Machine Learning Model Management and Inference (MLMMI)
05 Model Serving Systems

Dr. Arnab Phani
BIFOLD, TU Berlin
DEEM Lab

20.05.2026

Announcements

- **No Lecture on June 03**
  - I will be in India to attend SIGMOD
- **Feedback on Nils's lecture**
  - Did you learn something new?
- **Assignment 1 Grades**
  - Mean: 9.71; min: 6.0; max: 10.0
- **Invited Talk 2: June 10**
  - Matthias Seeger (https://mseeger.github.io/)
- **Open-weight LLM API Access**

https://chat-ai.academiccloud.de/

Chat AI Access

1. **Log into** https://academiccloud.de/services/chatai/ **at least once** with your TU Berlin account
2. **Click on Book** at https://kisski.gwdg.de/en/leistungen/2-02-llm-service
3. **Fill out the form**, here is some required information

   Group leader: Prof. Dr. Sebastian Schelter
   Working group/ department: FG Management of Data Science Processes
   Organisation: Technische Universitaet Berlin
   Website: https://deem.berlin
   Address: Technische Universität Berlin, FG Management of Data Science Processes, Sekr. FR 7-3, Franklinstr. 28 / 29, 10587 Berlin, Germany
   Requirements: <briefly describe that you need LLMs for a course project>
   Critical data in regards to GDPR or ISO27001 is processed: No

Agenda

- **Model Serving Systems**
- **Serving Optimizations**
- **Exercise 3: Model Serving Systems**
- **Projects Overview**

Model Serving Systems

Serving Overview

- **Serving Constraints**
  - Low latency (under SLO); user facing applications
  - High throughput
  - Improved accuracy (model selection, ensemble)
  - Scalability during high loads
- **Model/System Diversity**
  - Diverse ML models
  - Varying latency (SVM vs. DNNs)
  - Different frameworks (e.g., TF serving)
  - Hardware (CPUs, GPUs, edge devices)
- **Necessity of Serving Systems**
  - System-level optimizations
  - Scheduling and resource management

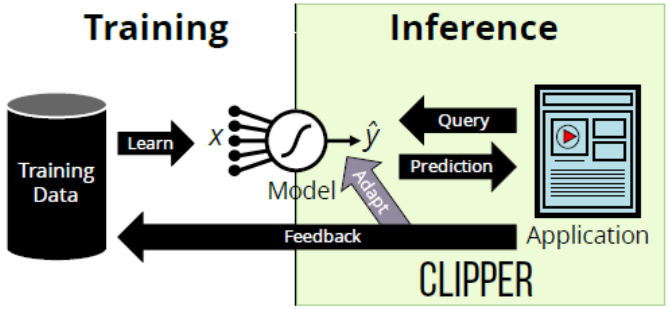


Figure 2: Machine Learning Lifecycle.

Example Applications

- Real-time speech translation
- Object-recognition in self-driving cars

ML Serving Systems

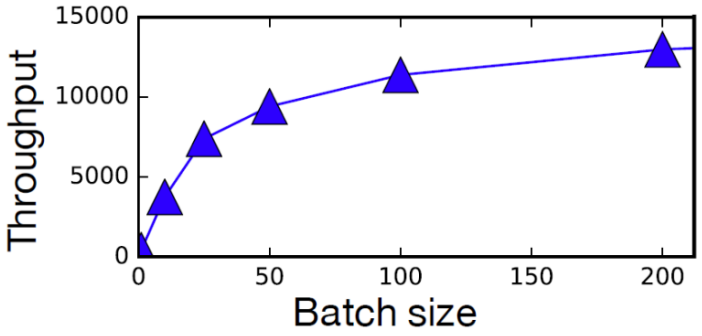
- **Embedded ML Serving**
  - TensorFlow Lite: small footprint, dedicated HW acceleration, APIs, models: MobileNet, SqueezeNet
  - TorchScript: Compile Python functions into ScriptModule/ScriptFunction
  - IREE: data centers / edge, small footprint

