FASTRADE
FPGA-Accelerated Strategy & Trading for Enhanced Decisions

Weitao Lu, Wenbo Liu, Xiaolei Zhao, Yixuan Li

Content

I. Introduction

II. Hardware-Software Interfaces

III. Hardware

IV. Software

V. Conclusion

VI. Demo

Background

- **Raw Features:**

from Kaggle dataset

Date	# Open	# High	# Low	# Close	# Volume
2021-01-13 00:00:00-05:00	90.9000015258789	103.0	90.01000213623047	96.36499786376953	25159000
2021-01-14 00:00:00-05:00	103.5	137.97999572753906	101.0	114.94000244140625	20858700

- **Factors:**

Price_Momentum	Volume_Factor	RSI	30d_Moving_Average
-0.0342966884080463	0.988596441118506	43.9104444455574	111.486499532064
-0.204454493656199	1.11369138843955	26.8411671915214	110.70316619873
-0.215384589301215	1.10666654750281	17.1911588916814	109.86316630045600

- **Factor Model:** **Multi-Factor Model (5 Raw Features + 4 Factors)**

Reference:

- [Rendle, 2010] Steffen Rendle. Factorization machines. In ICDM, 2010.

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Motivation

- **FPGAs are widely used at quant companies.**

  But quantitative researchers prefer Python.

We are interested to reproduce the investment pipeline on FPGA

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

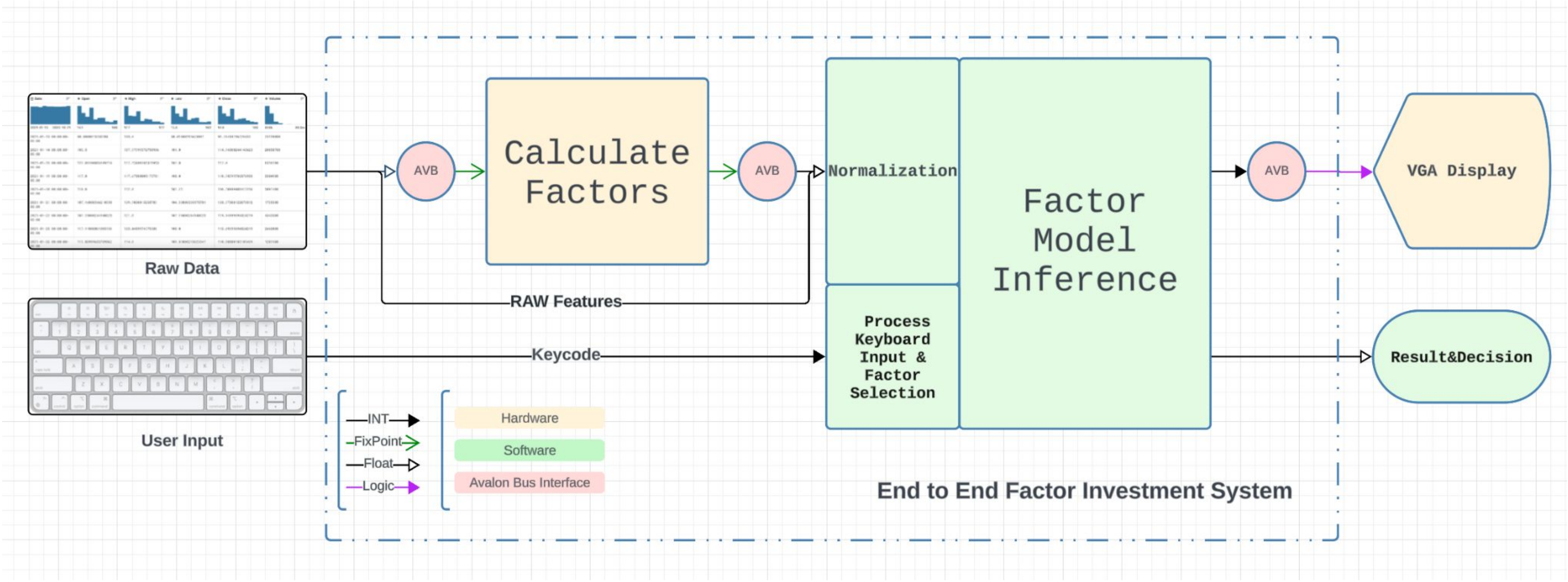
Outcome

A Well Composed End-2-End Multi-Factor Investment Pipeline :

