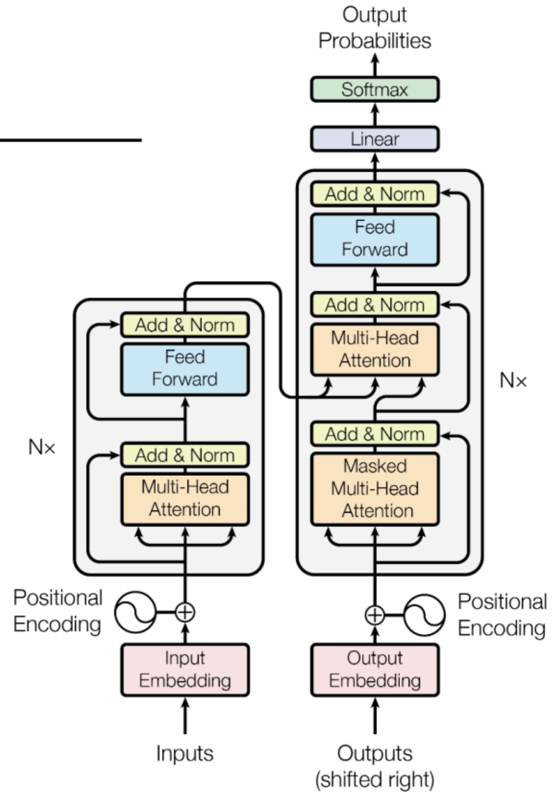
- What Is an LLM – The Model Itself
- Training – Brilliant Ideas, Tremendous Costs
- Serving – Optimized for Every User
- Now You Know the Internals – How to Use LLMs Wisely

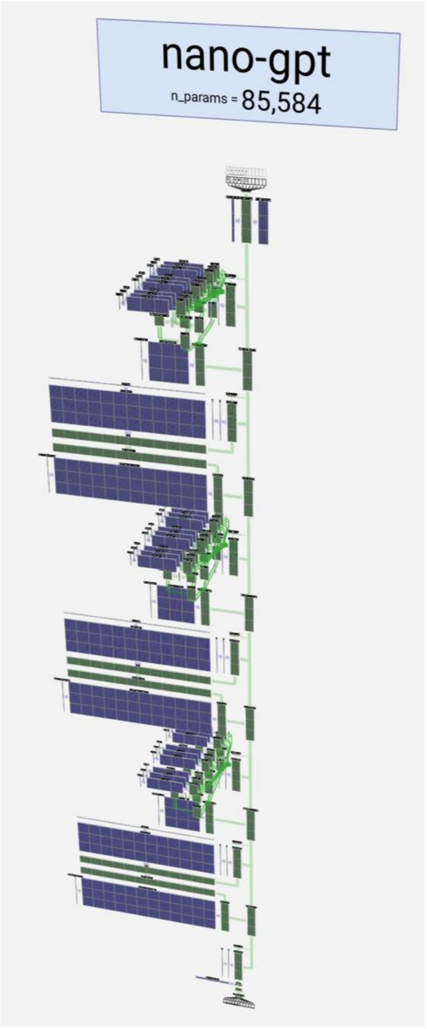
Some material taken/derived from:
- LLM Visualization (https://bbycroft.net/llm)
- Attention Is All You Need (https://arxiv.org/pdf/1706.03762)
- LLM tutorial videos from Andrej Karpathy (https://karpathy.ai/)

What Is an LLM – The Model Itself

- Transformer Architecture (https://bbycroft.net/llm)
- Self-Attention Mechanism (https://arxiv.org/pdf/1706.03762)

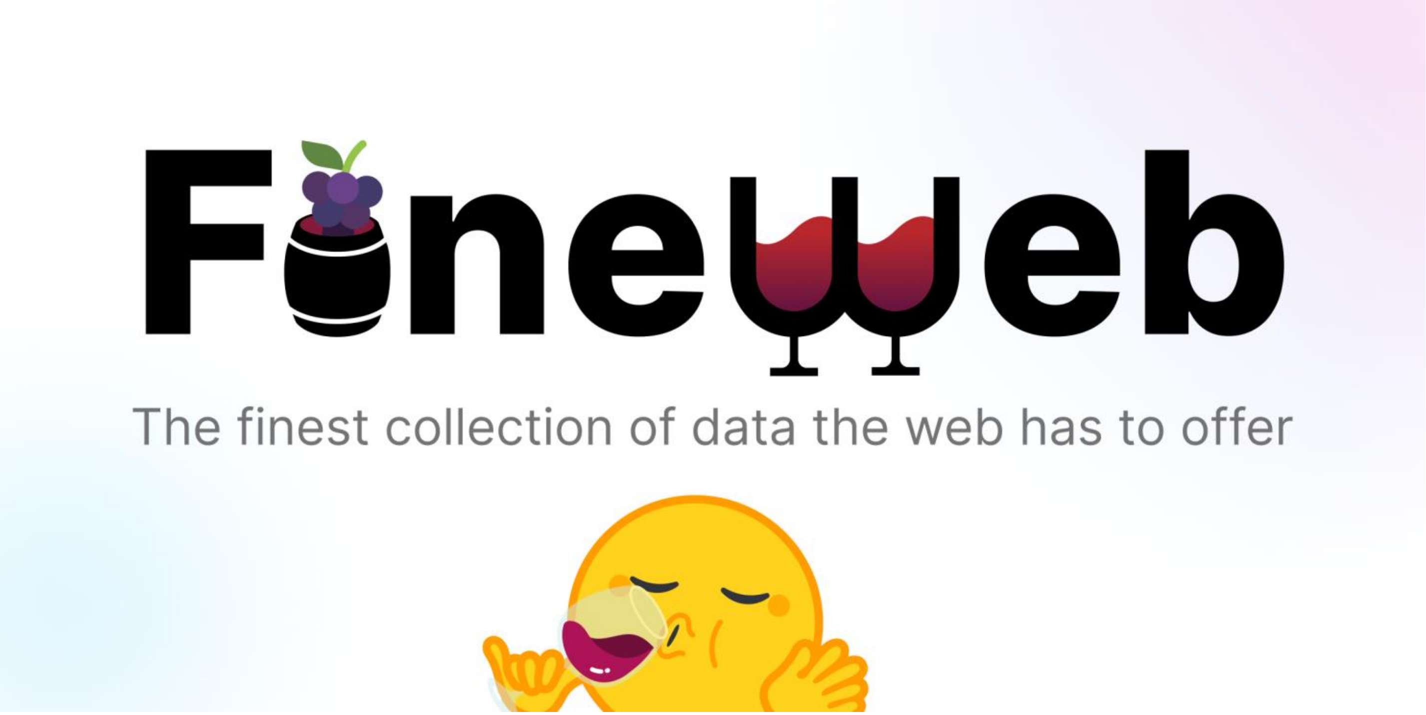






Training – Brilliant Ideas, Tremendous Costs

- Preparing Massive Datasets (text, code, filtered web)
  - Web crawlers
  - Text dataset (https://huggingface.co/datasets/HuggingFaceFW/fineweb)