Google Translate
140B words/day → 82K GPUs in 2016

- **ML Serving Services**
  - Motivation: Complex models run on dedicated HW
  - RPC/REST interface for applications
  - TensorFlow Serving: configurable serving w/ batching
  - TorchServe: Specialized model for HW, batching/parallelism
  - Clipper: Decoupled multi-framework scoring, w/ batching and result caching
  - Pretzel: Batching and multi-model optimizations in ML.NET
  - Rafiki: Optimizations for accuracy s.t. latency constraints, batching, multi-model opt

Serving Optimizations

Batching and Caching



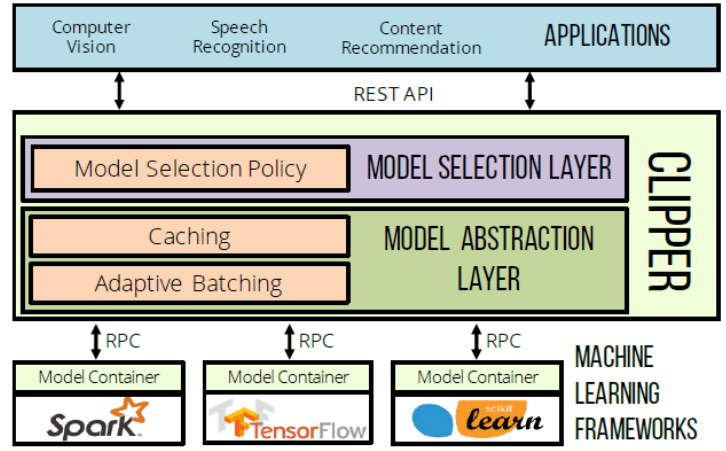
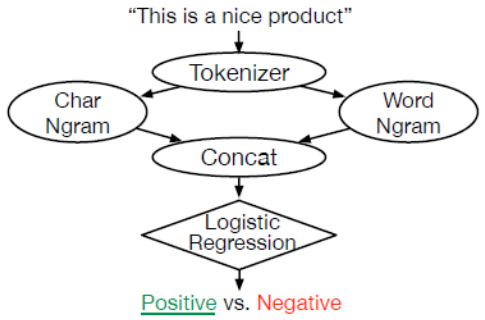


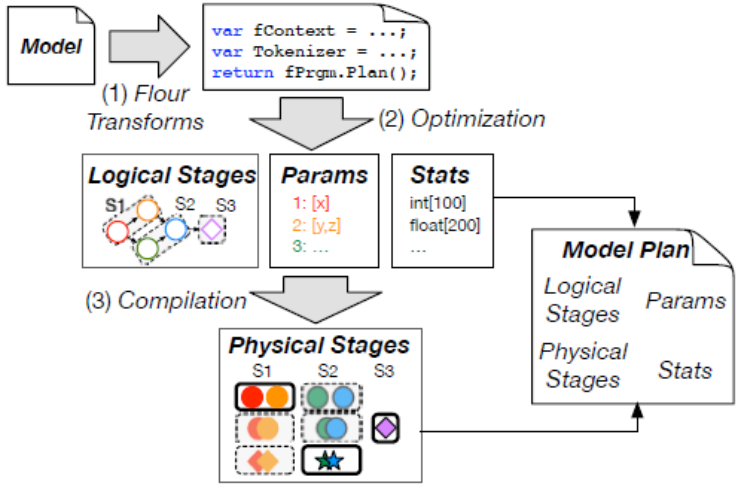
Figure 1: The Clipper Architecture.

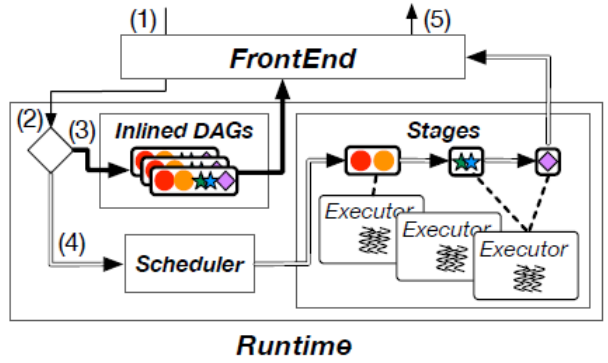
- **Motivation**
  - Improves throughput, utilize HW efficiently
  - Amortize RPC calls overhead and enable data-parallel optimizations (GPUs / BLAS)
  - Popular items are requested frequently
- **Clipper**
  - Latency-throughput trade-off; user-provided latency SLO
  - Dynamic batch size via additive-increase-manipulative-decrease (AIMD)
  - Delayed batching
  - Cache and reuse model-specific inference

* Clipper: A Low-Latency Online Prediction Serving System. NSDI 2017.

