CSE 234: Data Systems for Machine Learning, Winter 2025

Guest Lecture 2: LLM360

Lecturer: Hector Liu

Scribe: Tino Trangia, Max Irby, Arnav Khinvasara, Shi Tianai, Juhak Lee, Shanglin Zeng, Jiaxian Xiang, Marcus Chang, Prashasth Katukojwala, Jinhui Xu, Youngju Jeon

1 Introduction

1.1 About Hector Liu

Hector Liu is a researcher in NLP and computational linguistics. He is currently the director of the MBZUAI Institute of Foundational Models - Silicon Valley Lab. Previously, Hector was the head of engineering at Petuum, where he led research initiatives for NLP pipelines and LLM development. He completed his PhD at Carnegie Mellon University.

1.2 Current state of open-source AI

The open-source paradigm refers to decentralized software development that emphasizes peer production and transparent collaboration. The development of foundation models, however, does not fit the criteria for open-source very well. The current LLM landscape includes closed source models such as GPT-4o, Claude 3.5, and Gemini. Several open weight models exist, such as Llama, DeepSeek, and Mistral. Finally, "360-Open" (i.e. truly open source) models include LLM360 projects as well as emerging players (e.g. OlMo).

Several key challenges hinder open science for AI: (a) the cost of reproduction is high, (b) results are sensitive to permutations and variations in setup, and (c) evaluation is difficult and results can often be misleading. These issues make collaboration between industry and academics problematic – industry groups own most of the training pipeline, while academics have limited access to the pipeline and necessary resources.

1.3 Open kitchen paradigm

Under the metaphorical "open kitchen" approach, there are three levels of openness: transparent, reproducible, and accessible. Transparency requires the LLM training process to be revealed, rather than just delivering the final weights. Reproducibility requires that, given enough information about the training pipeline, academics can achieve the same results. Finally, accessibility necessitates that there are no barriers (e.g. secret paywalls) to the artifacts and that reproduction is economically feasible.

2 LLM360: Data systems and open-source AI

The LLM360 project is significant to data systems because it emphasizes the critical role of data and data management in large language model (LLM) development. It focuses on:

- **Data-Centric AI:**
  - LLM360 highlights that the quality and characteristics of training data are paramount to LLM performance. The TxT360 dataset is a prime example, emphasizing curated, high-quality data to ensure transparent and reproducible model training.
  - This aligns with data systems principles of data quality, provenance, and management.
- **Open-Source and Reproducibility:**
  - A key aspect of LLM360 is the commitment to "360° open," meaning full transparency in data and model development. This involves releasing not just the models, but also the complete training datasets, code, and intermediate checkpoints.
  - This is crucial for data systems because it promotes reproducibility, allowing researchers and practitioners to verify and build upon existing work.
- **Large-Scale Data Management:**
  - Training LLMs requires managing massive datasets, which presents significant challenges in data storage, processing, and distribution. LLM360's efforts in creating and sharing datasets like TxT360 contribute to advancements in large-scale data management techniques.
  - The project also emphasizes the importance of data deduplication and data lineage, which are very important in modern data systems.
- **System Architecture:**
  - The LLM360 project also has implications regarding the system architecture needed to train LLMs. This includes considerations for distributed computing, parallel processing, and efficient data pipelines.
  - The release of training code and hyperparameter information allows for the study and optimization of these large-scale systems.

3 LLM360 projects

Overview

Amber: first model with lot of bugs, Crystal, K2, TxT360

Overview of the models

Crystal can balance language and coding ability

K2

- can beat Llama2 performance with 35% less total FLOPS
- released a lot of artifacts (actual data sequence(training data), intermediate checkpoints, intermediate results, metric curves, evaluation metrics)
- Closing the gap to SOTA

The Landscape of LLMs

LLMs can be grouped into three categories: Close source LLMs, Open Weight LLMs, and 360-Open LLMs. Hoping to reach top-tier ranking models.

4 Demystifying the LLM training process

Large Language Model training requires a deep understanding of engineering trade-offs and the underlying scientific principles. The training pipeline for an LLM is structured to provide guide each stage of development. At its core, the pipeline is about planning. Since the training phase offers limited real-time control, every decision made during planning can significantly affect the final outcome.