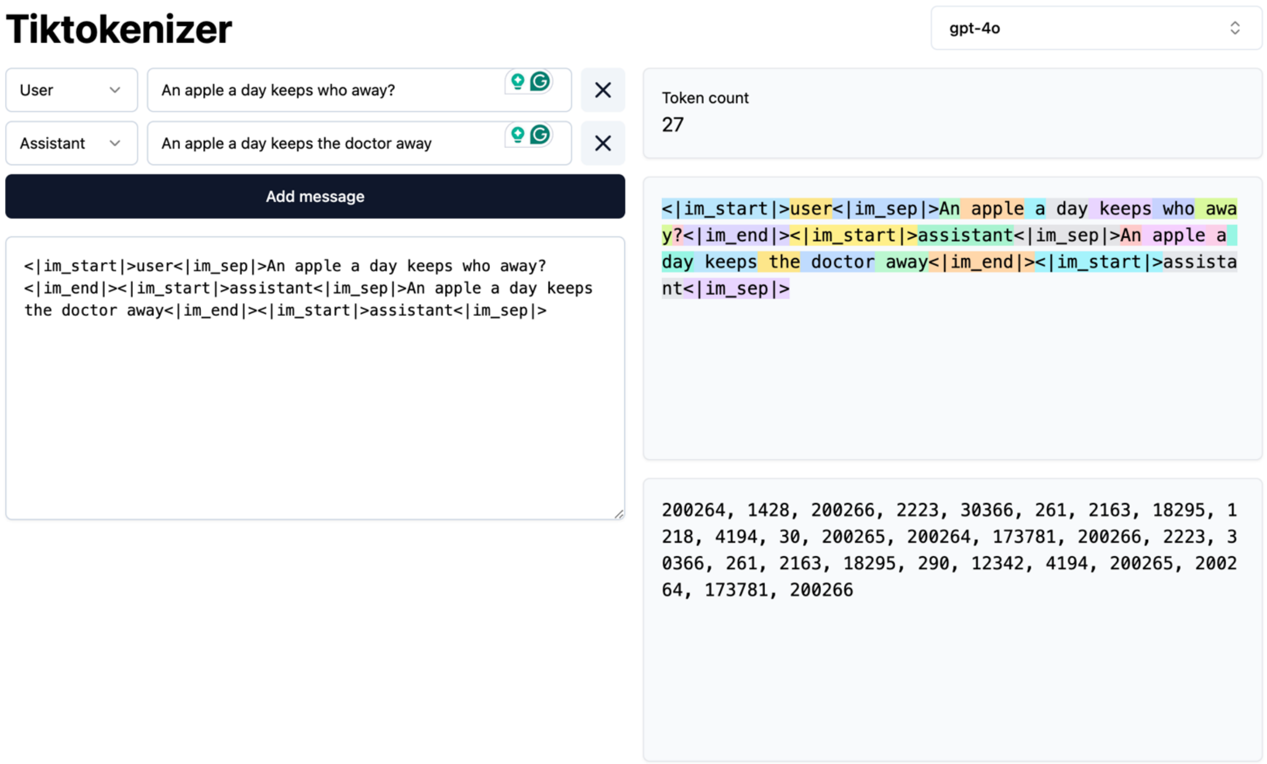


Training – Brilliant Ideas, Tremendous Costs

- Preparing Massive Datasets (text, code, filtered web)
  - Web crawlers
  - Text dataset (https://huggingface.co/datasets/HuggingFaceFW/fineweb)
- Pretraining (masked LM, batched, very expensive)
  - Tokenization (https://tiktokenizer.vercel.app/)
    - An apple a day --> [2223, 30366, 261, 2163]
  - Embedding (https://projector.tensorflow.org/,)
    - apple -> 30366 -> [1.8616e-03, -3.3722e-03, ..., 2.5787e-03, -3.9368e-03] (4096)
  - Training by predicting next token (An apple a day keeps who away?)
    - An apple a day -> keeps -> the -> {doctor(low loss), dog(high loss) -> away

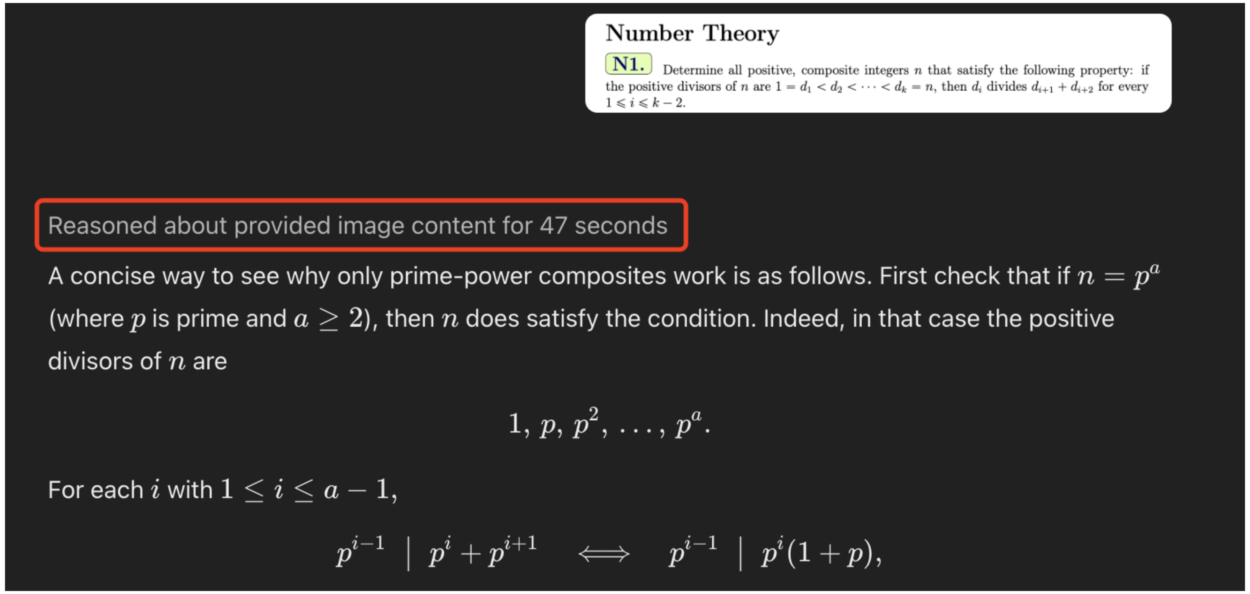
Training – Brilliant Ideas, Tremendous Costs

