Waiting to Decompress

The Economics of LLM-Based Compression

Andreas Kipf, Tobias Schmidt*, Ping-Lin Kuo, Skander Krid, Moritz Rengert, Luca Heller,
Andreas Zimmerer, Mihail Stoian, Varun Pandey, Alexander van Renen

University of Technology Nuremberg, *Technical University of Munich

CIDR 2026, Chaminade, USA

60% of enterprise data is classified as cold when it's created

IDC Worldwide Global StorageSphere Forecast

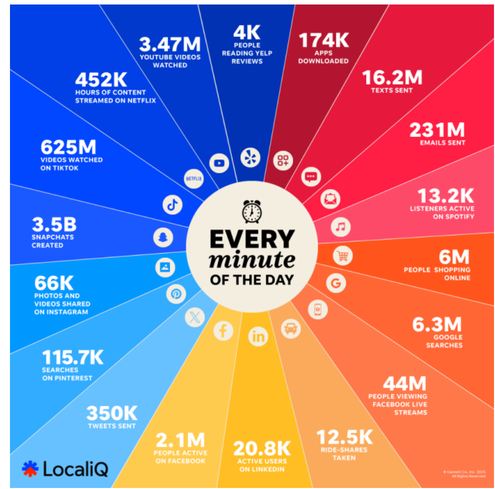
Can AI help us save data storage costs?

The paradox of using $1M models to save $20/month on storage

The Problem

We Keep Everything!

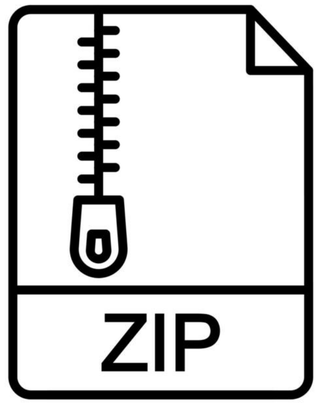
- We generate **massive amounts** of data, but most of it is “**cold**”
- We store data for compliance or backup, but we rarely ever access it
- Storage costs can be substantial (**$200k/year for 1 PiB on S3**)



The Current Solution

Zip It!

- Systems utilize **block-based compression** like Zstd, Snappy, or LZ4
- They are fast, cheap, and run on standard CPUs
- However, they have **hit a wall** in terms of compression ratios



The Idea
AI To The Rescue?

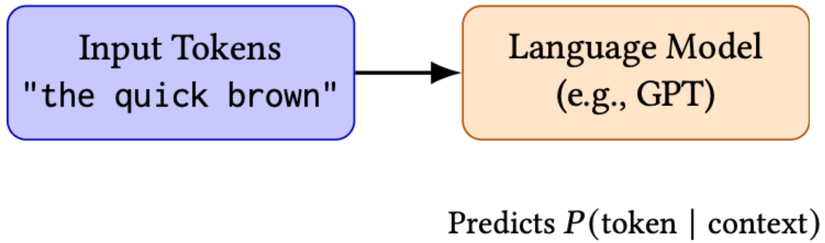
- Researchers started looking into using LLMs for data compression
- LLMs are incredibly good at predicting text. If they can **predict your data**, they can **compress it better** than any traditional tool.
- Assumption: We aren’t considering the LLM’s storage footprint, as the model is independent of the specific dataset

Language Modeling is Compression

Google DeepMind, ICLR 2024

- Advocates for using **lossless** compression to study foundation models
- Covers text, images, and speech data
- Finds that foundation models (Chinchilla 70B) outperform domain-specific compressors on all three modalities

Compression



Extract logits and apply softmax to obtain **probabilities**

Probability distribution over the **150k possible tokens**

Encode actual token using **entropy coding**

Entropy Coding

- **Huffman Coding**: Fast but limited to power-of-two bits per token
- **Arithmetic Coding**: Achieves near-optimal compression by encoding an entire data sequence into a single fractional value between 0 and 1
- **Asymmetric Numeral Systems**: Combines best of both

The Reality Check

- Prior work conveniently left out **runtime and cost considerations**
- We benchmarked existing approaches and optimized them for **speed**
- Developed a **cost model** showing the full picture

Handling Nondeterminism

- **Bypass high-level APIs** (use base variants of models)
- Set **temperature to 0**
- **Fix batch size** to prevent floating-point differences due to changes in GPU execution order

Defeating Nondeterminism in LLM Inference, Thinking Machines
https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

Optimizations

- Optimize **batch** and **context window sizes**
- Enable **KV caching**: Avoid recomputing previously processed context (2.5 KiB/s to 57 KiB/s in throughput)
- **Decouple AC**: Arithmetic coding runs on the CPU (algorithm is bottlenecked by GPU inference)