4.1 Goals and budgets

The first step is to determine goals and budgets. There is a set of aspects to be taken into consideration that might be insightful for figuring out the goals of the model. To be specific, this problem could be approached from three aspects, the major use case of the model, the major knowledge domains that need to be covered, and the abilities the model should have.

The goal will help determine the budget as the minimum requirements for achieving the goals is identified. The optimization goals can be categorized into performance, performance per dollar, and returns on investment (ROI).

Beyond performance, determining the budget involves weighing training costs against ROI. This not only involves calculating performance per dollar during training but also long-term operational costs like inference. After all, a model might require periodic retraining as it evolves or becomes obsolete.

It must be decided how much resource to invest in scaling the model's capacity, as it is essential to strike a balance between performance gains and practical limitations. Scaling laws such as Chinchilla offer valuable insights into how changes in training time, dataset size, and model parameters affect the loss and overall performance. They can also forecast emerging abilities that models develop as they grow larger.

Understanding scaling laws not only helps in planning the training process but also in making strategic decisions about model architecture. By predicting loss trajectories and performance improvements, teams can optimize their resources and better estimate the trade-offs between training cost and model capability.

4.2 Data preparation

Background

- Collect high quality and large corpus is essential in producing in the final model.
- To deal with Internet scale data, the typical way is to filter documents based on heuristic rules, which is different from different domains.

Methods

- **Deduplication**
  - Most study confirm that deduplication can help improve model quality.
  - Hypothesize that duplicate data would cause the model to replace generalization ability with memorization.
  - If a portion of the dataset repeats 1M times, it causes a worse performance.
  - Correctly repeat training data improves performance.
- **Vocabulary**
  - A hyperparameters selection method without doing backpropagation.
  - If the model is larger, then we need more vocabulary.

Resource

- **TxT360 Blog**: A detailed blog describing every steps of the implementation.

4.3 Model architecture choices

Available Architectures:

- Transformer based: GPT series, Llama variants
- State Space Model: Mamba, Striped Hyena
- RNN like models: RWKV

Reasons for choosing Transformer based model:

- Transformer based model works consistently well
- SSMs are hard to be trained for coding tasks

Hardware Aware Decisions:

- Engineering process is highly dependent on underlying hardware.
- On Nvidia GPUs, matrix dimensions should be multiples of 8 or 16 for optimal performance.
- Recommended parameter choices:
  - Vocab size: divisible by 64
  - Other dimensions: divisible by a power of 2

  - Certain ratios should be integers
  - Microbatch size: as small as possible

Model Architecture Efficiency:

- Memory constraints are major bottlenecks in LLM models.
- Techniques to mitigate memory issues:
  - KV-Cache Memory
  - Group Query Attention instead of MQA
  - FlashAttention
- Architecture design should consider hardware efficiency.
- Mamba has better FLOPs but may be slower in terms of Wall Time.

4.4 Hyperparameter study

Hyperparameter is important when planning because it makes a big difference in the long term. Most of the time, the initial hyperparameters define the final model because due to the scaling law, if the loss is adjusted only slightly, the final benchmark results can still deviate significantly.

The goal is to ensure that the model behaves consistently throughout the entire training process. In order to do that we need to make some assumption. For example, if there are 10 training tokens, each token should contribute equally to the final model. The entire training process relies on stochastic gradient descent (SGD), making proper gradient updates crucial. To achieve this, noise must be controlled, which requires balancing batch size and learning rate. A larger batch size reduces noise but results in fewer updates, while a smaller batch size increases noise but allows more updates.

Scaling law studies help predict which hyperparameters work well across different model sizes. One approach is measurement parameter update, which aligns the optimal learning rate across different scales. Without this method, the best learning rate can vary significantly depending on model width.

Batch size is one of the most important hyperparameters. If it is too small, each update becomes highly unstable, increasing the risk of training in the wrong direction. If it is too large, the model receives fewer updates and may struggle to reach an optimal state. Research on Critical Batch Size provides a method for estimating an optimal batch size by analyzing gradient noise.

Additionally, batch size should be adjusted based on the total number of training tokens. If a model trains on more tokens, increasing the batch size helps reduce noise while maintaining enough updates. However, too many updates can cause the model to forget earlier training samples. Batch size is one of the most significant factors affecting loss, so it should be determined before tuning other hyperparameters.