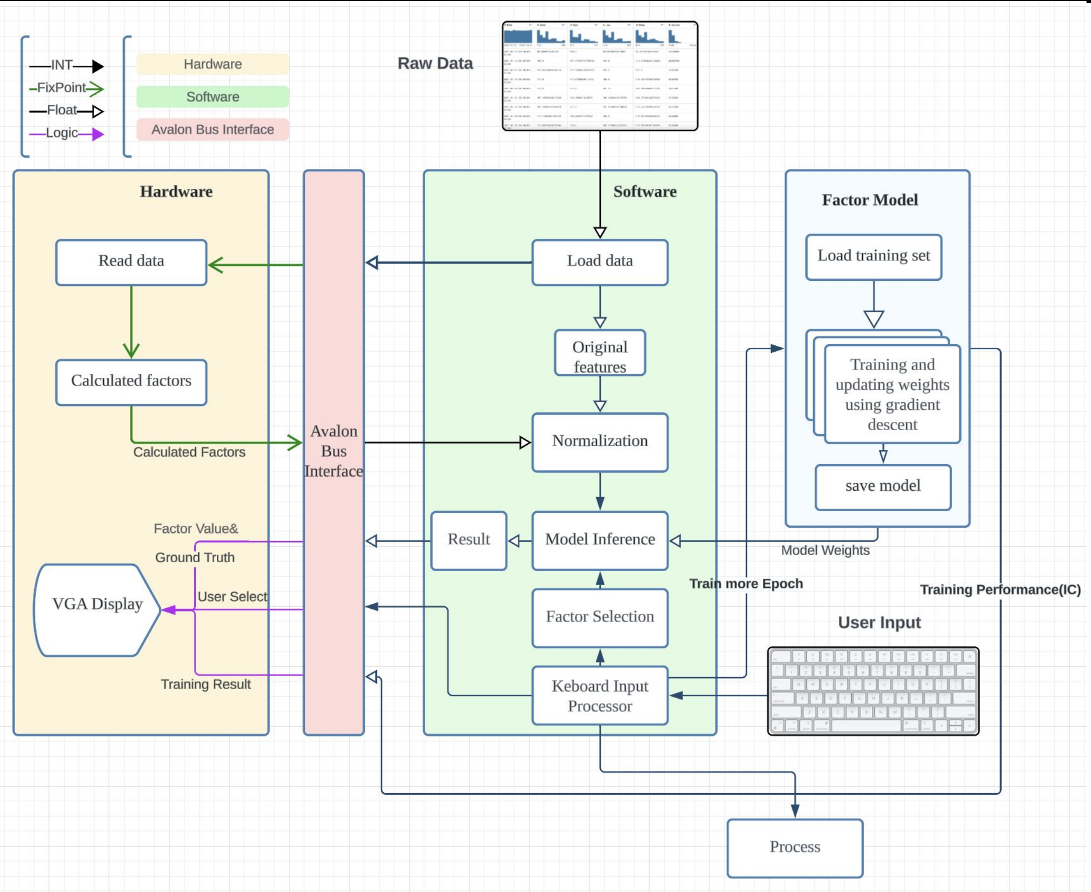
- **Components:Hardware Factor Calculation, Software model training/inference**
- **Well designed Hardware/Software data interfaces**
- **User-Friendly VGA display and Keyboard interaction.**
- **Verified hardware calculation waveform and acceleration improvement**