Optimizations
Parallization

Data Block 0:

token0 token1 token2 token3

Data Block 1:

token4 token5 token6 token7

Batch Size = 2, Context Window Size = 2

Optimizations
Parallization

Data Block 0:

token0 token1 token2 token3

Batch 0

Data Block 1:

token4 token5 token6 token7

Batch Size = 2, Context Window Size = 2

Optimizations
Parallization

Data Block 0:

token0 token1 token2 token3

Batch 1

Data Block 1:

token4 token5 token6 token7

Batch Size = 2, Context Window Size = 2

Optimizations
Parallization

Data Block 0:

token0 token1 token2 token3

Batch 3

Data Block 1:

token4 token5 token6 token7

Batch Size = 2, Context Window Size = 2

Optimizations
Parallization

Data Block 0:

token0 token1 token2 token3

Batch 4

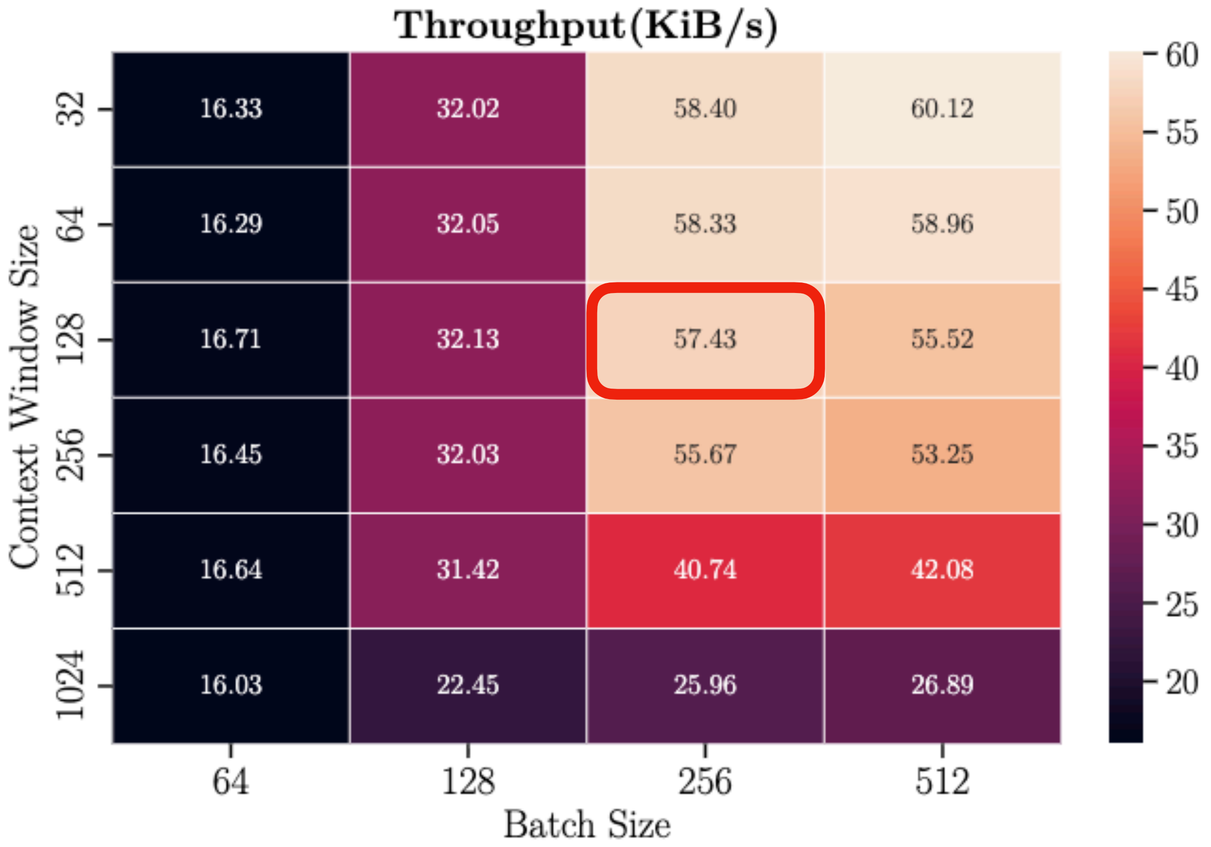
Data Block 1:

token4 token5 token6 token7

Batch Size = 2, Context Window Size = 2

Optimizations

Effect of Batch and Context Window Sizes

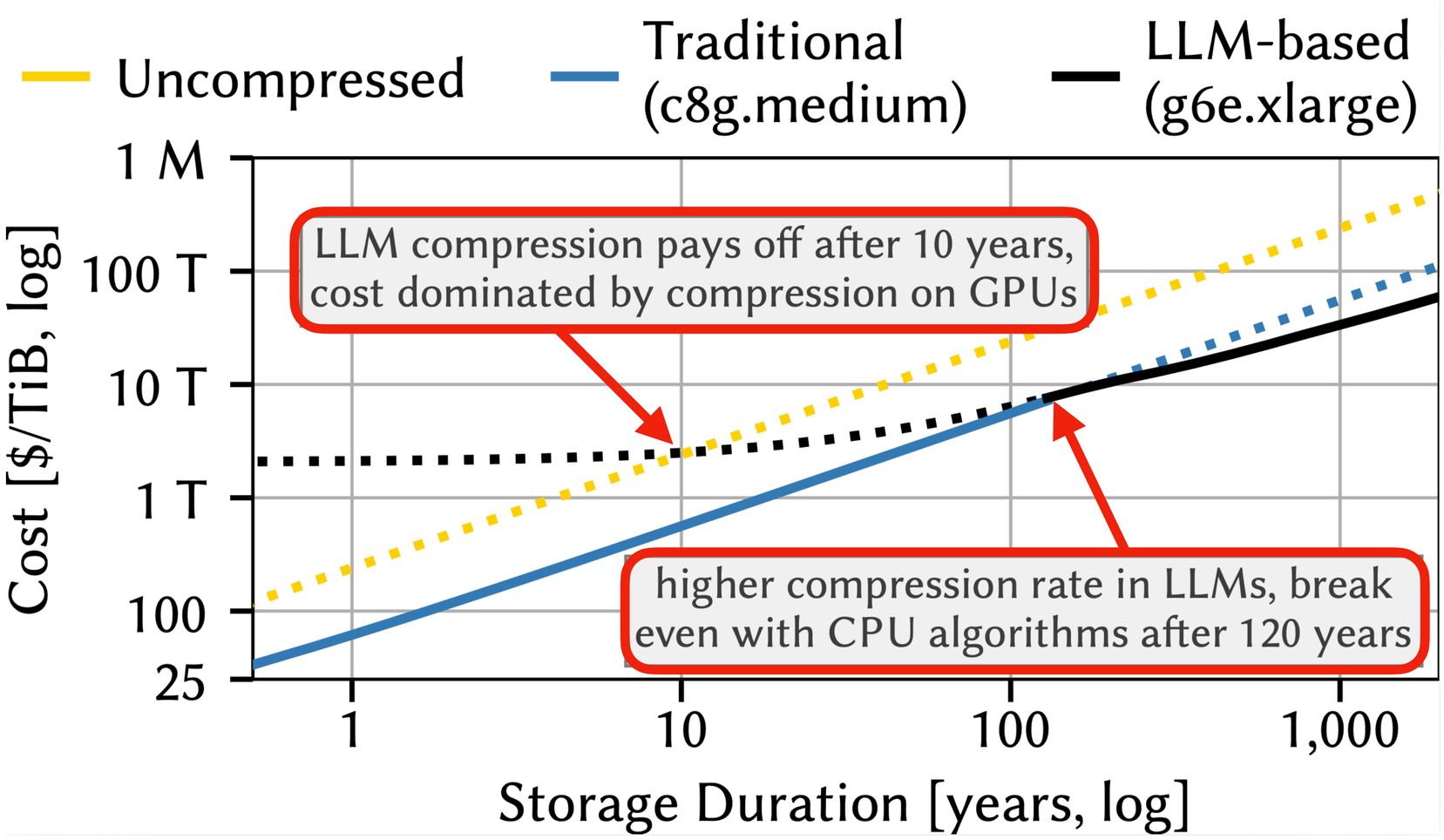


Evaluation
Setup

- Compressing Wikipedia **text** and Python **code**
- NVIDIA **L40S** GPU, $0.804 per hour (g6e.xlarge)
- **Graviton** instance $0.019 per hour (c8g.medium)
- $20 per TiB per month (**Amazon S3**)

