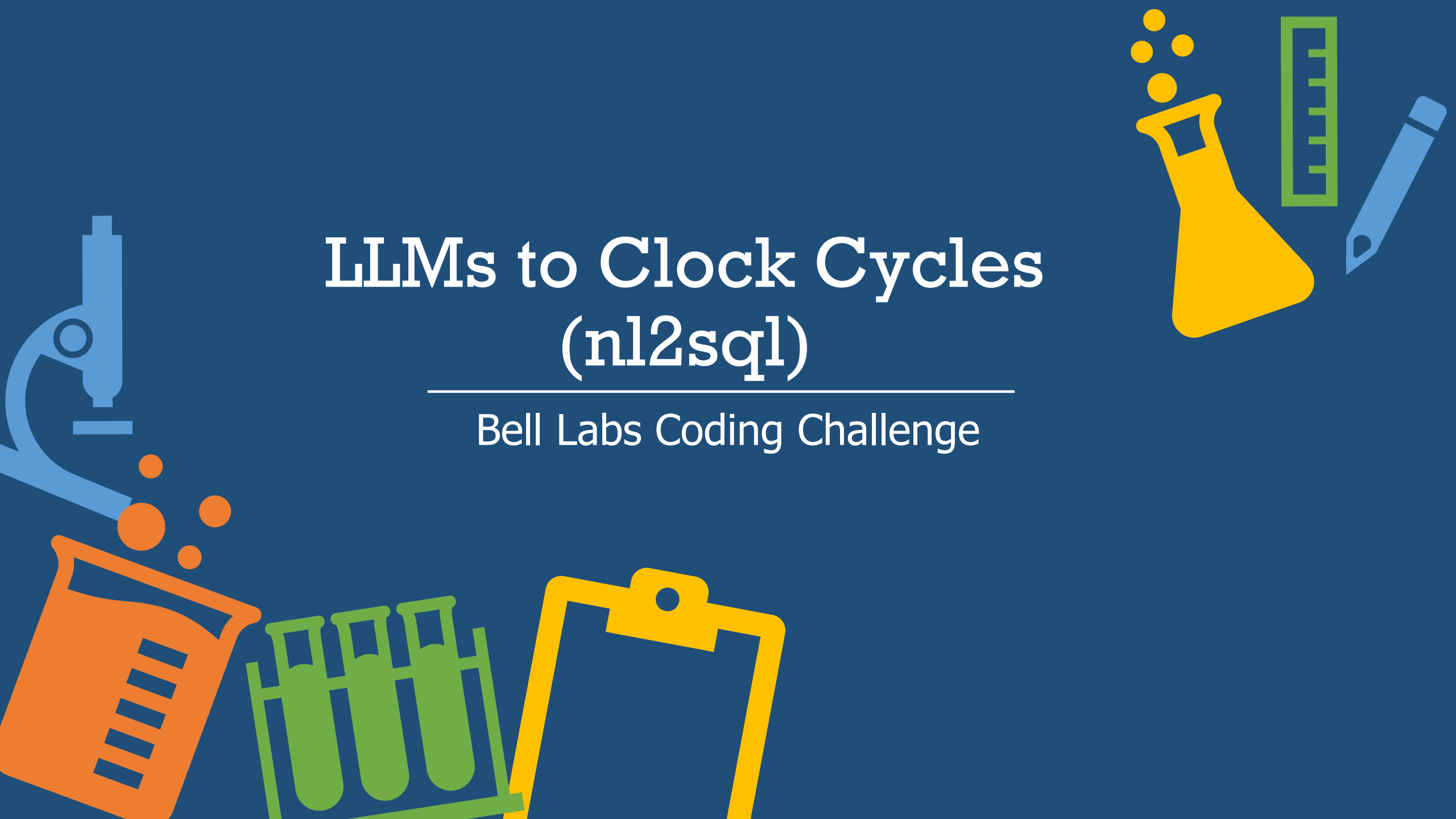


LLMs to Clock Cycles (nl2sql)

Bell Labs Coding Challenge



Approach to Fine-Tune LLMs

- Preparation and Setup
- Data Preprocessing and Exploration
- Model Fine-Tuning
- Evaluation and Testing
- GUI-Development



Abstractions for Optimization

The three layers of possible optimization

High-Level Optimization

Hyper-Parameters

Low-Level Optimization



High-Level Optimization...

- **Model and Dataset:** Choosing apt base model and dataset to avoid too many pre-processing steps and a difficult setup.
- **Metrics and Testing:** Implement early stopping based on validation loss, and use model checkpointing to save the best model. Choose custom test cases that reflect specific business scenarios.
- **Adaptor-Rank Estimation:** Use a rank estimation algorithm to perform empirical testing over a range of ranks.
- **Fine-Tuning Techniques:** Choose a suitable techniques like LoRA, qLoRA, PEFT, Iterative-Refinement, BitNet b1.58, etc.
- **Parallelizing and Automating the Workflow:** Build a parallelized and automated pipeline for the entire fine-tuning workflow as it's an iterative process.



