Introduction

Aaron Zhao, Imperial College London

General Introduction

Introduction - Myself

My name is Aaron, and my research looks at the intersections between algorithm, hardware and security in Deep Learning (DL) Systems.

My email is a.zhao@imperial.ac.uk, and my office is 903.

Introduction - Myself

- **Automatic Co-designing AI Systems with MASE**

  MASE aims to provide a unified representation for software-defined ML heterogeneous system exploration.

- **Beyond Structure Data**

  Investigate on unstructured and multimodal data, such as graphs, hypergraphs and combinatorial complex.

- **Efficient AI**

  Different algorithmic aspects of efficient AI, including efficient training, efficient inference, efficient model search and efficient model deployment with state-of-the-art GenAI models (eg. language and diffusion models).

- **System-level AI Safety**

  This includes robustness, security, and red-teaming these models to understand new vulnerabilities.

Introduction - People

The course is also supported by the following people:

- Cheng Zhang (GTA)
- Pedro Gimenese (GTA)
- Mingzhu Shen (GTA)
- Jeffrey Wong (GTA)

Introduction - Where to find stuff?

Everything is online, you simply a few urls

- The course wiki (https://aaron-zhao123.github.io/teaching/adls)
- Bugs should be reported on Github Issues (if it is lab related)
- There are also in-person lab sessions for questions
- Email me (a.zhao@imperial.ac.uk)

Introduction - Why this course?

- Learning pure DL theories and model architectures is superficial

You need to understand:

- What is actually running in hardware when you do "torch.matmul"?
- How do these ML/DL frameworks (eg. Tensorflow, Pytorch) work?
- How to deploy your model on different backends?
  - eg. CPUs, GPUs, *etc.*
- How to make them run better or faster? What are the software and hardware tricks for these optimisations?

Hopefully you will have an idea of the answers of these questions.

Introduction - Assessments

The course and its labs are designed around an in-house compilation framework that we have built, named MASE.

- Practicals (total 30%)
  - Answer all the questions in practicals in a single markdown file.
  - Higher marks would be awarded with more extensions (optional tasks) answered/finished.
  - A lab oral with a GTA to show your progress on the practicals and check whether you have actually understood the material.
- A team project (3-4 people) to self-propose a topic, implement and evaluate the an automated optimization flow (70%).
  - You can build on top of MASE if you think that is easier.
  - Or you can start your own project from scratch.

This course contains both software and hardware aspects, this is a **CHALLENGING** course, and you will be expected to write a lot of code!

More on practical assessments

The lab orals and assessments for

- MSc ADIC (more hardware focused) and MSc AML (more software focused) students are different.
- MEngs, feel free to pick your path.
- The purpose of the labs is for you to learn what is involved in a DL framework, we have four lab
- The sessions will be supported by GTAs:
  - Lab 0: Introduction to Mase
  - Lab 1: Model Compression (Quantization and Pruning)
  - Lab 2: Neural Architecture Search
  - Lab 3: Mixed Precision Search
  - Lab 4 (Hardware): Emitting Hardware
  - Lab 4 (Software): Performance Engineering

More on practical assessments (ii)

- Questions about MASE functionalities, apart from asking the GTAs in Lab sessions, can also be queried through Github Issues.

More on practical assessments

- The labs are intended to assist you in navigating and comprehending a substantial codebase.
- To ensure your understanding, we will facilitate a one-on-one lab oral (each 15-min) with you that aims to make sure:
  - You finished it individually.
  - Verify that you have grasped the material thoroughly.
  - Ensure that you are engaging with the content rather than merely replicating commands or code.
- Show us your code with your comments, and reason about it.

More on team projects

You will need to form a team 3-4 people team. If you cannot find a team yourself, this is fine, we will find you one.

- Planning
  - Basic functionality requirements:
    - An automated optimization flow (software)
    - FPGA hardware components library (hardware)
  - You propose your own project, and we will provide you with feedback.
    - Does your project meet the requirements?
    - Is there a clear path to completion?
    - Is there enough engineering work to be done? And is it suitable for a 3-4 people team?
  - A roadmap meeting with a GTA or me (not assessed).
    - We expect to see an outline of functionalities that you would like to implement

More on team projects (ii)

- An arrangement of who is doing what.
- This is to provide you feedback with your work.