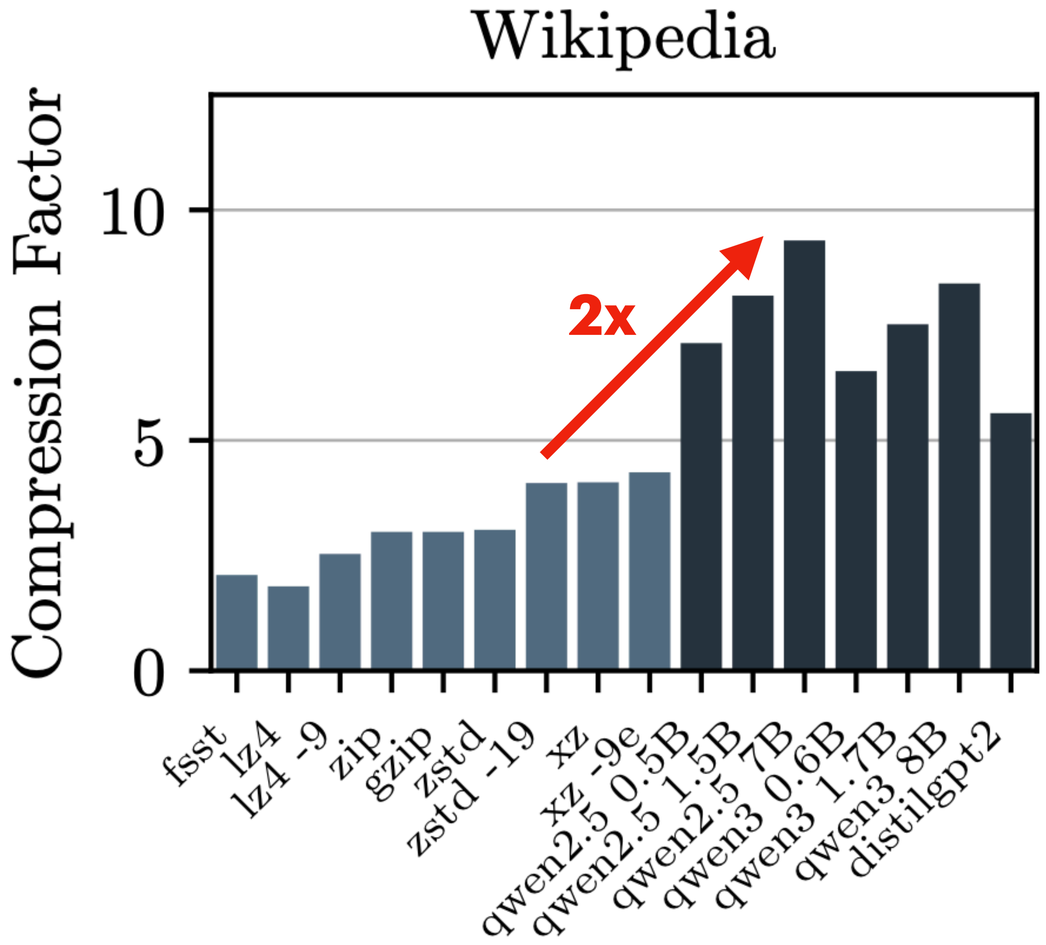
Evaluation

Cost Model



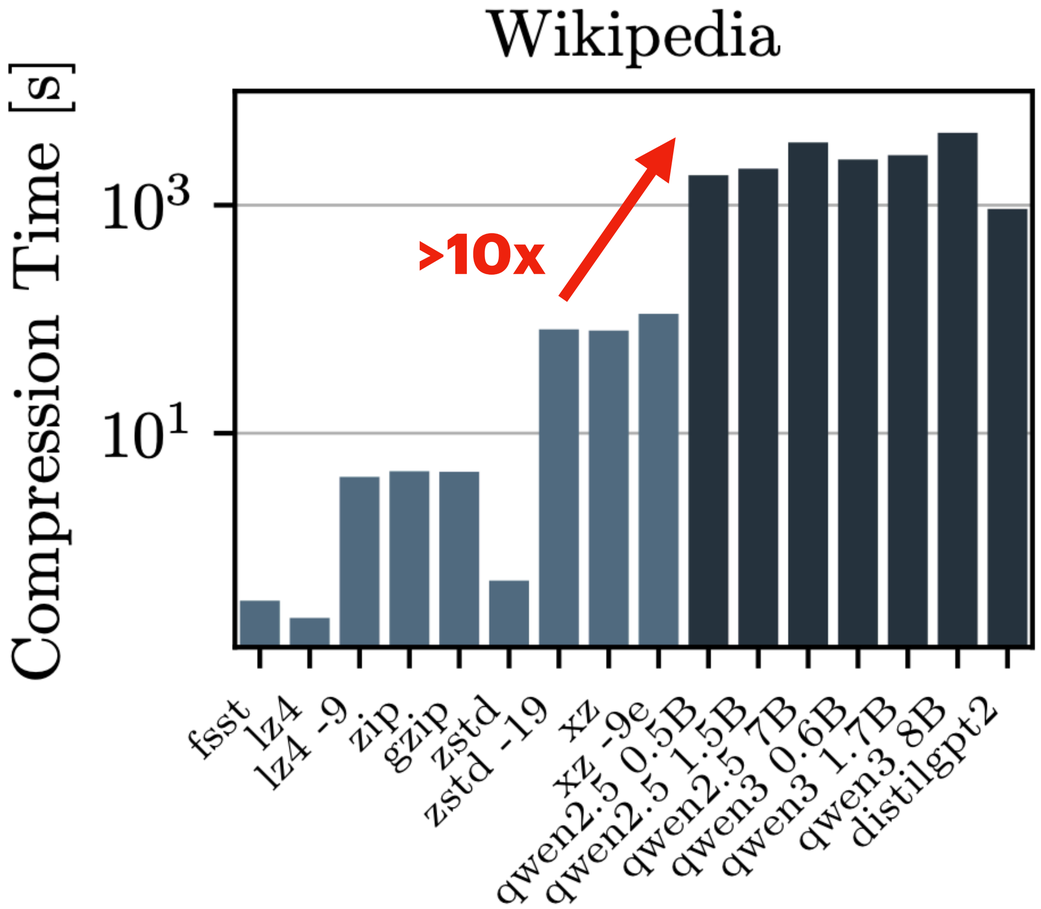
Evaluation