Hyper-Parameters

Parameters that affect the entire workflow of fine-tuning.



Rank



Quantization
Precision



Learning Rate



Batch Size



Epochs



Max Sequence
Length



Low-Level Optimization...

- **Model Quantization:** Reducing the precision of the model's weights from 32-bit floating point to 16-bit floating point or 8-bit Int.
- **Model Pruning:** Removing redundant or less important weights from the model, reducing its size and inference time.
- **Mixed Precision Inference:** Using both FP32 and FP16 operations during inference, allowing for faster computations while maintaining accuracy.
- **Caching Mechanisms and Hardware Specific Optimizations:** Leveraging capabilities like AVX-512, Tensor Cores, TPUs, sophisticated Caches, GPUs, Threads, ONNX, etc.
- **Batch Inference (Parallelism):** Process multiple inputs simultaneously, leveraging parallelism to improve throughput.



Performance Evaluation



Inference Speed



Throughput



Model Size and
Memory Usage



Quantization Impact



Deployment
Considerations



Hardware Profiling



Business Metrics...

- **Query Success Rate:** Measuring the percentage of correct query results without errors and the correct data. Also include time efficiency.
- **Use Case Simulation:** Divide the test dataset into a real-world business scenarios and then run the model. Generate numerically precise reports to compare the results.
- **Evaluation of Query Accuracy:** Involvement of human-experts to generate the SQL queries for the scenarios and categorize the observed errors for better understanding. (*where the model fails*)
- **Assess Business Impact:** Training Evaluate whether the model helps business users make faster and more informed decisions. Can setup continuous monitoring or reporting tools to keep an eye on the model every time it's used.