- Preparing Massive Datasets (text, code, filtered web)
- Pretraining (masked LM, batched, very expensive)
- Post-Training (where the magic happens)
  - Supervised fine-tuning (SFT): Turning the model into a helpful assistant



Training – Brilliant Ideas, Tremendous Costs

- Preparing Massive Datasets (text, code, filtered web)
- Pretraining (masked LM, batched, very expensive)
- Post-Training (where the magic happens)
  - Supervised fine-tuning (SFT): Turning the model into a helpful assistant
  - Reinforcement Learning (RL): Teaching the model to behave and let it create

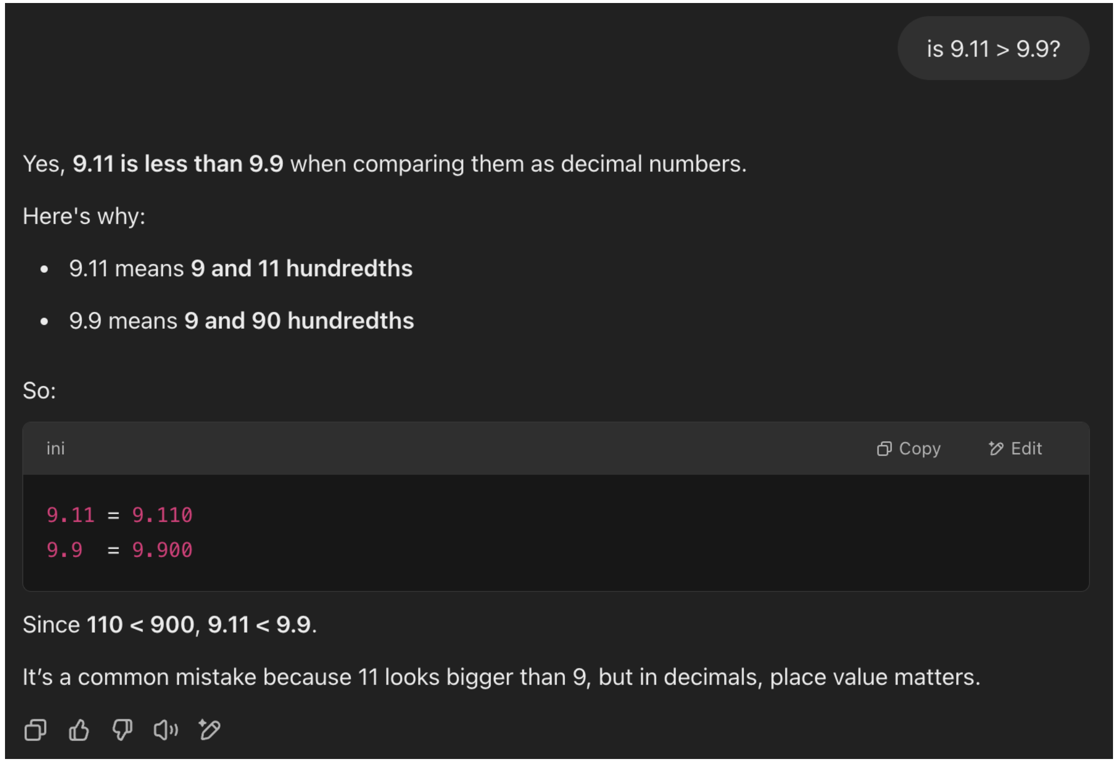


Serving – Optimized for Every User

- Request Batching for Throughput
  - Group multiple user prompts to maximize GPU efficiency and reduce idle time.
- KV Caching for Fast Decoding
  - Store intermediate attention states to avoid redundant computation during generation.
- Prefill-Decoding Disaggregation
  - Split heavy first-token processing from fast token generation for better parallelism.
- Model Compression (Quantization, Distillation)
  - Shrink model size and speed up inference while maintaining accuracy.

How to Use LLMs Wisely

- Now You Know the Internals – How to Use LLMs Wisely
  - Prompting Tips (Few-shot, Chain-of-Thought)
    - What is the results of 234568 * 24432 / 9876? (Fast, but may not be correct. Give them more intermediate steps.)
    - Let's solve this step by step, write the solution process of the question of 234568 * 24432 / 9876. OR Please write python script to solve the question ….



How to Use LLMs Wisely

- Now You Know the Internals – How to Use LLMs Wisely
  - Prompting Tips (Few-shot, Chain-of-Thought)
    - What is the results of 234568 * 24432 / 9876? (Fast, but may not be correct. Give them more intermediate steps.)
    - Let’s solve this step by step, write the solution process of the question of 234568 * 24432 / 9876. OR Please write python script to solve the question ….
  - Hallucinations & Limitations
    - Give me sources that support the claim that coffee prevents cancer. (“According to a 2015 study published in the Journal of Coffee Research...”. But the journal and study don’t exist. -- It may have learned this style from conversations during fine-tuning)
    - Cite sources with URLs or DOIs; And double check the results.
  - Chatbots vs APIs
    - Leveraging chatbots and APIs in different scenarios