Compression Factor



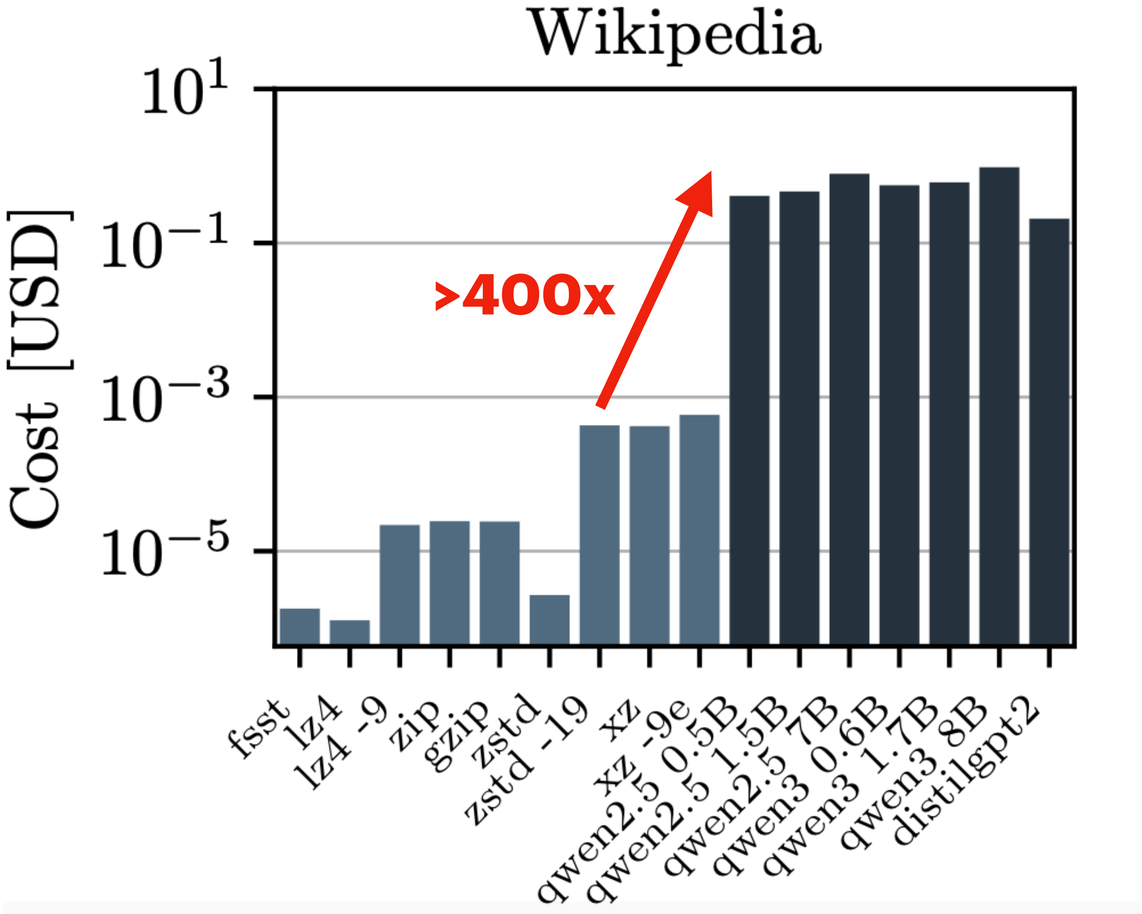
Evaluation

Compression Time