More on team projects

- Assessments
  - Code submission as a pull request or a repository on Github (assessed).
    - We will actually run your code and test it.
    - Your code (correctness, functionality and quality)
    - Your augmented testing script (coverage)
    - A short readme file and a few commands to verify it works (clarity)
  - A show me your code session (assessed, 20 mins).
    - Our GTAs may ask questions on your code in the PR.
    - In this way, I expect you to learn how to work on an open-source project.
  - A 4-page report per group in ACM format (assessed).

Introduction - My assumptions

I assume you know

- Basic terminal commands (*eg.* cp, mkdir ...).
- Basic version control knowledge (*eg.* git, github ...)
- Python and Object Orientated Programming
- System Verilog (for hardware-stream students)

If you never heard of these words, and are not willing to pick them up yourself through Google/Youtube/ChatGPT, think twice before you take this course.

Jianyi has prepared an instructional on basic git usage https://jianyicheng-research.notion.site/Git-Tutorial-516864ab8fa04242ad520652744b931f

Dates and structures

Important Dates

- 7th Feb, Practical submission
- 9th Feb, Team project sign-up ends
- 10th Feb, Midterm lab oral
- 18th Mar, Show me your code session (for Group Project)
- 28th Mar, Report submission and pull request finalization

Dates and structures

- Lectures: you have in total 12 lectures.
  - The lectures do not necessarily cover all aspects of the coding skills.
  - The lectures aim to deliver theoretical knowledge, background and an overview of recent progresses in the field of Deep Learning Systems.

I believe you learn coding by doing it (that's why labs exist), and you do good research/engineering by understanding the field and code it up.

That's why there are lectures and team project.

The team projects are expected to be related to the lectures' contents.

Dates and structures

Labs: you have in total 10-hour labs.

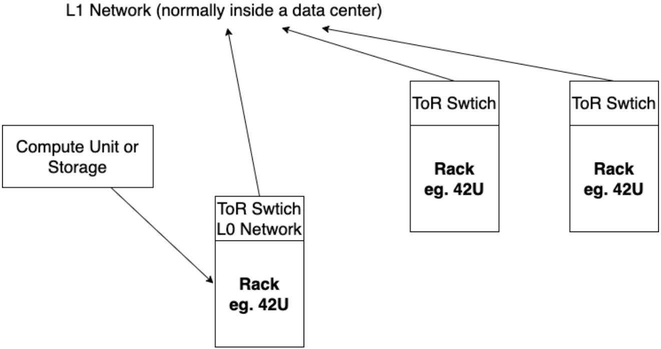
- **Training phase**: in the first two lab sessions, you will follow instructions to finish a number of tasks.
- **Project phase**: the other lab sessions are for you to consult with the GTAs for your team projects.

Additional independent study and lab hours are needed.

An Introduction to Modern Deep Learning Systems

The limitations on large-scale systems

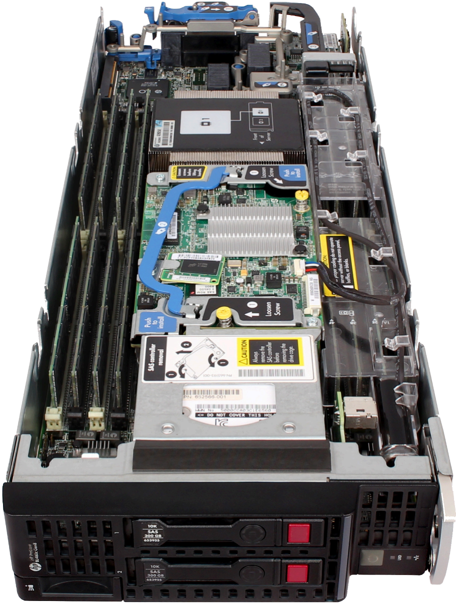
- The absence of shared memory
- Synchronization issues
- Communication latency and network congestions



Blade servers in a rack

- Compute hardware (CPU and GPU)
- RAM
- Disks

Blade servers in a rack (ii)



The limitations on hardware

- The end (or slow-down) of Moore's law
- Wire delays
- Power/Cooling is a key limitation (dark silicon)

The slowdown of Moore's Law The Moore's law states that the number of transistors in an integrated circuit (IC) doubles about every two years.