Fine-Tuning Techniques



LoRA



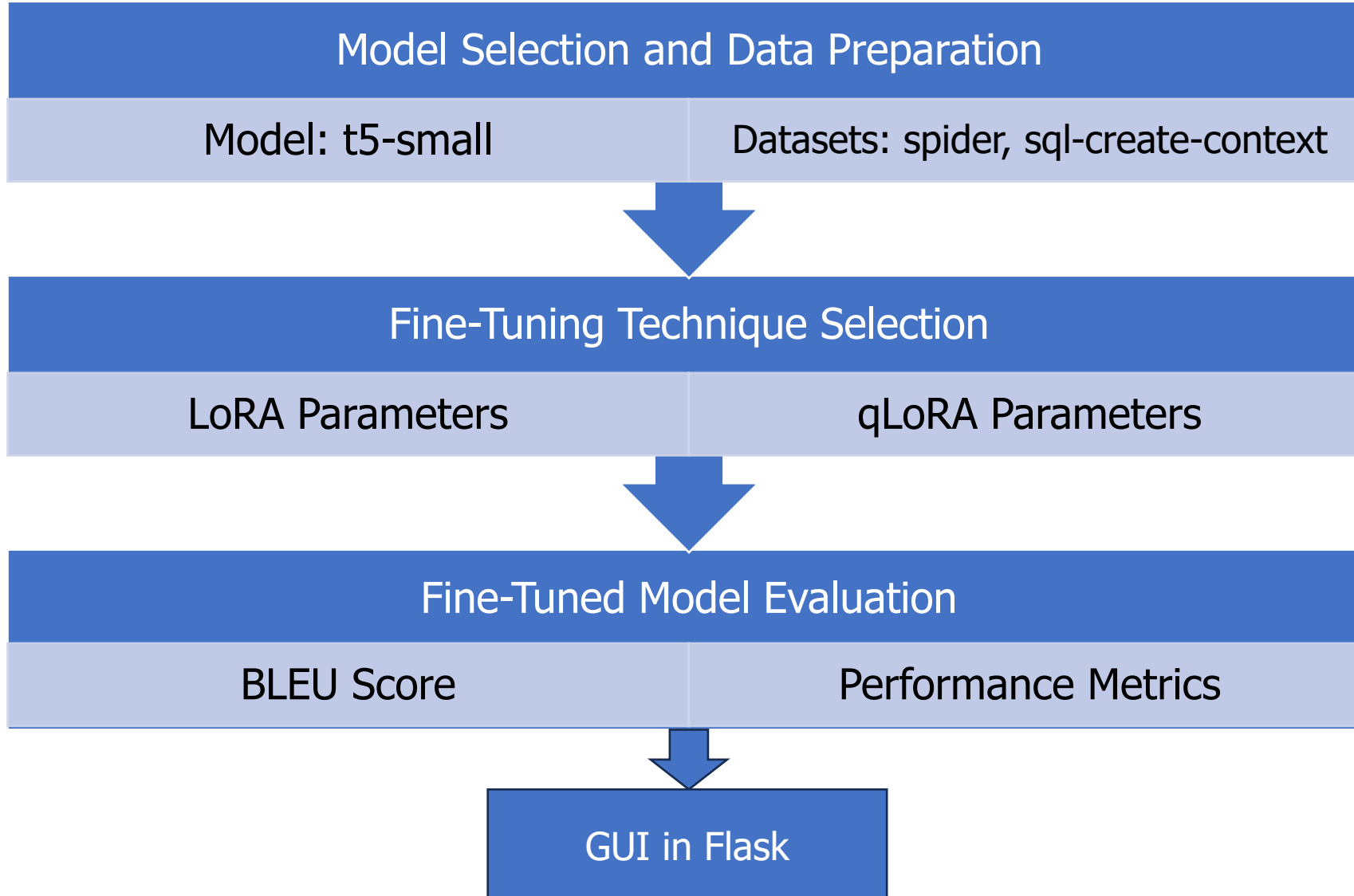
qLoRA



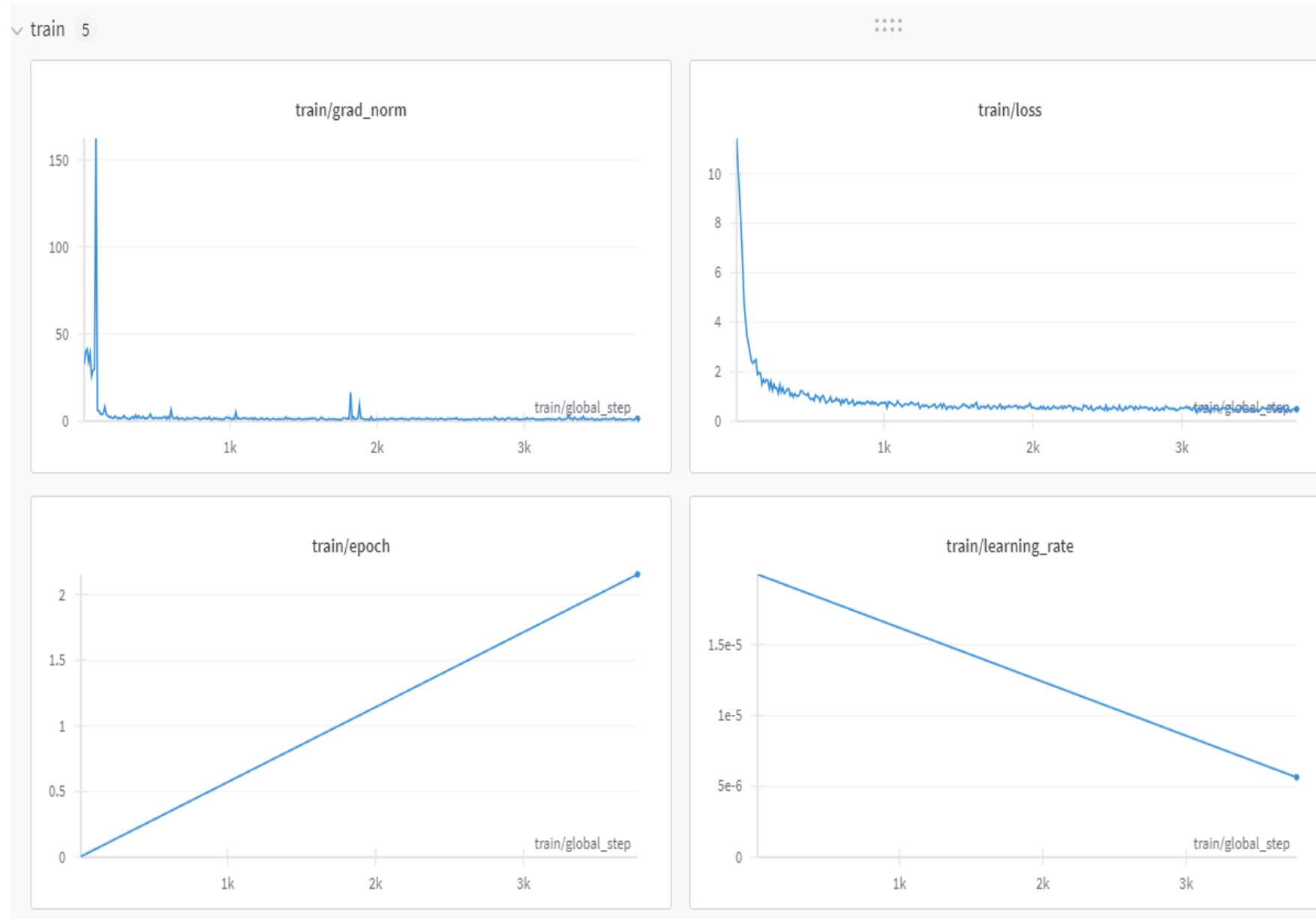
BitNet



My Strategy



Process – Only on CPU



BitNet 1.58b Implementation

The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits

Shuming Ma* Hongyu Wang* Lingxiao Ma Lei Wang Wenhui Wang
Shaohan Huang Li Dong Ruiping Wang Jilong Xue Furu Wei^o
<https://aka.ms/GeneralAI>

Abstract

Recent research, such as BitNet [WMD⁺23], is paving the way for a new era of 1-bit Large Language Models (LLMs). In this work, we introduce a 1-bit LLM variant, namely **BitNet b1.58**, in which every single parameter (or weight) of the LLM is ternary {-1, 0, 1}. It matches the full-precision (i.e., FP16 or BF16) Transformer LLM with the same model size and training tokens in terms of both perplexity and end-task performance, while being significantly more cost-effective in terms of latency, memory, throughput, and energy consumption. More profoundly, the 1.58-bit LLM defines a new scaling law and recipe for training new generations of LLMs that are both high-performance and cost-effective. Furthermore, it enables a new computation paradigm and opens the door for designing specific hardware optimized for 1-bit LLMs.

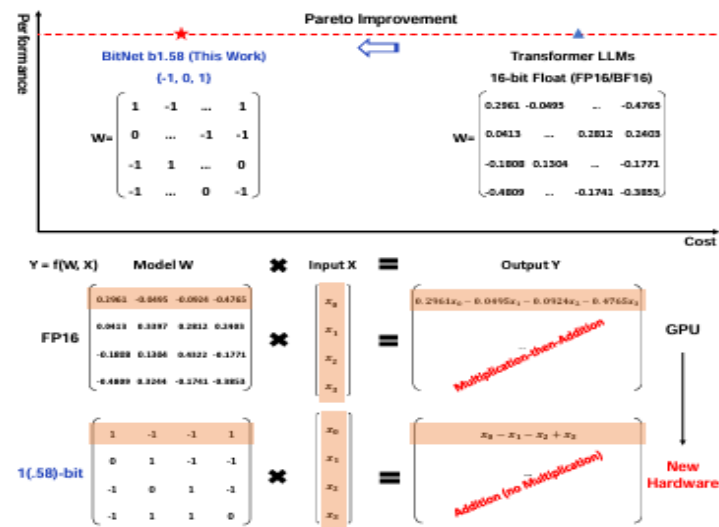



Figure 1: 1-bit LLMs (e.g., BitNet b1.58) provide a Pareto solution to reduce inference cost (latency, throughput, and energy) of LLMs while maintaining model performance. The new computation paradigm of BitNet b1.58 calls for actions to design new hardware optimized for 1-bit LLMs.