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

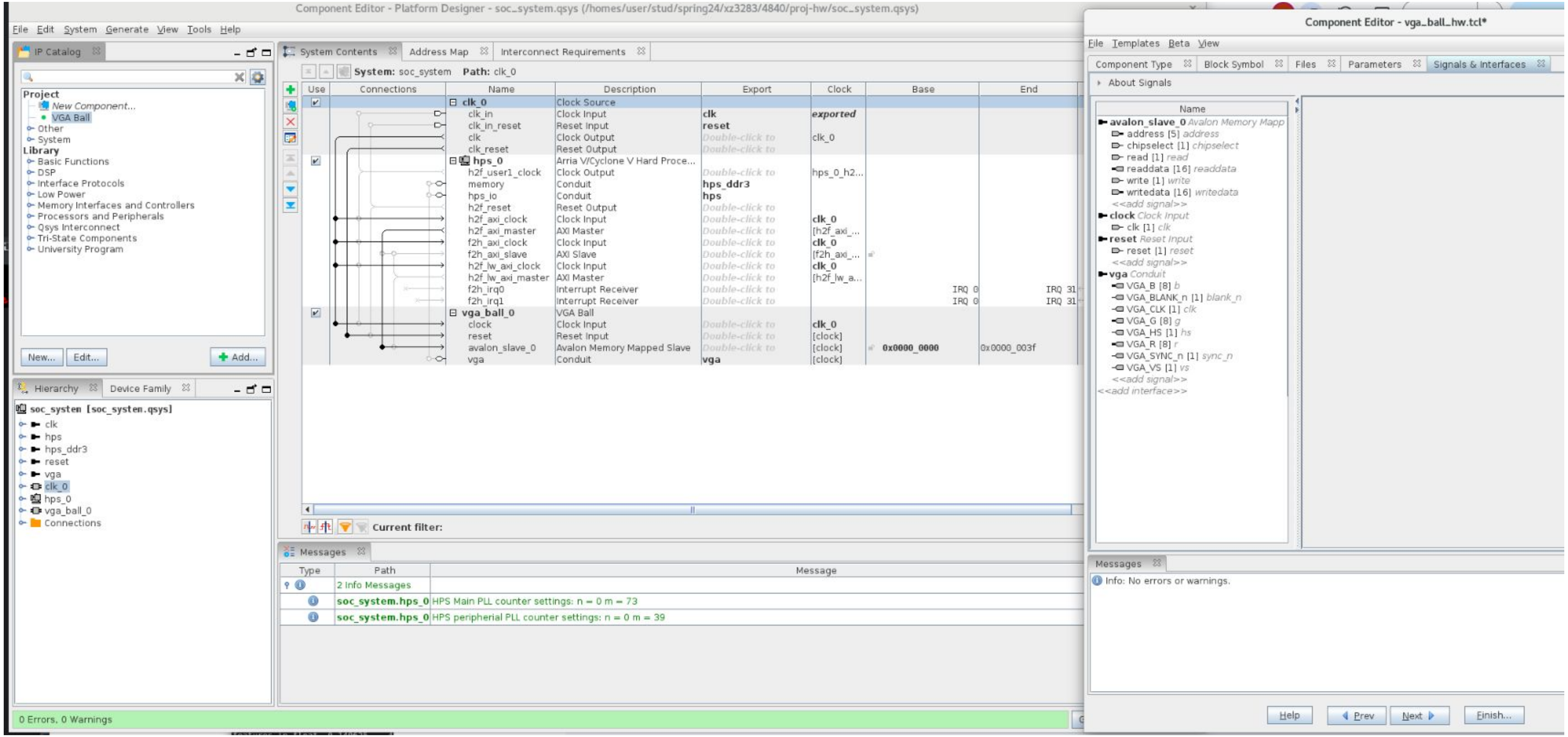
End2End System Overview



Detailed System Design



Hardware - Software Interfaces



Address Map - VGA

addr 0	addr 1	addr 2	addr 3	addr 4	addr 5	addr 6	addr 17	addr 18	addr 19
Status	Actual Return	Actual Return	IC	IC	IC	Sign	Factor	Factor	Cursor

Bit 0	Bit 1
Sign for Actual Return	Sign for Factor

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Address Map - Calculator

Write to Hardware

addr 7	addr 8	addr 9	addr 10	addr 11
Open	High	Low	Close	Volume

Read from Hardware

addr 12	addr 13	addr 14	addr 15	addr 16
Momentum	Volume	RSI	MA	Done

Division Module

This module was designed for signed division of fixed-point numbers.

Inputs: Dividend, Divisor
Outputs: Quotient, Signal Completion, Busy, Divide by zero flag

Used Gaussian Rounding to implement the module and handled some special cases, including divided by zero.

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Calculator Module

This module was designed to calculate the factors.