Fun fact: Gordon Moore is the co-founder of Intel, so this served as the company strategy for a great number of years.

The slowdown of Moore’s Law

- Small feature size is very hard!
- More transistors do not translate directly to performance.
- Parallel hardware is the way to go but instruction level of parallelism (ILP) is limited.

Many vendors are now looking at new technologies such as 3D die stacking, optics interconnects, and other methods to sustain this growth.

Wire delays

As feature size shrinks, wires become slow relative to logic.

Performance would then be bounded by **the amount of data reachable** in a single clock cycle, not by the number of transistors manufactured on a chip or the clock frequency of these transistors.

This builds a dependency upon the processor architecture, cooling technology, and application workloads.

Dark silicon

Dark silicon is the amount of circuitry of an integrated circuit that cannot be powered-on at the nominal operating voltage for a given thermal design power (TDP) constraint.

At 7 nm technology nodes, it is estimated that the amount of dark silicon reaches up to 50–80%.

The general trend in hardware computation Heterogeneous Computing: Divide the workload and build specialized hardware for that workload.

In this course, we will look at specialized hardware for ML.

Trends in ML model designs

Concurrently, it seems like it is going from domain-specific ML models to general (or foundation) models.

- The rise of foundation models (eg. GPTs for language, Diffusion for vision).
- New models are getting larger and larger.
- The idea of handling multi-modality inputs (eg. Google Pathway) is rising.
- The idea of inference scaling (eg. Chain of Thoughts) is also rising.

We are also designing systems that adapts to different use scenarios:

- Federated learning, because of privacy concerns (or not).
- On-device learning.
- the list really goes on

Two key deployment scenarios

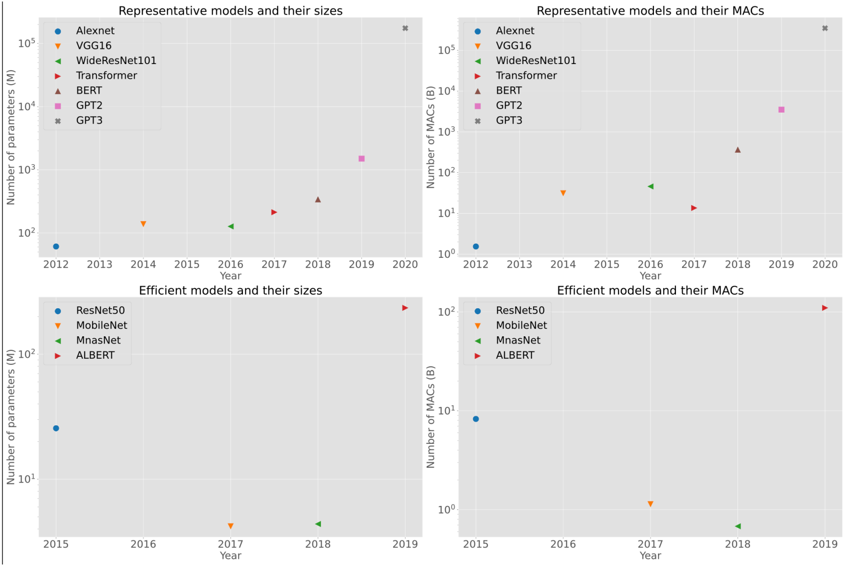
We consider mainly two systems: the cloud system and the edge system:

- Difference in scale
- Difference in budgeting (power, speed, reliability, input rate ...)
- Difference in optimization techniques

Why ML is a very challenging workload?