Thanks!



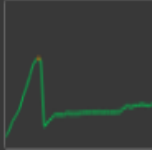

Process

 [125/125 00:44]

```
-----  
OutOfMemoryError                                Traceback (most recent call last)  
<ipython-input-27-143bd5d76c6c> in <cell line: 18>()  
    16  
    17 # Get predictions from the original model  
--> 18 original_preds = get_predictions(evaluator, val_dataset)  
    19  
    20 # Get predictions from the LoRA model  
  
----- 8 frames -----  
/usr/local/lib/python3.10/dist-packages/transformers/trainer_pt_utils.py in  
torch_pad_and_concatenate(tensor1, tensor2, padding_index)  
    92  
    93     if len(tensor1.shape) == 1 or tensor1.shape[1] == tensor2.shape[1]:  
--> 94         return torch.cat((tensor1, tensor2), dim=0)  
    95  
    96     # Let's figure out the new shape  
  
OutOfMemoryError: CUDA out of memory. Tried to allocate 5.64 GiB. GPU 0 has a total capacity of 14.75 GiB  
of which 5.47 GiB is free. Process 78270 has 9.28 GiB memory in use. Of the allocated memory 7.58 GiB is  
allocated by PyTorch, and 1.57 GiB is reserved by PyTorch but unallocated. If reserved but unallocated  
memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True to avoid fragmentation.  See  
documentation for Memory Management (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

Manage sessions

Python 3 Google Compute Engine backend (GPU)
Showing resources from 12:31 AM to 1:58 AM

System RAM	GPU RAM
3.8 / 12.7 GB	9.3 / 15.0 GB
	
Disk 37.9 / 78.2 GB	