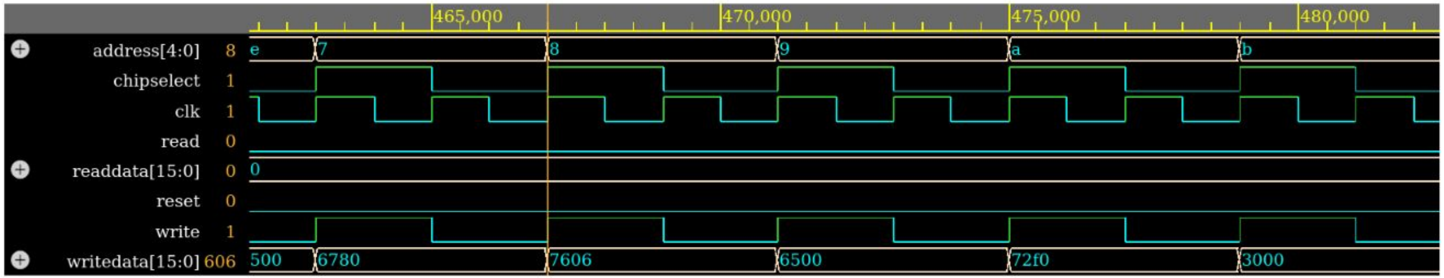
Use a 2D register array to store the original data and the calculation results(factors).

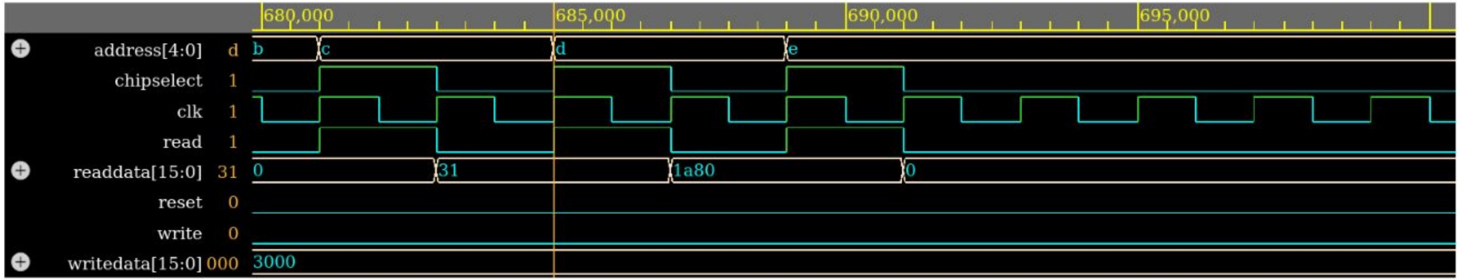
The hardware will read or write data according to the signals.

Only when all the five base data have been received, the module will start to calculate the factors.

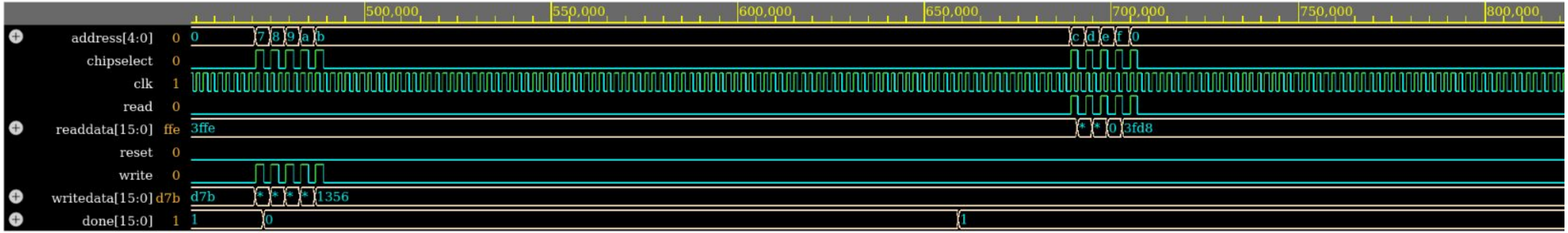
Only when all the factors of one day have been calculated, the software part can read the results.