LLM Storage Compression
&
LLM Systems – from Training to Serving

DS 5110: Big Data Systems

Spring 2025

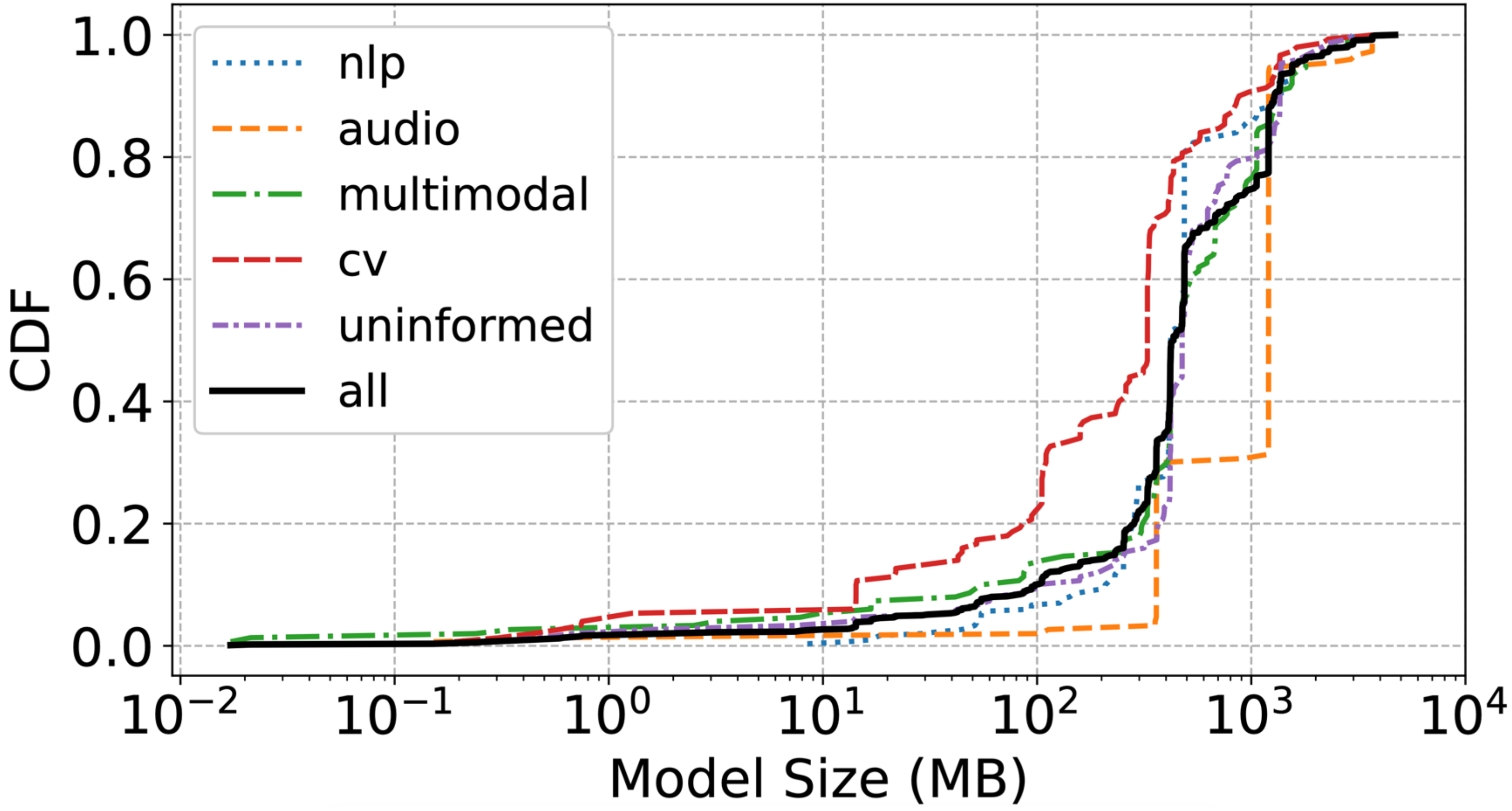
Lecture 15

Zhaoyuan Su

University of Virginia

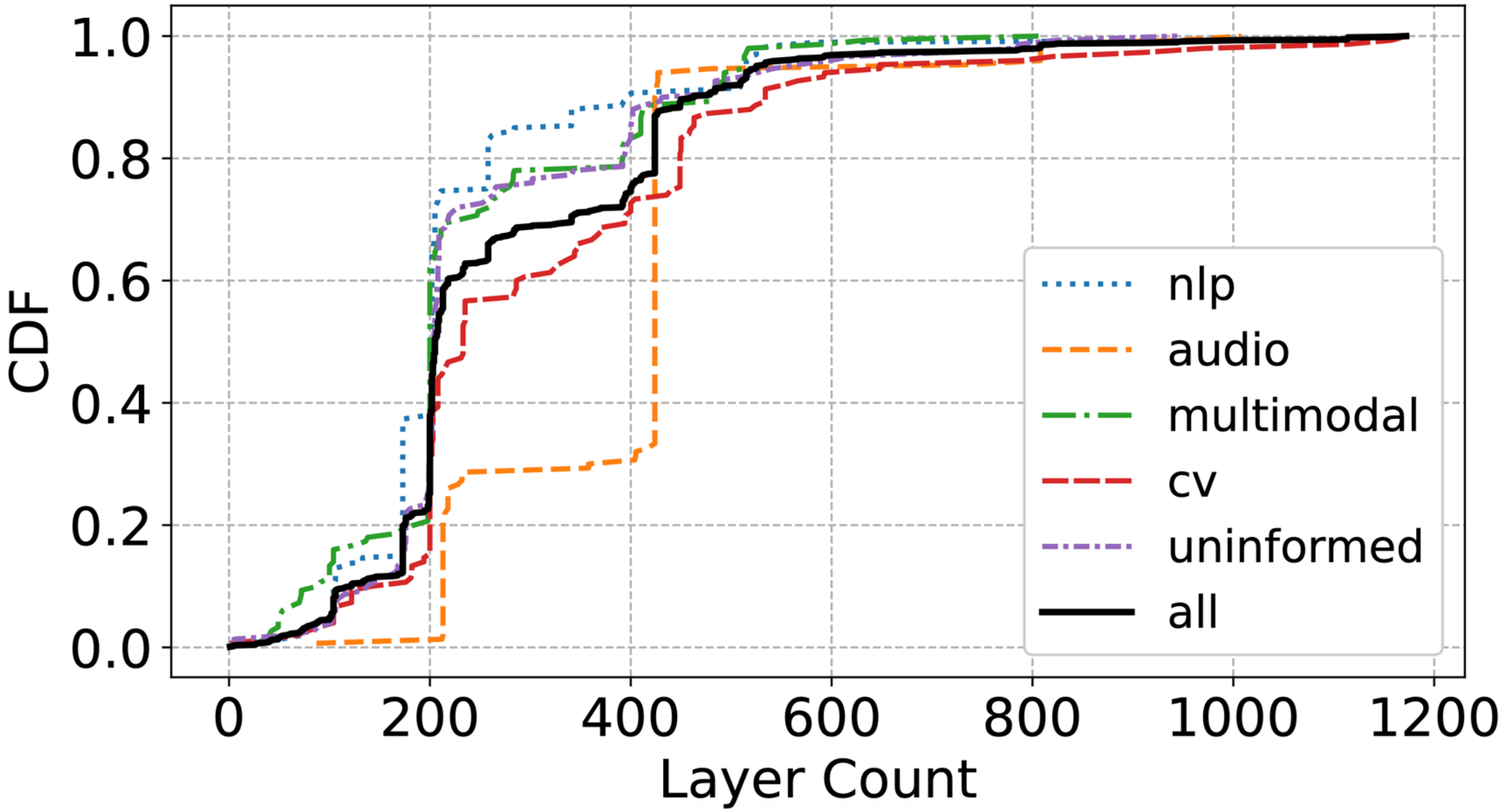
Backup Slides

Model Sizes



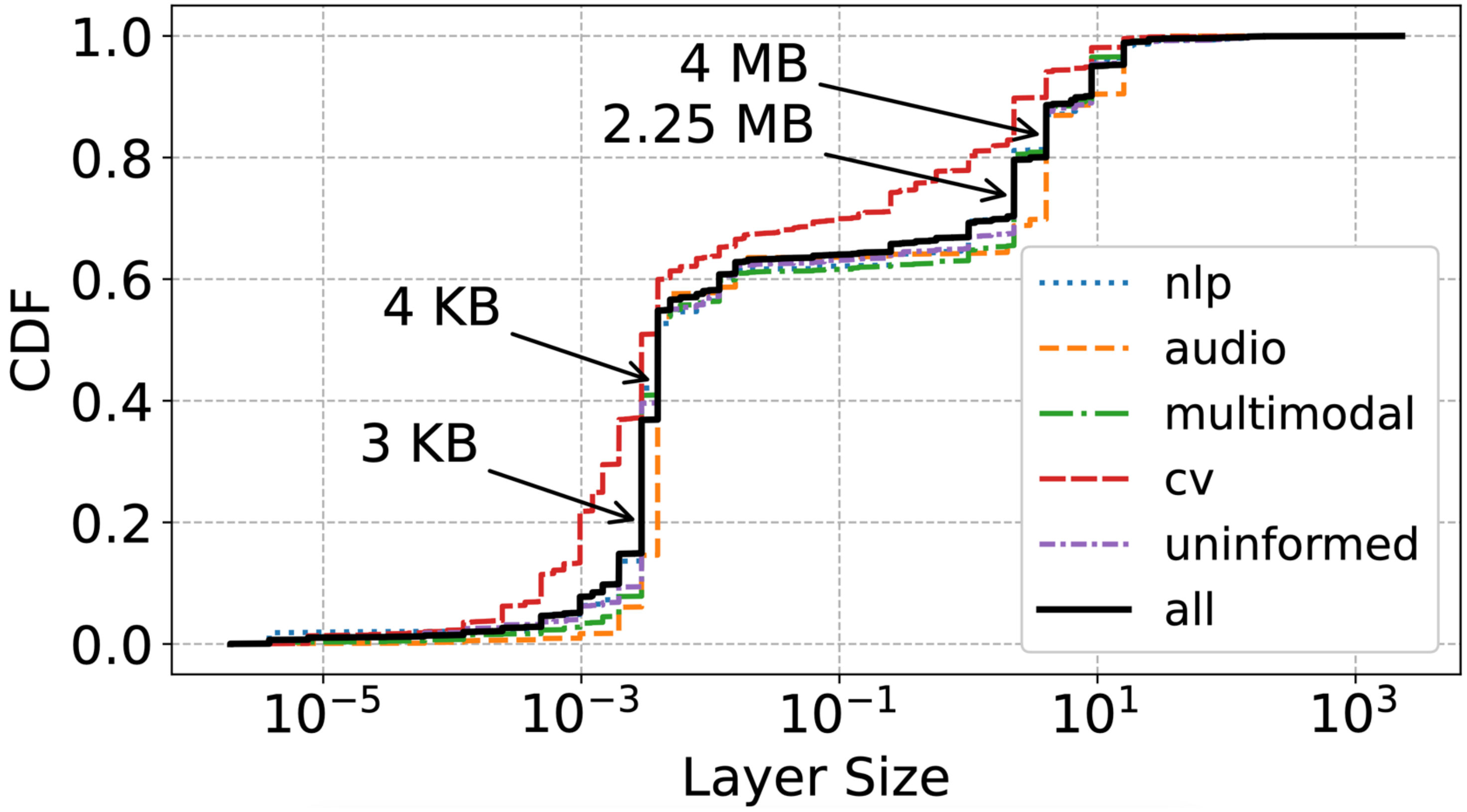
PTM sizes are generally **large**, with **90%** of models exceeding **100 MB**, and **25.22%** surpassing **1 GB**.

Model Layer Counts



PTMs tend to be **deep**, with approximately **75%** of models having over **200 layers**, and **audio** models stand out, with **70%** containing more than **400 layers**.

Model Layer Sizes



PTM layer sizes show a **step-like** distribution, with **57.84%** of sizes clustered around **3 KB**, **4 KB**, **2.25 MB**, and **4 MB**.

Would Layer-level Dedup Help?

Layer Type	Count	Dup %	Total Sz in GB	Dup Sz in GB (%)
float32	240,966	8.35%	557.84	30.14 (5.40%)
float16	4,018	3.61%	14.51	0.14 (0.96%)
float64	199	0%	0.81	0 (0%)
uint8	1,597	99.81%	1.75	1.74 (99.43%)
int64	1,765	96.77%	0.97	0.94 (96.91%)
Overall	248,545	9.48%	575.88	32.96 (5.72%)

