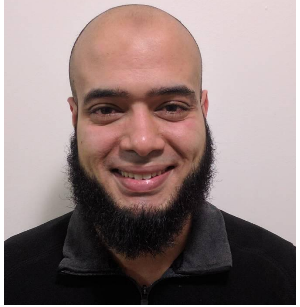
Machine Learning
Development and Training

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Standardizing the Development Environment

- Crucial for ensuring reproducibility, consistency, and team collaboration
  - Avoid "it works on my machine" problem
- Key items
  - Virtual Environments
    - Conda and Virtualenv
      - pip freeze > requirements.txt
      - pip install -r requirements.txt
      - conda env export > environment.yml
    - Note: alone is not enough. You will face issues (e.g. Non-Python Dependencies)
  - **Containerization**: Use **Docker** containers to encapsulate your development environment.
    - **all dependencies are packaged together**

Standardizing the Development Environment

- Develop your project without local development environments
- Pros
  - Accessibility, easily standardized, Scalability, Automatic Backups, Resource Management
- Cons
  - **Latency**: Internet connection speed can affect performance
  - Security Risks: pros (fast fixes), cons (data off-site for rented cloud)
- Cloud Development Environments
  - AWS Cloud9, Visual Studio Codespaces

Visual Studio Code: remote coding and debugging

- Personally I mix between VS and remote desktop view (on PyCharm)

Docker

- Docker is a platform for developing, and running applications in containers.
    - A container is an executable package that includes **everything** needed to **run** the code
- Containers are based on an **image**, which is an executable package that includes **everything** needed to run the code
    - **Dockerfile**: A text document that contains all the commands you would normally execute manually to **build a Docker image**.
    - Docker **Container**: A runtime instance of a Docker image
- **Portability**: Since containers encapsulate all dependencies, they can be easily moved between different systems and clouds.

Complex Model Development Environment

create_dataset()
create_model()
build_workflow()

calls

Complex Framework
Inversion of Control

Reserve 4 GPus

Cluster
- 200 GPUs
- 1 Terabyte ram

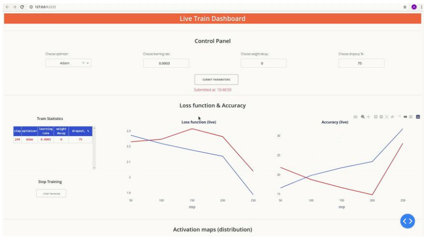
GitHub

GitLab

docker

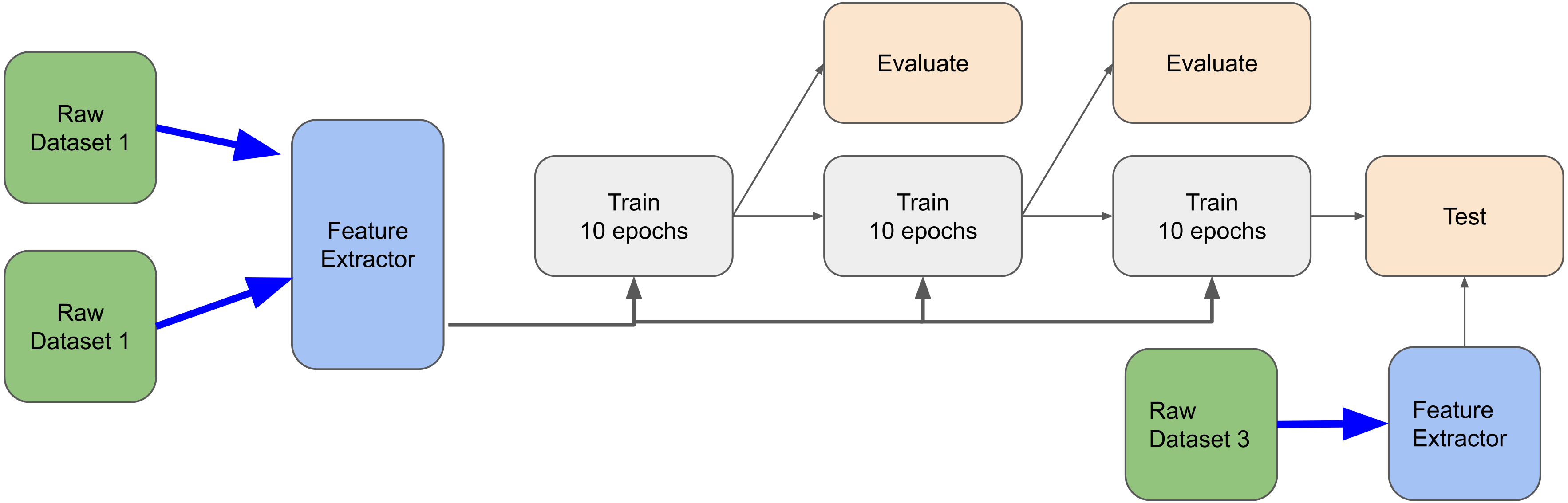
Versions!