Multi-Query Optimization



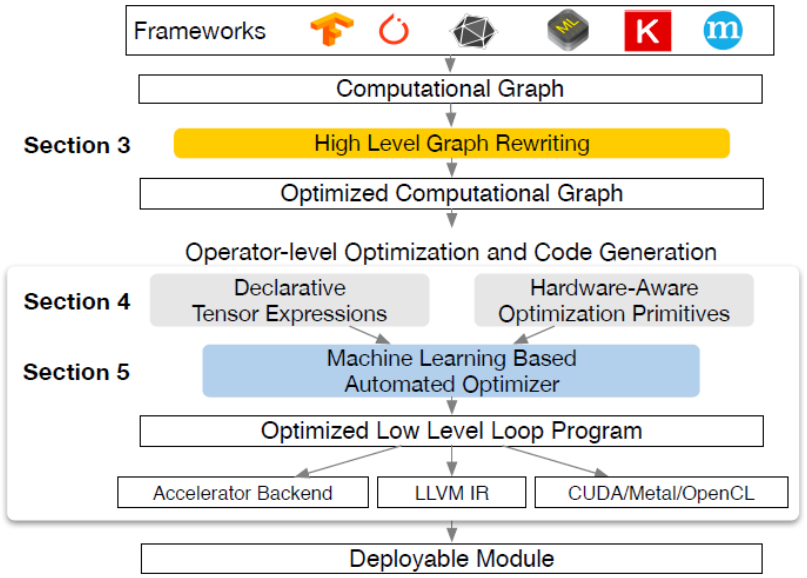




- **Pretzel**
  - Problems: duplicate preprocessing, memory wastage due to containerization, coarse-grained scheduling
  - White-box prediction serving with end-to-end optimization
  - Optimizing multiple pipelines is cast as multi-query optimization
  - Logical rewrites: CSE, branch removal, merging stages, model pushdown
  - Runtime optimizations: schedular, sub-plan materialization, caching, object store

* Pretzel: Opening the Black Box of Machine Learning Prediction Serving Systems. OSDI. 2018.

Compilation



- **Overview**
  - Diverse HW characteristics of target devices
  - Compile entire graph to HW-specific binary functions
  - Large optimization space: diverse memory access, threading patterns, primitives, codegen optimizations (loop tiling, caching, unrolling)
  - MLIR-based compilation
  - Operator fusion; data-layout transformation; nested parallelism;

```
A = t.placeholder((1024, 1024))
B = t.placeholder((1024, 1024))
k = t.reduce_axis((0, 1024))
C = t.compute((1024, 1024), lambda y, x:
                t.sum(A[k, y] * B[k, x], axis=k))
s = t.create_schedule(C.op)
```

```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

+ Loop Tiling

```
yo, xo, ko, yi, xi, ki = s[C].tile(y, x, k, 8, 8, 8)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
              A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

+ Cache Data on Accelerator Special Buffer

```
CL = s.cache_write(C, vdla.acc_buffer)
AL = s.cache_read(A, vdla.inp_buffer)
# additional schedule steps omitted …
```

+ Map to Accelerator Tensor Instructions

```
s[CL].tensorize(yi, vdla.gemm8x8)
```