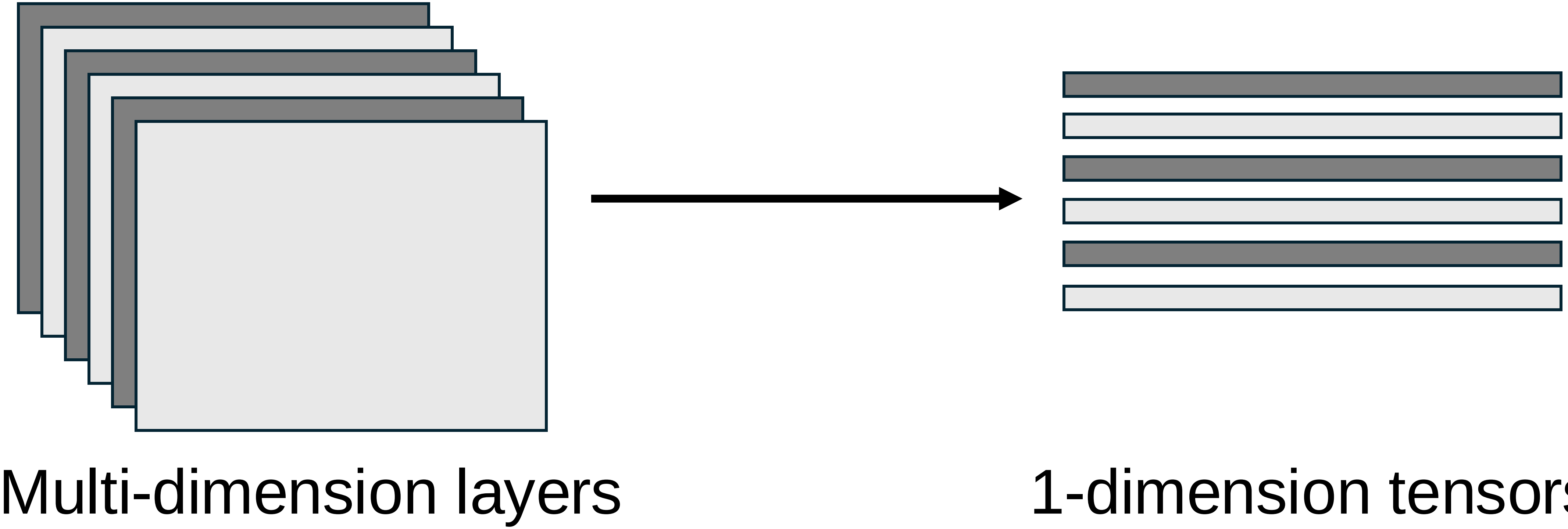
Would Layer-level Dedup Help?

Layer Type	Count	Dup %	Total Sz in GB	Dup Sz in GB (%)
float32	240,966	8.35%	557.84	30.14 (5.40%)
float16	4,018	3.61%	14.51	0.14 (0.96%)
float64	199	0%	0.81	0 (0%)
uint8	1,597	99.81%	1.75	1.74 (99.43%)
int64	1,765	96.77%	0.97	0.94 (96.91%)
Overall	248,545	9.48%	575.88	32.96 (5.72%)

The result of hash-based dedup is discouraging – with only 5.72% of storage footprint attributed to duplicate layers