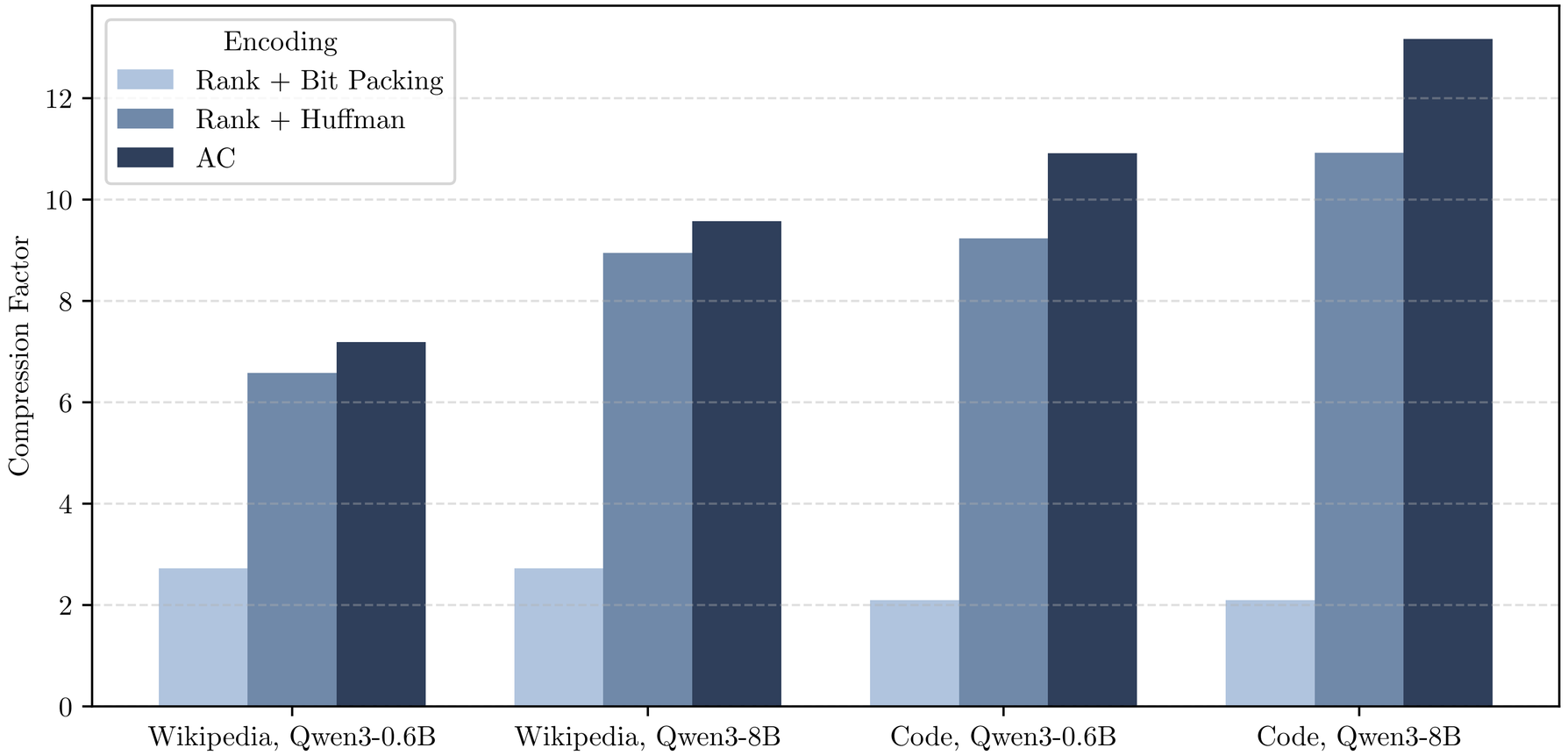
Evaluation

Compression Cost



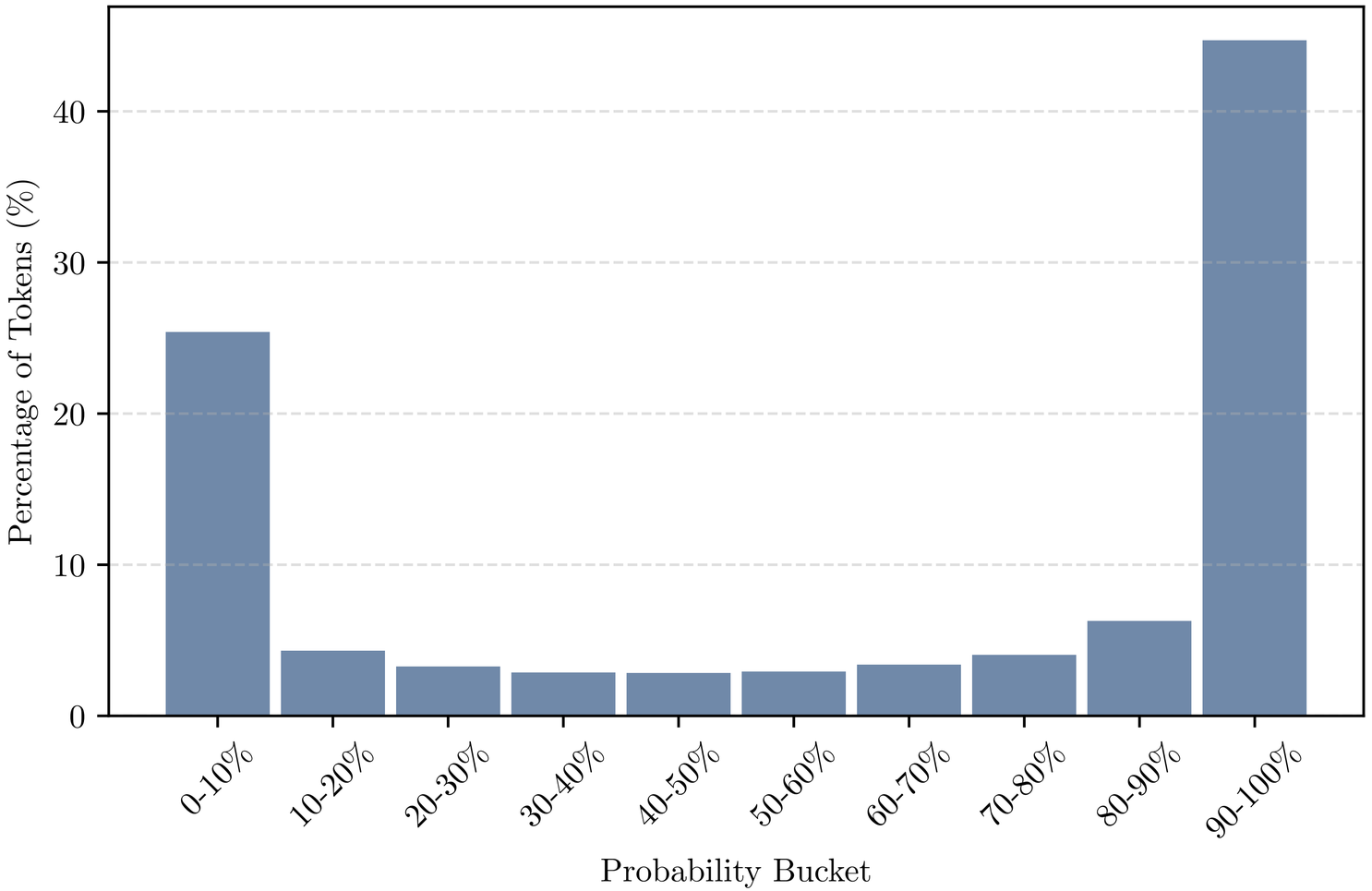
Experiments