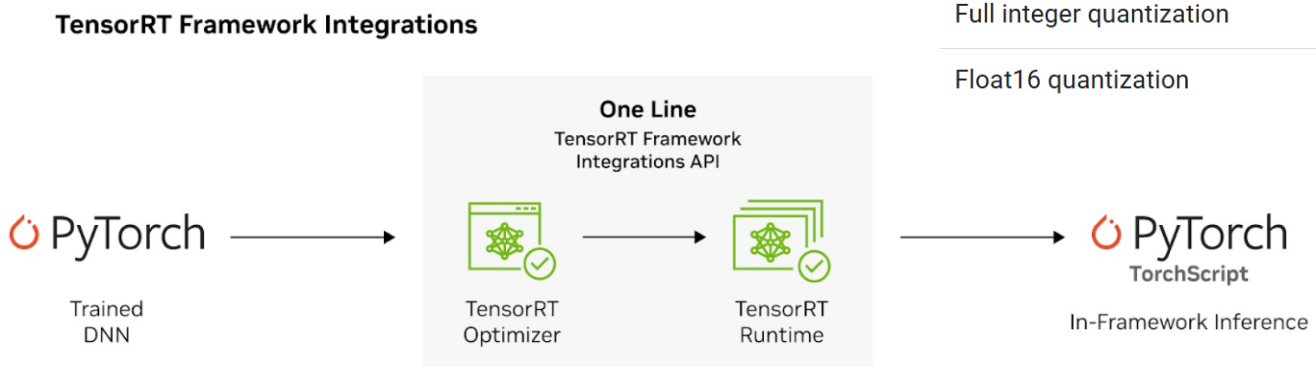
```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
    vdla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

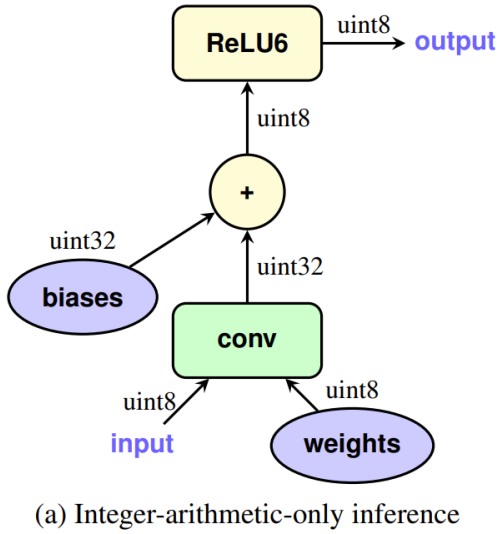
schedule → schedule transformation ⇢ corresponding low-level code

* TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. OSDI. 2018.

Resource-Efficient Inference

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

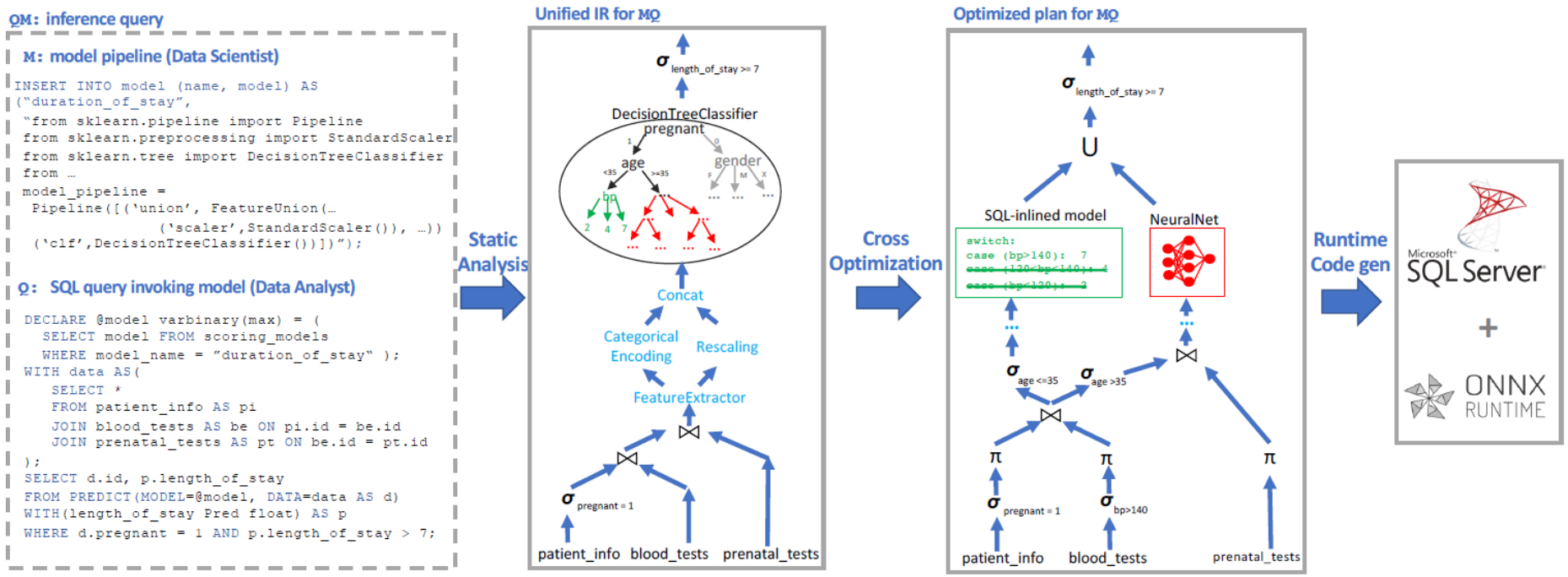




(a) Integer-arithmetic-only inference

- **Quantization**
  - Lossy compression via ultra-low precision / fixed-point
  - Ex.: 62.7% energy spent on data movement
  - Quantization of model weights, and sometimes also activations (e.g., UNIT8)
  - reduced memory requirements and better latency / throughput (SIMD)
- **Distillation (Model Compression)**
  - Train a compact “student” network to mimic a larger “teacher” network’s output probabilities

In-Database ML Serving



"Find pregnant patients with a high likelihood of staying in the hospital for more than a week"

- **Raven**
  - Problems: data movement, serialization, security, lack of transactional support
  - Integrating ML serving deep into SQL with a unified IR and many optimizations
  - Cross-optimizer: predicate-based model pruning (pushing projection to model), model splitting, NN translation, standard DB optimizations

* Extending Relational Query Processing with ML Inference. CIDR 2020.

Serverless Inference

- **Serverless Computing**
  - FaaS: functions-as-a-service (event-driven, stateless input-output mapping)
  - Infrastructure for deployment and auto-scaling of APIs/functions (Amazon Lambda, Azure Functions)
- **Serverless Inference**
  - Challenges: Cold starts (model loading overhead), large ML models, no batching
  - Memory size: 1 GB to 6GB
  - Provisioned concurrency to keep instances warm

* https://docs.aws.amazon.com/sagemaker/latest/dg/serverless-endpoints.html