Database



Training Workflows

- **Workflow**: **Defines** what series of tasks to run (**DAG**)
  - **Scheduler**: Focuses on **when** tasks should be executed on **which** resources
  - **Orchestrator**: Manages **how** tasks/resources are coordinated and executed (e.g. Apache Airflow)



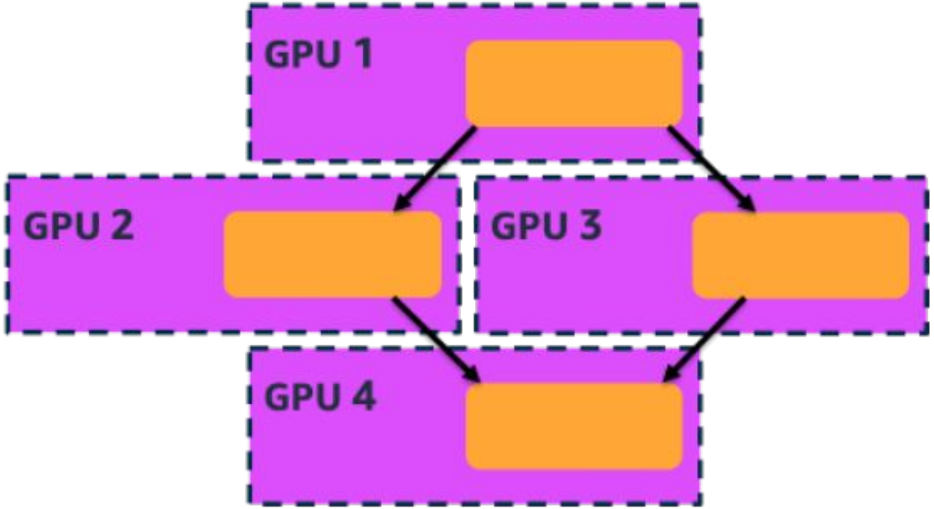
Distributed Training

- In large scale data, we parallelizing the process across multiple machines, CPUs, or GPUs to speed up as much as we can
  - Data can be easily broken into multiple sub-datasets
  - Deep learning models also can be
    - Seperated in multiple stages
    - Separate parallel branches
- Behind the scenes: algorithms for **communication** between the nodes
- Challenges
  - Network Latency: Sending and receiving updates between machines can introduce delays.
  - Balancing Workload: Not all tasks can be perfectly parallelized, leading to imbalances in workload.

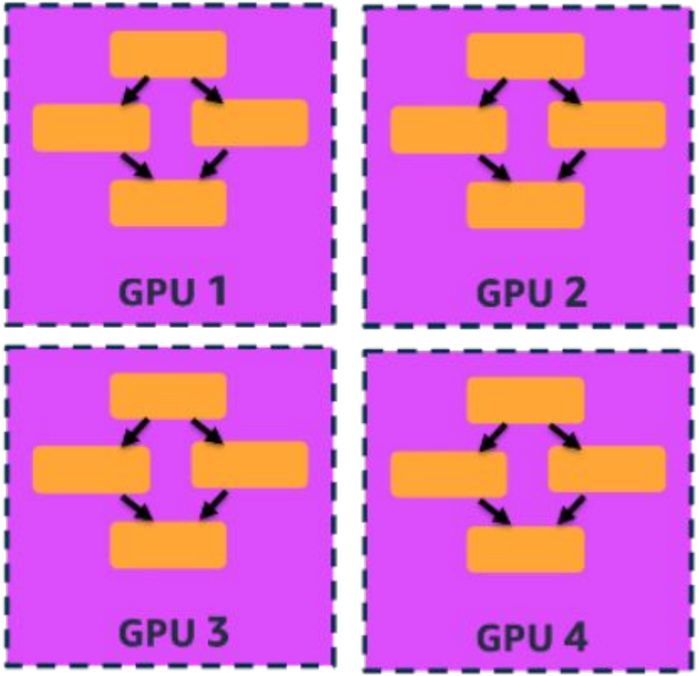
Distributed Training

- **In data parallelism**, each machine in the distributed system works on a different **subset** of the dataset.
    - model parameters are **shared** and updated collectively (most common)
- In **model parallelism**, different machines work on **different parts** of the model itself
    - Useful for very very deep models that don't fit in the memory of a single machine
        - Mainly also due to the big batch size (not just the model itself)
- **Hybrid Parallelism** This is a combination of both data and model parallelism