- Models are getting larger and larger
- We are developing efficient models, but it requires time

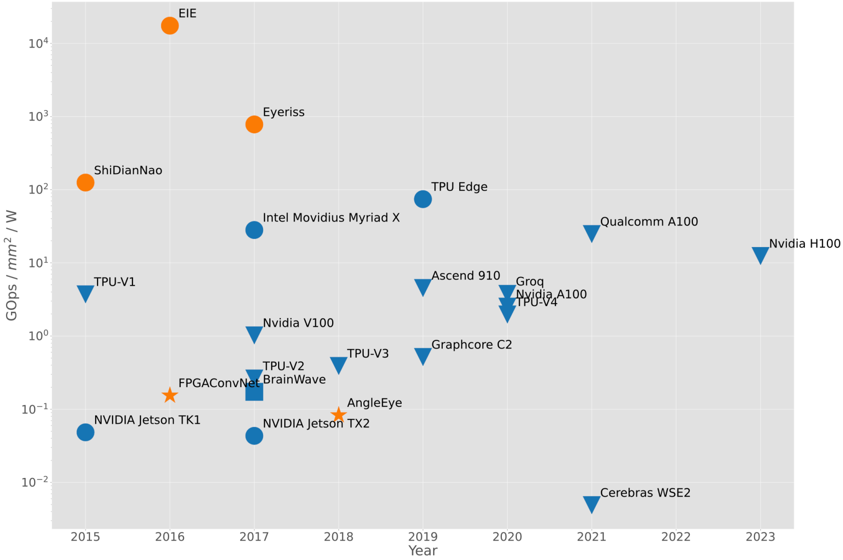
Why ML is a very challenging workload? (ii)



Why ML is a very challenging workload?

- Hardware scaling is a lot slower!
- Basically just Moore's law…

Why ML is a very challenging workload? (ii)

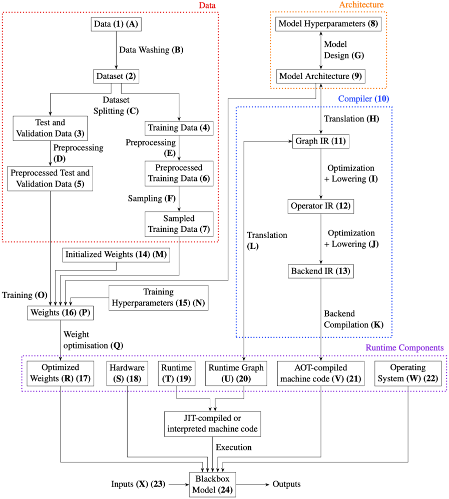


How should we deal with it?

- Software level optimizations (eg. pruning, quantization), typically offers you 2 to $50 \times$ performance gains.
- Compiler level optimizations (eg. loop tiling, loop unrolling)
- How can we better map the workload to existing hardware, typically offers another 2 to $5 \times$ performance gain.
- Hardware level optimizations (eg. PE design, memory system design), typically 2 to $5 \times$.

We will explore these optimizations in this course.

An overview of an ML System



Data

Data (1) (A)

Data Washing (B)

Dataset (2)

Dataset Splitting (C)

Test and Validation Data (3)

Training Data (4)

Preprocessing (D)

Preprocessing (E)

Preprocessed Test and Validation Data (5)

Preprocessed Training Data (6)

Sampling (F)

Sampled Training Data (7)

Architecture

Model Hyperparameters (8)

Model Design (G)

Model Architecture (9)

Compiler (10)

Translation (H)

Graph IR (11)

Optimization + Lowering (I)

Operator IR (12)

Translation (L)

Optimization + Lowering (J)

Backend IR (13)

Backend Compilation (K)

Initialized Weights (14) (M)

Training (O)

Training Hyperparameters (15) (N)

Weights (16) (P)

Weight optimisation (Q)

Runtime Components

Optimized Weights (R) (17)

Hardware (S) (18)

Runtime (T) (19)

Runtime Graph (U) (20)

AOT-compiled machine code (V) (21)

Operating System (W) (22)

JIT-compiled or interpreted machine code

Execution

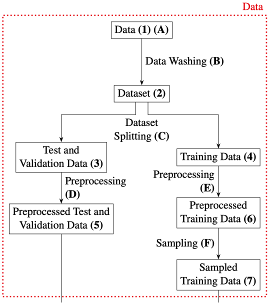
Inputs (X) (23)

Blackbox Model (24)

Outputs

https://mlbackdoors.soc.srcf.net/

ML System: Data



A: The original data

B: The process of removing useless datapoints, outliers, poorly labeled data, and so on.