Calculator


465,000
470,000
475,000
480,000
address[4:0] 8
e
7
8
9
a
b
chipselect 1
clk 1
read 0
readdata[15:0] 0
0
reset 0
write 1
writedata[15:0] 606
500
6780
7606
6500
72f0
3000


680,000
685,000
690,000
695,000
address[4:0] d
b
c
d
e
chipselect 1
clk 1
read 1
readdata[15:0] 31
0
31
1a80
0
reset 0
write 0
writedata[15:0] 000
3000

Calculator

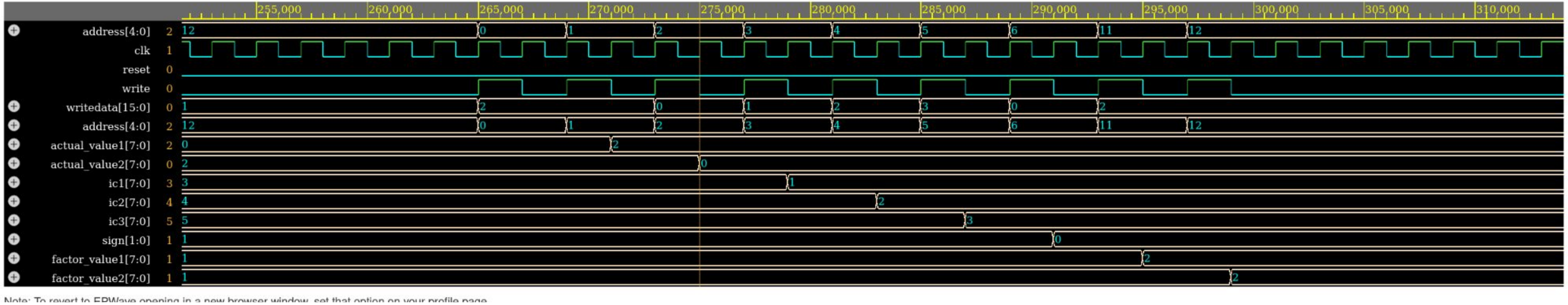

500,000
550,000
600,000
650,000
700,000
750,000
800,000
address[4:0] 0
chipselect 0
clk 1
read 0
readdata[15:0] ffe
reset 0
write 0
writedata[15:0] d7b
done[15:0] 1
0
7
8
9
a
b
c
d
e
f
0
3ffe
3fd8
d7b
1356
1
0
1

VGA

- Resolution: 680 * 480
  - 80 columns, 60 rows
- 50 character bitmaps: 8 * 8 bits/each
- Efficient storage(only store bitmaps)

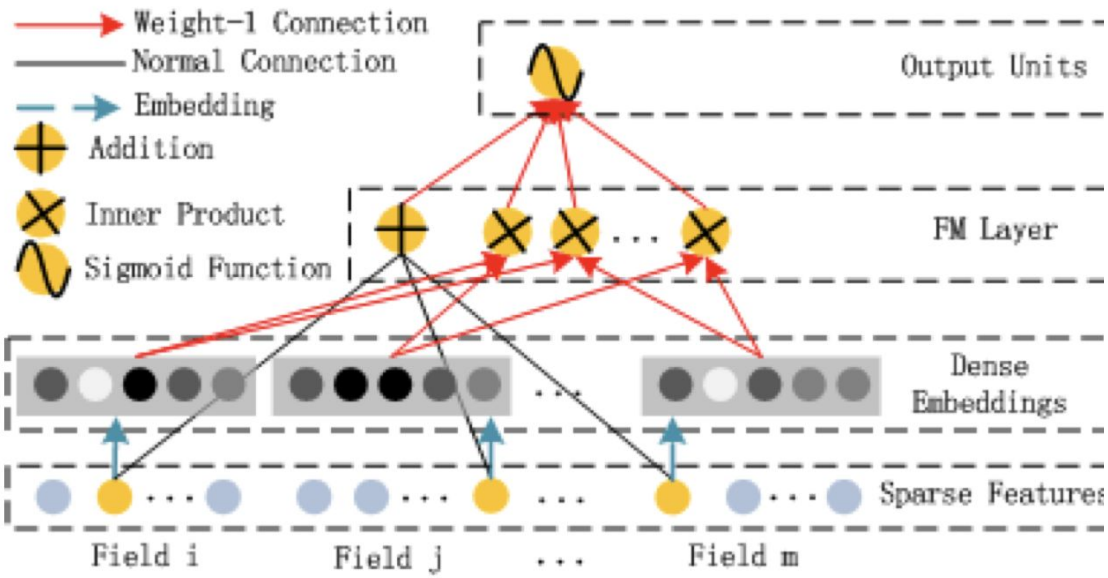
```
char_a[0] = 8'b00011000; //    **
char_a[1] = 8'b00100100; //   *  *
char_a[2] = 8'b01000010; //  *    *
char_a[3] = 8'b01111110; //  ******
char_a[4] = 8'b01000010; //  *    *
char_a[5] = 8'b01000010; //  *    *
char_a[6] = 8'b01000010; //  *    *
char_a[7] = 8'b01000010; //  *    *

char_b[0] = 8'b01111100; //  *****
char_b[1] = 8'b01000010; //  *    *
char_b[2] = 8'b01000010; //  *    *
char_b[3] = 8'b01111100; //  *****
char_b[4] = 8'b01000010; //  *    *
char_b[5] = 8'b01000010; //  *    *
char_b[6] = 8'b01000010; //  *    *
char_b[7] = 8'b01111100; //  *****

char_c[0] = 8'b00111100; //   ****
char_c[1] = 8'b01000010; //  *    *
char_c[2] = 8'b01000000; //  *
char_c[3] = 8'b01000000; //  *
char_c[4] = 8'b01000000; //  *
char_c[5] = 8'b01000000; //  *
char_c[6] = 8'b01000010; //  *    *
char_c[7] = 8'b00111100; //   ****
```