4.5 Preparing runtime

4.5.1 Training Parallelism

When designing an LLM training framework, we must ensure that it provides code stability and implementation correctness, as well as parallelization support.

Parallel training can be divided into two paradigms. The first is Data Paralellism (DP), used when the input dataset is very large, where the input data is partitioned across GPUs and each GPU contains a full replica of the model. The second is Model Parallelism (MP), used when the model is very large (e.g. LLMs), where the model itself is partitioned across GPUs. A key challenge with MP is how to partition the computational graph.

Additionally, some more advanced parallelism strategies are employed in LLM360: Tensor Parallelism ("TP", where consecutive layer weights are partitioned row-wise and column-wise), Sequence Parallelism ("SP", partition along the sequence dimension), and Pipeline Parallelism ("PP", where layers are placed acrossed GPUs and training mini-batch is split into micro-batches). There are tradeoffs between all of these parallelism strategies – TP and SP require addition cross-device communication compared to standard approaches (but can help ensure that the LLM can fit into a group of GPU nodes), and PP requires less communication than DP/TP but suffers from increased device idle time (aka pipeline bubbles).

The manage these tradeoffs, LLM360 used a heuristic tuning process allowing them to come to a hybrid parallelism strategy combining TP, SP, PP, and DP. This involved fixing the TP dimension to avoid communication bottlenecks, tuning the micro-batch size for PP mini-batches, tuning PP dimension, and finally tuning the DP dimension (from the K2 tech report linked here, also contains graphics on the final hybrid training parallelism structure of K2 Diamond). LLM360 is also currently working on training infrastructures for long context and mixture of expert training.

4.5.2 Fault Tolerance

LLM pre-training can encounter many potential issues and faults: hardware failure (e.g. CUDA NCCL error), unknown hardware or network slowing down, loss spikes during training, and NaN loss and training divergence.

Types of hardware failures include: NCCL test timeout, low fractions of active tensor core (slows down training), a GPU going down, GPU node-level HW failure, OS I/O errors, lustre error (causing GPU nodes to reboot), GPU node mount failure, and running out of storage on the cluster. Wherever possible, we should try to design fault-tolerant systems.

Precision is another important model setting – it involves a tradeoff between numerical accuracy and cost. Using a standard like FP16 can result in underflow, where small gradients are rounded down to zero due to FP16's small exponent range. Underflow can potentially be fixed by loss scaling (multiplying loss by a large value, then computing scaled gradients thru backprop, then scaling gradients back down before applying an update). Also, BF16 can be used which has more precision for small values (same exponent range as FP32, sacrificing precision for large values).

LLM360 encountered some problems in their runtime process: in Lit-llama and Megatron-LLM repos they found that models were sometimes stored at incorrect precision, and they found some precision changes midway through due to a Cerebras hardware upgrade. Debugging these issues was key in getting LLM360 to work.

4.6 Training job

4.6.1 Evaluation and Logging

Proper evaluation and logging are crucial to ensuring the effectiveness of LLM training. A key challenge during LLM360 training was the lack of sufficient evaluation machines for the 65B model. Frequent evaluations prevent wasted computation on underperforming models. However, there is a tradeoff between allocating

resources for evaluation and training.

- **Held-out perplexity** is a direct measure of model performance.
  - Trends in training loss generally align well if data is **deduplicated**.
  - A separate **held-out dataset** remains necessary to avoid data leakage.
- **Task-based benchmarks** measure target metrics and ensure no unintentional **data contamination**.
- **Test-out generation** involves continuous sampling of model outputs to assess quality.

4.6.2 Benchmarks

LLM benchmarking consists of two primary categories: generation-based and multiple-choice benchmarks.

- **Generation-based benchmarks**
  - Common examples include **BigBench, GSM8K, and MBPP**.
  - These tasks often involve **Chain-of-Thought (CoT)** reasoning or complex generation.
  - While more reliable for evaluating generative capabilities, they tend to have **high variance**, particularly for small models.
- **Multiple-choice benchmarks**
  - Examples include **MMLU** and **ARC**.
  - These benchmarks are **easier to implement and evaluate**.
  - Two primary methods of implementation:
    - **Actual generation**, which is difficult to control.
    - **Perplexity-based choice selection**, which requires **normalization over choice length**.
  - Some metrics may be **misleading** and fail to detect **degenerated models**.