C: The process of splitting the entire dataset into training data and test/ validation data.

ML System: Data

D: The preprocessing of the test/validation dataset, e.g. random rotation and color jittering.

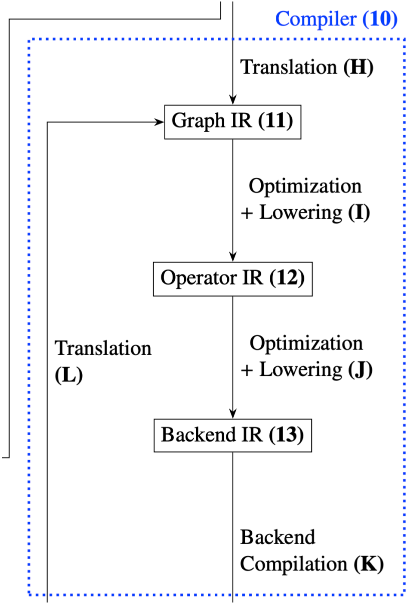
E: The preprocessing of the training dataset, e.g. random rotation and color jittering.

F: The sampling of the training dataset, e.g. to separate it into batches for stochastic gradient descent.

From the system perspective, data may come from sensors (edge systems) or from networked devices (server systems).

Is there enough preprocessing power and bandwidth for receiving the data so we do not starve the computing units?

ML System: Compiler



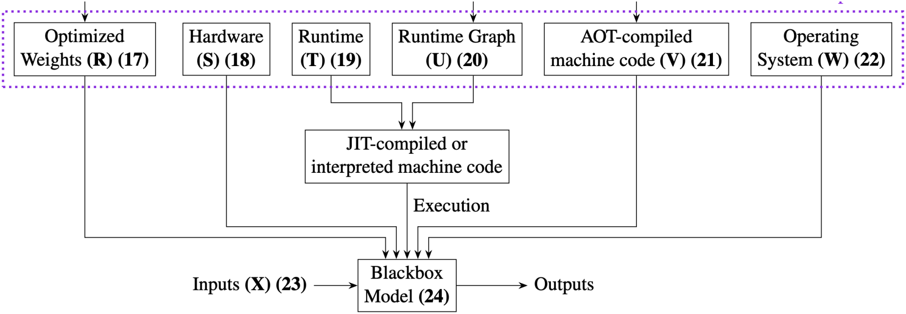
ML System: Compiler (ii)

Graph IR is a high-level IR of an ML model. Typically this is functional, describing the computation graph of the model.

Operator IR is a lower-level IR that is closer to machine code, often including explicit parallelism and memory allocation.

Backend IR is the language used by the backend(s) that the ML compiler uses. For example, the CUDA language is a Backend IR for CUDA, LLVM IR is a Backend IR for LLVM, and so on. The ML compiler might use multiple backends, for example if both CPU and GPU can be utilized.

ML System: Hardware


Optimized Weights (R) (17)
Hardware (S) (18)
Runtime (T) (19)
Runtime Graph (U) (20)
AOT-compiled machine code (V) (21)
Operating System (W) (22)
JIT-compiled or interpreted machine code
Execution
Inputs (X) (23)
Blackbox Model (24)
Outputs

If Just In Time (JIT) Compilation is used, we use a Runtime (T) that interprets or JIT-compile the Graph IR (V) to run the model.

- Interpreter: Reads your source code or some IR (bytecode) of it, and executes it directly (normally 1-to-1 mapping).

ML System: Hardware (ii)

- JIT compiler: Reads your source code, or more typically some IR (bytecode) of it, compiles that on the fly and executes native code (segments to segments).

If Ahead-of-Time (AoT) Compilation is used, machine code (V) is directly executed on the target hardware.

Summary

- Dates and assessments schedules.
- Limitations on current distributed systems and hardware.
- Trends in ML models and why they are challenging.
- A high-level overview of current ML systems.