VGA



COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

- **Factor Model**



COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Software Problems and Solution

Everything works fine on Python, yet hard to program in C.

Lack of machine learning libs: -> Reproduced every from scrach by stand C libs.

```c
// matrix.h
#ifndef MATRIX_H
#define MATRIX_H

typedef struct {
    double **data;
    int rows;
    int cols;
} Matrix;

Matrix create_matrix(int rows, int cols);
void free_matrix(Matrix m);
Matrix multiply_matrices(Matrix a, Matrix b);
void elementwise_multiply(Matrix a, Matrix b, Matrix result);
double sum_matrix_elements(Matrix m);
void init_random_normal(Matrix m, double mean, double stddev);
double sigmoid(double x);
void compute_means(Matrix features, double *means);
void compute_stddevs(Matrix features, double *means, double
*stddevs);
void standardize_features(Matrix features, double *means, double
*stddevs);
double pearson_correlation(double *x, double **y, int n);
#endif
```

```c
// fm_model.h
#ifndef FM_MODEL_H
#define FM_MODEL_H

#include "matrix.h"
#include <stdbool.h>

typedef struct {
    double bias;    // Scalar bias term w0 in the model
    Matrix weights; // Weight vector w in the model
    Matrix factors; // Factorization matrix V in the model
} FMModel;

void fit(FMModel *model, Matrix X, Matrix y, int feature_potential,
          double alpha, int iter, int batch_size, double decay_rate);
double *predict(FMModel *model, Matrix X);
double *predict_active(FMModel *model, Matrix X, Matrix Ref ,bool
*active_features);
#endif
```

Software Problems and Solution

Everything works fine on Python, yet hard to program in C.

Overflow during gradient descent: -> Improved Training Strategy

- **Various Feature Scale, too large data:**
  - Zscore Normalization

- **Overflow caused by gradient accumulation:**
  - Mini-batch Gradient Descent
  - Learning Rate decay

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Evaluation

- Hardware(Python runs on Google Colab)

Calculation Time using Python(Colab)	Calculation Time using Python(M1)	Calculation Time using SV
0.0050289 s	0.0014479 s	0.0018 s

- Software

Python Training /20epoch(Colab)	Python Inference Time(Colab)	Python Training /20epoch(M1)	Python Inference Time(M1)	On Board Training /20epoch	On Board Inference Time
1.103962 s	0.000150 s	0.225156 s	0.000026 s	0.130262 s	0.000035 s

*M1 refers to Apple M1 Pro and Colab refers to Google Colab's CPU

Demo

- Blue - current place of cursor
- Green - selected features
- Train - train the model
- Test - prediction

- Press ↑ and ↓ to move cursor
- Press enter to select/unselect feature

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Thank You!