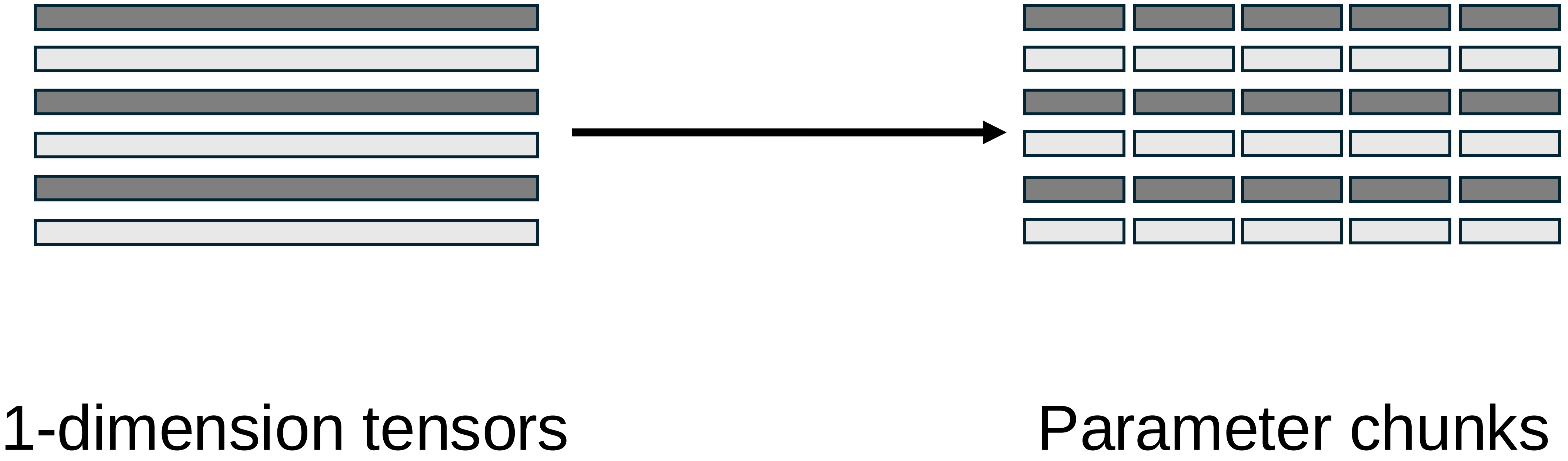
ELF Compression

1. Flatten FP layers into 1D tensors



ELF Compression

2. Split tensors into multi chunks to enable parallel processing.



ELF Compression

3.1 Save parameter as it is for $|p_j| \geq 1$

$[p_1, p_4, ..., p_j]$
Floating points

1%

$[1, 4, ..., j]$
Position array

$[p_0, p_1, p_2, p_3, p_4, ..., p_n]$
Parameter chunk

ELF Compression

3.2. Perform ELF for $p_i \in (-1,1)$

$[p_0, p_1, p_2, p_3, p_4, \ldots, p_n]$

Parameter chunk

1%

$[p_1, p_4, \ldots, p_j]$

Floating points

$[1, \quad 4, \ldots, \quad j]$

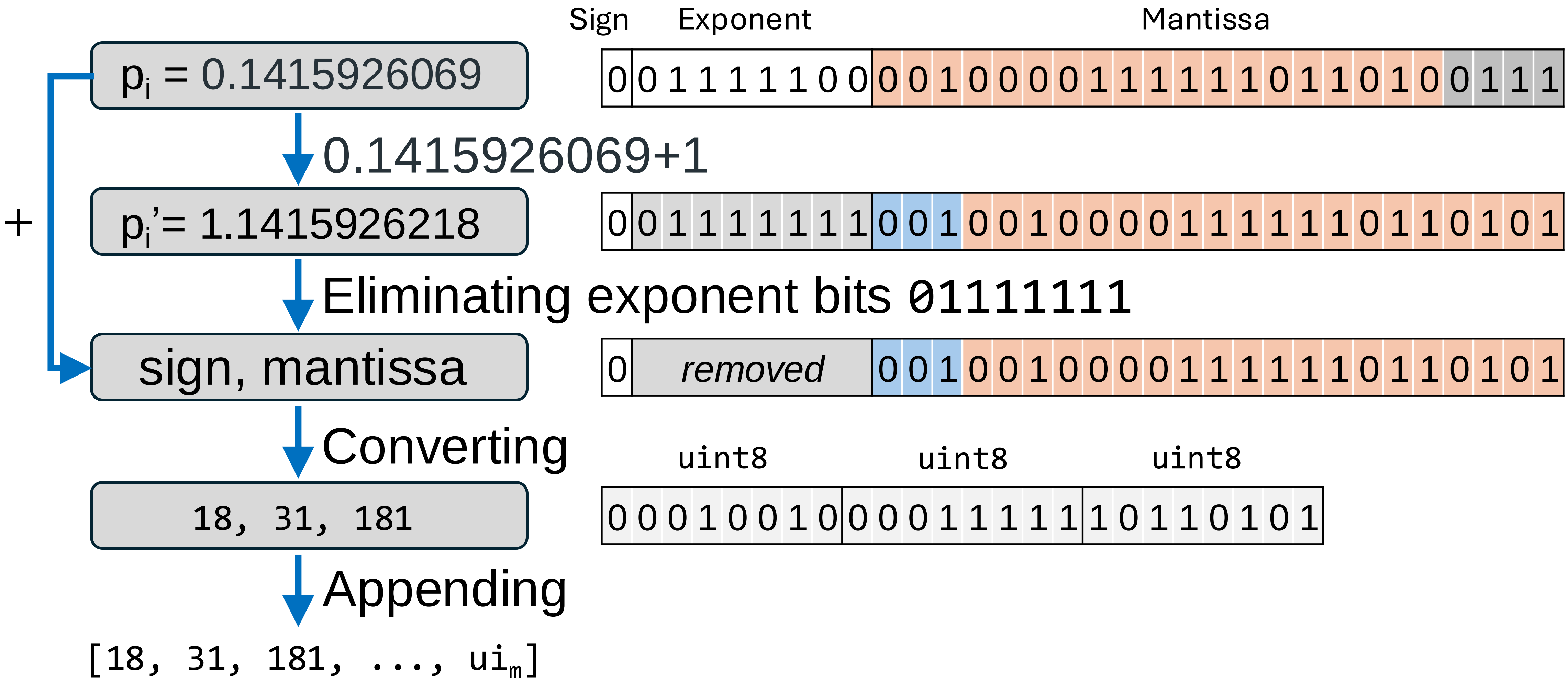
Position array

99%

$[ui_0, ui_1, \ldots, {}_m]$

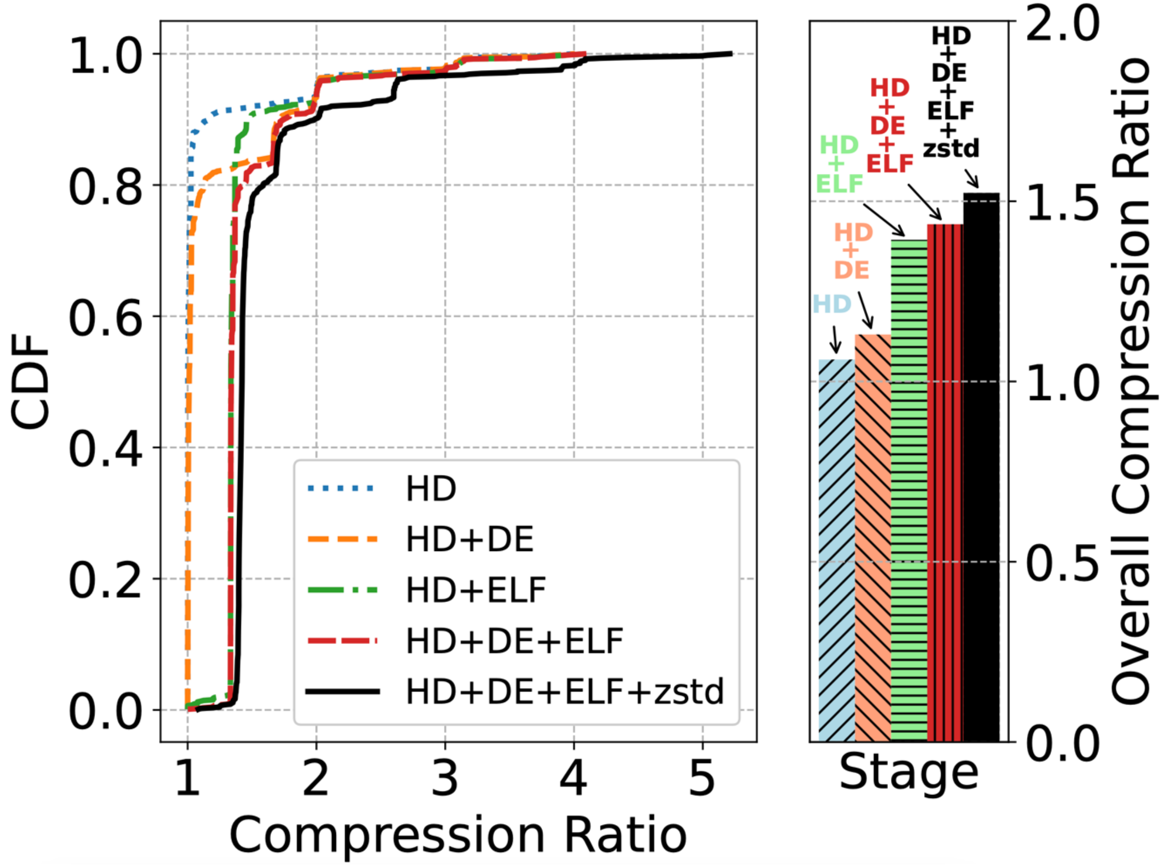
`uint8` array

ELF Example



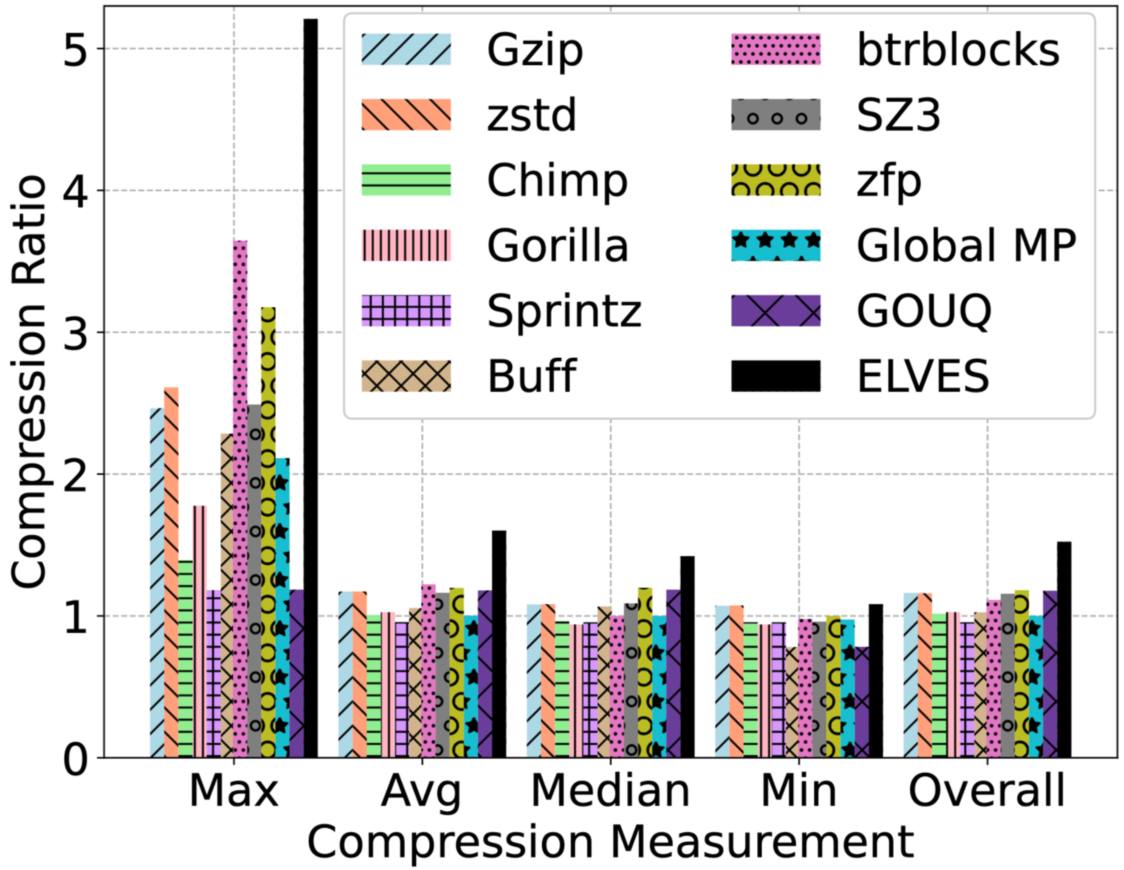
Sign
Exponent
Mantissa
p_i = 0.1415926069
0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 1 1
0.1415926069+1
+
p_i'= 1.1415926218
0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1
Eliminating exponent bits 01111111
sign, mantissa
0 *removed* 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1
Converting
uint8
uint8
uint8
18, 31, 181
0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1
Appending
[18, 31, 181, ..., ui_m]

Evaluating ELVES Stages



ELF contributes the largest (65%) to the compression ratio improvement across all stages

Compression Ratio Breakdown



Quantifying Accuracy Impact

Model Task (Category)	Count (%)	Accuracy Degradation
Image Classification (CV)	69 (23.00%)	0.87%
Text Generation (NLP)	68 (22.67%)	0%
Text Classification (NLP)	60 (20.00%)	0%
Token Classification (NLP)	30 (10.00%)	0%
Translation (NLP)	25 (8.33%)	0.4%
Question Answering (NLP)	24 (8.00%)	0%