Summary

- Latency, throughput, accuracy trade-off of model serving
- Black box and white box serving optimizations
- Model compilation and compression for low latency/edge devices
- In-DB inference to avoid data movement
- Serverless inference benefits and challenges

Please read the recommended papers

Exercise 3: Model Serving System

Implement & Optimize a Model Serving System

- **Motivation**
  - Implement a model serving system
  - Realize latency – throughput trade-off and optimize accordingly
- **Inference Server Baseline**
  - Load four models to the model registry from assignment 1
  - Maintain a request queue for serving requests and invoke models
- **Black-box Optimizations**
  - Implement batching and prediction reuse (model-specific)
- **White-box optimizations**
  - CSE of preprocessing within a batch, embedding reuse
- **Benchmark**
  - Varying # serving requests per second

Deadline: 01.06.2026

Fixed use case

Text topic classification on the 20 Newsgroups dataset.

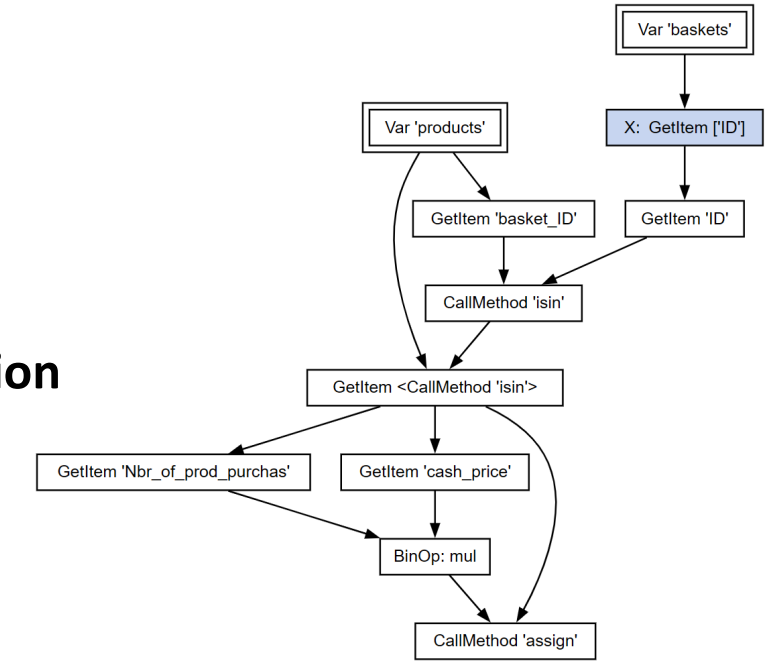
- **Preprocessing (the expensive shared step):** a forward pass through `sentence-transformers/all-MiniLM-L6-v2`, producing a 384-dim sentence embedding.
- **Heads:** four models trained on top of those embeddings, all interchangeable behind the same encoder:
  - `logreg` — logistic regression
  - `rf` — random forest
  - `hgb` — HistGradientBoosting (one-vs-rest)
  - `mlp` — small PyTorch MLP head (the DNN)

Projects Overview

LLM-Assited/Agentic Frameworks

- **1. Skrubifier: LLM-Assisted Conversion of ML Pipelines to Skrub DataOps**
  - AI system to convert ML pipelines to skrub DataOps format
  - Tabular ML pipelines
  - Provide a robust framework
  - Provide 10 converted pipelines runnable on skrub/stratum

- **2. AIDE-skrub: Extending AIDE Agent for Skrub DataOps Generation**
  - Extend AIDE to generate skrub DataOps pipelines