Entropy Coding



Experiments

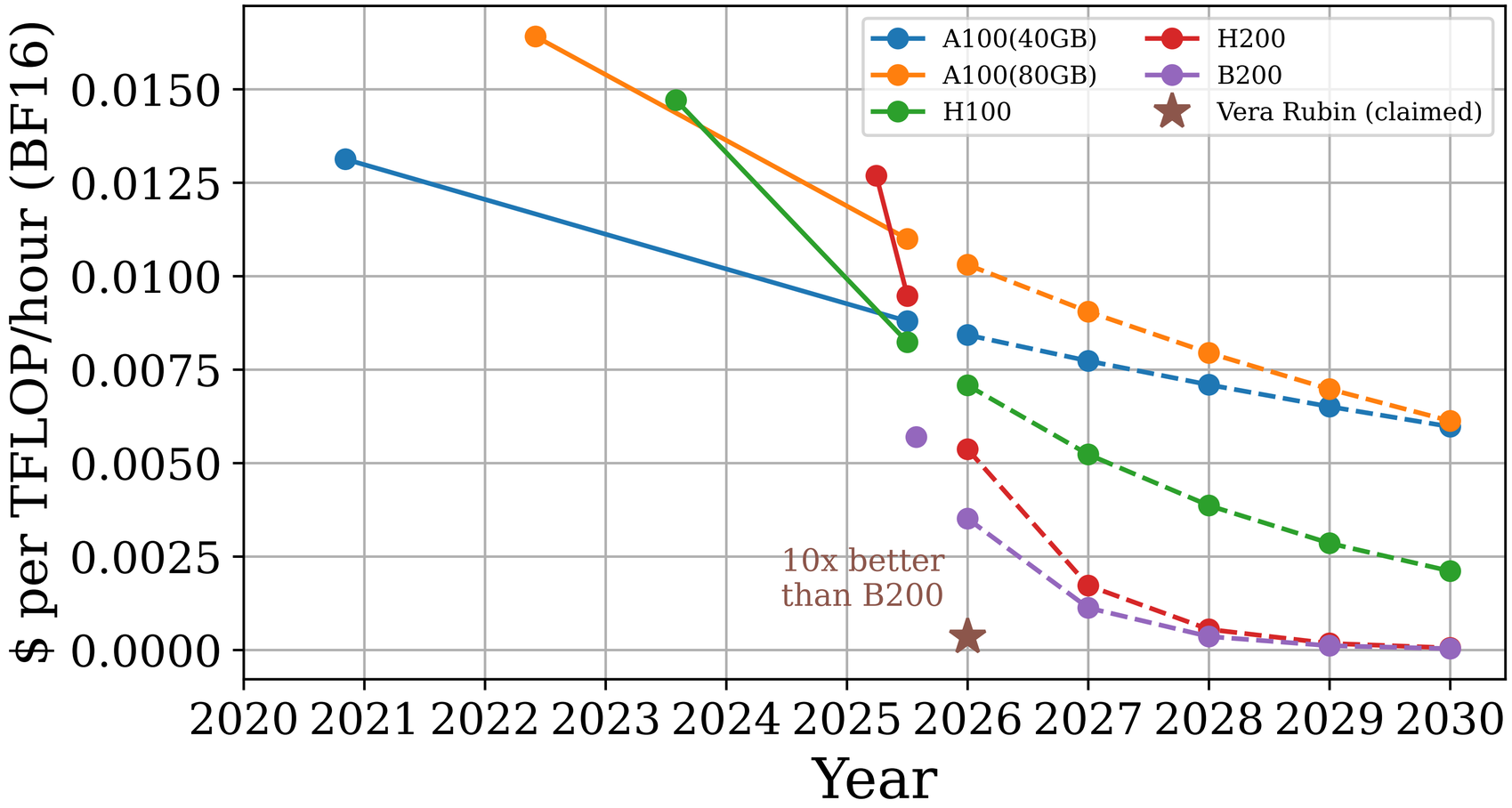
Probability Distribution (text)



What to do next?