Audio Classification (Audio)	9 (3.00%)	0%
Summarization (NLP)	9 (3.00%)	1.11%
Speech Recognition (Audio)	6 (2.00%)	0%
Overall	**300 (100%)**	**0.27%**

ELVES achieves a 0% accuracy degradation in 6 out of 9 model prediction tasks for 300 sampled PTMs

Quantifying Accuracy Impact

Domain	Task(# of tested model)	Dataset	Accuracy Degradation							
			ELVES	SZ3	zfp	mp	mp2e	gouq	gouq2e	half
CV	image classification(4)	mini_imagenet	0.2%	0.3%	0.2%	0.1%	0.2%	0.4%	1.1%	65.0%
		cifar100	0.2%	0.3%	0.1%	0.2%	0.2%	0.4%	1.2%	48.4%
	object detection(4)	detection-datasets/coco	0.1%	0.2%	0.2%	0.1%	0.2%	0.2%	0.2%	1.6%
		cppe-5	0.2%	0.3%	0.2%	0.2%	0.3%	0.2%	0.3%	2.6%
	image segmentation(6)	scene_parse_150	0.2%	0.6%	0.4%	0.1%	0.2%	0.2%	0.8%	38.6%
		sidewalk-semantic	0.3%	1.4%	0.5%	0.2%	0.3%	0.2%	0.7%	35.1%
Multimodal	feature extraction(7)	Open-Orca/OpenOrca	0.1%	0.2%	0.1%	0.1%	0.1%	0.2%	0.3%	18.1%
		imdb-movie-reviews	0.1%	0.1%	0.1%	0.1%	0.1%	0.2%	0.5%	24.5%
	image-to-text(4)	conceptual_captions	0%	0%	0%	0%	0%	0%	0%	0%
		red_caps	0%	0%	0%	0%	0%	0%	0%	0%
Audio	speech recognition(5)	librispeech_asr_dummy	0%	0%	0%	0%	0%	0%	0%	0%
		lj_speech	0%	0%	0%	0%	0%	0%	0%	0%
NLP	sentiment classification(7)	glue-sst2	0%	0%	0%	0%	0%	0%	0%	0%
		imdb	0%	0%	0%	0%	0%	0%	0%	0%
	sentence similarity(5)	glue-stsb	0%	0%	0%	0%	0.1%	0.1%	0.2%	3.6%
		paws-x	0%	0%	0%	0%	0.1%	0.1%	0.2%	4.2%
	Fill-mask(4)	wikitext	0%	0%	0%	0%	0.1%	0.1%	0.1%	0.1%
		ptb_text_only	0%	0%	0%	0%	0.1%	0.1%	0.1%	0.1%
Overall AD (Overall CR)			0.07% (1.52)	0.18% (1.16)	0.1% (1.18)	0.06% (1.00)	0.22% (1.01)	0.13% (1.18)	0.32% (1.20)	13.44% (1.99)

ELVES achieves both low accuracy degradation and high compression ratio for all 9 tasks spanning 18 benchmark datasets