Model Parallelism



Data Parallelism



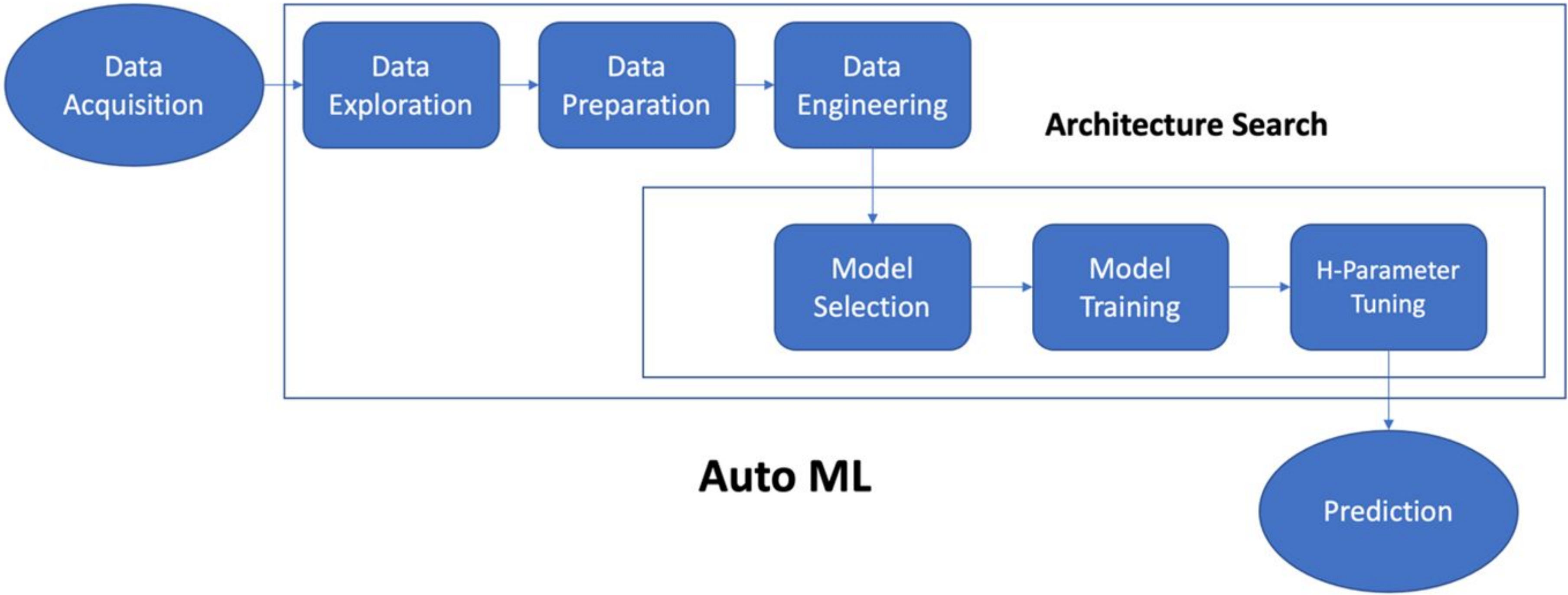
Img [src]

Multithreading

- Multithreading can be used in ML pipeline
  - Debugging tip: cancel threading and work on one thread for issues
- Data Preprocessing
  - Data Augmentation: Image or text data can be augmented on-the-fly using multiple threads
- Feature Engineering
  - Feature Calculation for heavy features / tokenization, stemming, and lemmatization
- Model Training
  - One thread is training on GPU while another is loading the next data batch
  - We may have several models: thread per model for inference queries
- Model Evaluation: Different threads can handle different folds in k-fold cross-validation

AutoML

- Automated Machine Learning (AutoML) refers to the process of automating the end-to-end process of applying machine learning to real-world problems
  - **Data Preprocessing**: e.g. handling missing values, outlier detection
  - **Feature Engineering**: generates new features from existing ones and selects the most relevant features
  - **Model Selection**: automatically choose the best machine learning algorithm
  - **Hyperparameter Tuning**: automatically optimizes model hyperparameters
  - **Model Evaluation**: automated compare metrics (e.g. f1-score)
  - **Neural Architecture Search (NAS)**: automating the design of neural network architectures
    - Cons: significant computational power and time: search space, training, etc


Data Acquisition
Data Exploration
Data Preparation
Data Engineering
Architecture Search
Model Selection
Model Training
H-Parameter Tuning
Auto ML
Prediction

Img src

Tips

- Start in a simple way to POC
  - Simple features before complex ones ⇒ Build a baseline
  - A strong ready pretrained model (e.g. BERT)
  - Be careful from paper SOTA (typically fails in production)
  - 20% 80% rule applies. 20% effort brings 80% of customer satisfaction.
  - *Later go to a complex model*
- Do proper algorithms comparison
  - Similar data
  - Similar experimentation effort
- Keep updating data (train, kpi) - keep training and evaluating
- Each model has its own assumptions
  - E.g. IID (neural network), smoothness (supervised learning), linear data (linear regression)

Tips

- Be Metrics Oriented
  - Keep identifying metrics that can help your ML model
  - E.g. in social network: expands per read, reshares per read, etc
- Push more toward ML
  - Can something be learned effectively within the DNN? Do it
  - ML vs complex heuristic? ML it
- Migrations: Be very careful
  - Migrating from old code base (infrastructure) to new code base (infrastructure)
- Logging fully and properly
  - MistakeL Logged what post the user clicked but did not log what is not clicked!
  - Log critical messages with all info you need for **debugging and labeling**

Tips

- Document the invented features
  - Maintain a document about the used features, especially the invented ones
    - What is the intuition about them? Performance with and without
- Specific features may work
  - We typically discard features that doesn’t correlate with the output
  - What if a group of features correlate one for subset of data? Use them
  - Make sure the overall features cover most of your data
- Remove useless features
  - We keep improving our model. If some features are now useless, remove them!

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."