  - Add LLB-based component or a sub-agent
  - Provide the modified AIDE
  - Provide 10 generated pipelines runnable on skrub/stratum



Tabular Pipelines:
https://github.com/openai/mle-bench/tree/main
https://www.kaggle.com/code/sudalairajkumar/winning-solutions-of-kaggle-competitions

Stratum Internals

- **3. Stratum Optimizer: Rule-Based Rewrites for stratum**
  - Add a collection of rule-based rewrites and unit tests
  - 5 rewrites / each group member
  - Create a pull request for each rewrite

- **4. High-Performance Rust Kernels for Stratum**
  - High-performance Rust implementation for common operators
  - Support limited hyperparameters
  - At least 2x speedup over sklearn
  - 2 operators / each group member
  - Create a pull request for each operator

Stratum Code and Paper:
https://github.com/deem-data/stratum
https://arxiv.org/pdf/2603.03589

Exemple Rewrites

eliminate add(x, 0)
eliminate mul(x, 1)
eliminate sub(x, 0)
eliminate div(x, 1)
rewrite log(1+x) to log1p(x)
rewrite exp(x)-1 to expm1(x)
push projections below unary column-wise ops
push filters to join inputs

Exemple Operators

```
OrdinalEncoder
SimpleImputer
TargetEncoder
RobustScaler
MinMaxScaler
LogisticRegression ElasticNet/Lasso:
coordinate descent
DecisionTreeClassifier
DecisionTreeRegressor
```

AutoML Framework

- **5. AutoML on Skrub DataOps DAGs (Two Independent Groups)**
  - Inputs: Tabular Datasets, problem description, YAML specification
  - YAML spec contains candidate preprocessing techniques and models
  - Parse YAML to skrub DataOps DAG
  - Iteratively prune pipelines (e.g., successive halving)

- **Skrub and stratum Tutorials**
  - Elias will present in a consultation hour

- **Miscellaneous Points**
  - Independent pipelines across students/groups/projects
  - Only tabular datasets for simplicity
  - You will be graded for both framework and pipelines (projects 1, 2, 5)