4.6.3 Dealing with Problems

Loss spikes are a common issue in large-scale pretraining. Empirically, they were observed in the 65B model training but not in the 7B model training.

- **Reducing the influence of individual data points** can help stabilize training.
  - Techniques such as **embedding layer gradient shrink** can reduce loss spikes.
- **Skipping problematic instances** prevents training disruptions.
- **Model averaging**, including **early weight averaging**, has been reported to alleviate spikes.

4.6.4 LLM360 Traces

To ensure transparency and facilitate reproducibility, LLM360 provides detailed training traces. These traces include intermediate checkpoints, model outputs, and evaluation results. By comparing training traces, researchers can assess whether a model is behaving normally and identify potential anomalies in the training process.

4.7 Training wrapup

Completing the training process does not mark the end of the workflow; several post-training steps are necessary to refine the model.

One crucial step is learning rate annealing, applied at the final stage of training. During this process, the learning rate used to update the model's parameters is gradually reduced over time, eventually decaying linearly to zero. This approach enhances convergence by helping the model reach a more optimal solution while reducing the risk of getting trapped in local minima.

Another optimization technique involves applying quantization to represent the final model with lower precision, reducing its computational and memory requirements. However, prior research indicates that certain parts of the model may not be well-suited for quantization.

To ensure effective quantization, we employ Quantization-Aware Training (QAT). It simulates the impact of quantization during training by modeling the effects of quantization during training, allowing for higher accuracy compared to other quantization methods. Additionally, deep learning frameworks such as PyTorch and TensorFlow provide built-in support for QAT, facilitating its seamless implementation.

Finally, we can fine-tuning the model to enhance its precision and fairness. For instance, instruction-following or agent models can be fine-tuned to improve specific capabilities, such as arithmetic reasoning and coding proficiency. Additionally, efforts to integrate vision capabilities have led to the development of models like CrystalVision and K2Vision, which are still under refinement.

However, fine-tuning must be approached with considerations for safety and cultural alignment. Observations from our natively fine-tuned K2 model indicate that larger models are more prone to overfitting on fine-tuning datasets.

5 Conclusion

Overall, currently, there are many challenges affecting open-source AI development, mainly factoring down to how costly LLMs are to reproduce and how evaluation of LLMs, especially relative to other models, is a difficult task. However, the LLM360 project aims to directly address these challenges. The LLM360 intiative focuses on making training data, code, and intermediate checkpoints publicly available to allow for transparent and reproducible LLM research and development. It also provides access to check model behavior, conduct ablation studies, and refer to model statistics by making the full trace of each model available. The initiative firmly emphasizes that the performance of LLMs depend strongly on the management and quality of training data. The project also provides insights into what it takes to build, manage and optimize large-scale systems on different system architectures.

LLM360 is also able to help with LLM training specifically by providing traces of intermediate checkpoints, model outputs, and evaluation results. In this way, developers building an LLM can compare their training trace with the LLM360 references to ensure that their model is behaving normally. Using the intermediate checkpoints, a lifetime analysis of the model can be done to study its performance and make improvements. LLM360 will also be able to support training infrastructures for long context and mixture of expert training. Therefore, while the LLM training can be multifaceted and complex, LLM360 provides transparency of the training process for its open-source models and provides tools for the development of new models, paving the way to a greater future in open source LLM and AI development.

6 Contributions

- Tino Trangia: Section 1 (introduction)
- Arnav Khinvasara: Section 2 (LLM360 overview)
- Youngju Jeon: Section 3 (LLM360 projects)
- Shi Tianai: Section 4.1 (goals and budgets)
- Marcus Chang: Section 4.2 (data preparation)
- Shanglin Zeng: Section 4.3 (model architecture choices)
- Juhak Lee: Section 4.4 (hyperparameter study)
- Max Irby: Section 4.5 (runtime)
- Jiaxian Xiang: Section 4.6 (training job)
- Jinhui Xu: Section 4.7 (training wrapup)
- Prashasth Katukojwala: Section 5 (conclusion)