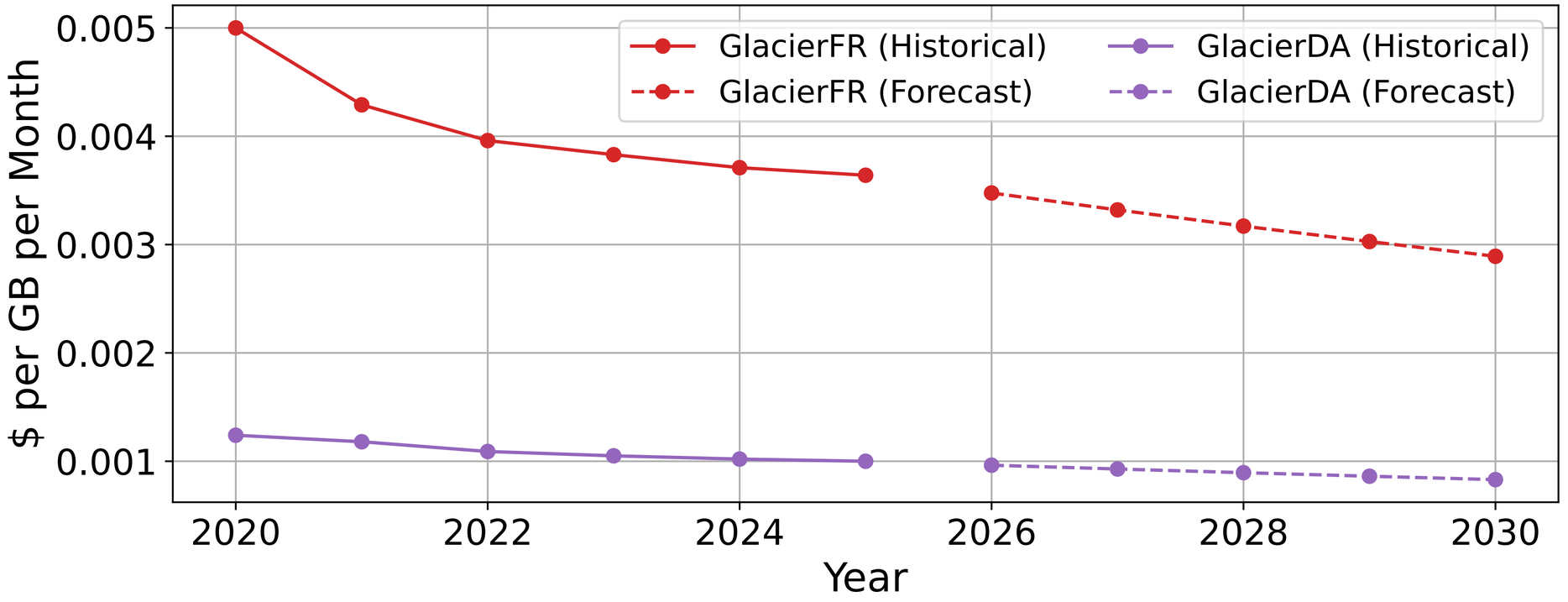
Option 1: Just Wait

GPU Costs



Option 1: Just Wait

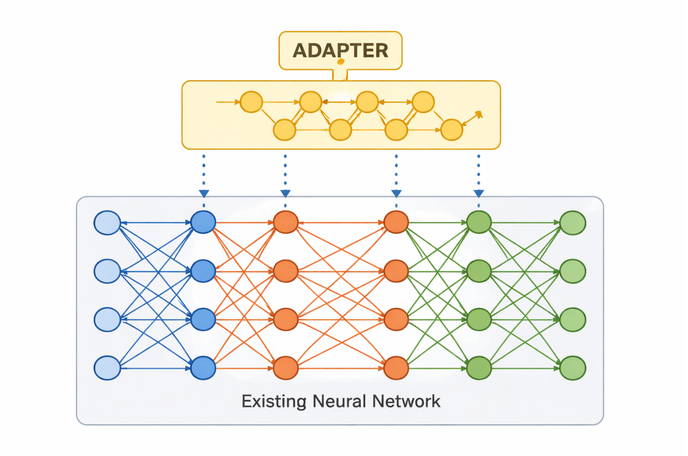
Storage Costs

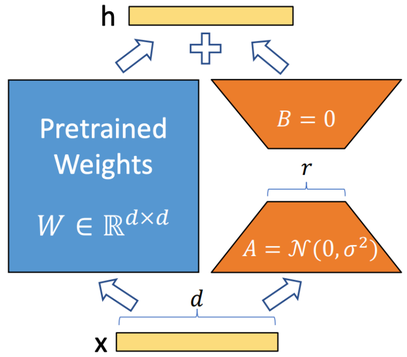


Option 2: Improve Model Accuracy

Better data / model alignment means fewer bits for entropy coding:

1. Could use a **custom model**: Compute and time intensive (bad idea!)
2. Perform parameter-efficient **fine tuning**
3. Train **adapter heads** for pre-trained LLMs





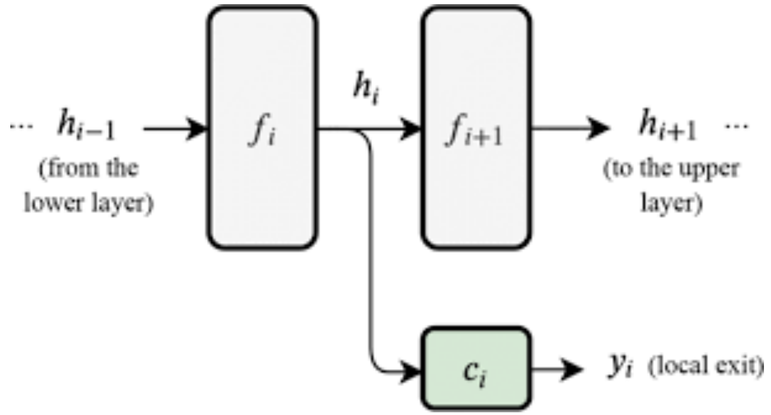
LoRA: Low-Rank Adaptation of Large Language Models

Option 3: Improve Inference Speed

- Intuition: For “easy-to-guess” parts of our data, not all layers are needed for accurate predictions
- **Early-exit networks** allow for exiting layers on a token basis
- Reduces compute (TFLOPS) but not necessarily wall-clock time

Divergence issue isn’t new to the DB community:

Make the Most out of Your SIMD Investments:
Counter Control Flow Divergence in Compiled Query Pipelines
Lang et al. DaMoN’18



Early-Exit Network

Summary

- LLM-based compression **isn’t economically practical yet**
- Cost trends ~~are~~ were promising
- Research helps too (e.g., parameter-efficient fine tuning, early-exit networks)
- Not just for text, also for other modalities (audio, images, videos)

andreas.kipf@utn.de