A High-Level Review: ML Horse Race Project

September 30, 2024

Wonhee, Jeff, and Kevin

STANFORD BUSINESS GRADUATE SCHOOL OF

Agenda

- Background
- Data Preparation
- Models/Tools
- Evaluations
- Scalability
- Challenges
- Discussion

Background

*I am at the early stages of considering several projects that would require a sophisticated machine learning trading model… The basic idea for the first project is **a horse race against mutual funds over the last 30 years.** In event time, we try to beat mutual funds using only **SEC filings, prices, and machine learning.** We then introduce constraints not the model to mimic mutual funds' investment styles and universes.*

*I would appreciate speaking with someone at DARC about the **potential of such models and what developing one would entail**. We will make a decision about the project after that.*

Can we develop a model that surpasses mutual funds' historical performances with current technology and public data?

How does our trading strategy translate model return rates into monetary gains to assess its financial impact?

I'm pleased to share that Prof. Miao Liu (cc'd) from Boston College has joined the project. We believe his experience in machine learning will be very valuable. He has shared preliminary code, data, and results that we could use to kickstart our work.

Goals & Deliverables

- **Phase 1**
  - Conduct literature review and develop a proof-of-concept for a Random Forest (RF) regression model
  - Provide working code for the RF model for the faculty to implement

- **Phase 2**
  - Refine the RF model by addressing time-series nature of data to prevent look-ahead biases and conducting further hyperparameter tuning
  - Provide training to Miao and his RA for a smooth transition

- **Phase 3**
  - Rerun the RF model with updated datasets to identify and resolve any issues
  - Build a Neural Network (NN) model using the updated datasets and compare the results
  - Provide updated quarterly and monthly outputs for the RF and NN models

Data Preparation

Input

- Years: 1980 - 2020
- Sources: Compustat, CRSP, I/B/E/S (160 variables)
- Content: Firm characteristics, macroeconomic and investor sentiment data
- Versions : 3
- Size: 1.4 GB

Output

- Quarterly and monthly stock return estimates
- Decile portfolios
- Individual stock rankings within each date/portfolio

Faculty

Provide the input data and initial code

- None-PIT (Point-In-Time) data, followed by PIT data with additional variables
- Missing value imputation, winsorization, and feature scaling over the entire dataset (data leakage)

DARC

Independently preprocess the input data for each training and test period

- Mean imputation for missing values
- Winsorization at the top and bottom 5 percentiles
- Feature scaling

Data Transformation for Neural Network

- Transform ‘permno’ variable as Pytorch embedding variable
- For a given year such as the year of 2016, create 4 different datasets:
    - Training data: 1980 - 2014
    - Validation data: 2015
    - Retraining data: 1980 - 2015
    - Prediction data: 2016

Data Transformation for Neural Network

- Create 4 custom Pytorch Dataset objects
  - Allow us to define how to load the data, including preprocessing, augmentation, or custom transformations
- Create 4 custom Pytorch Dataloader objects
  - Takes care of effectively and efficiently loading data from the Dataset
    - Batching
    - Shuffling
    - Parallel data loading
    - Handling last batch of different sizes
- By separating data handling into Dataset and Dataloader, the code is more modular and scalable so we can focus on model architecture and optimization

Models/Tools

RF

A type of ensemble learning method based on decision trees that can make robust prediction results

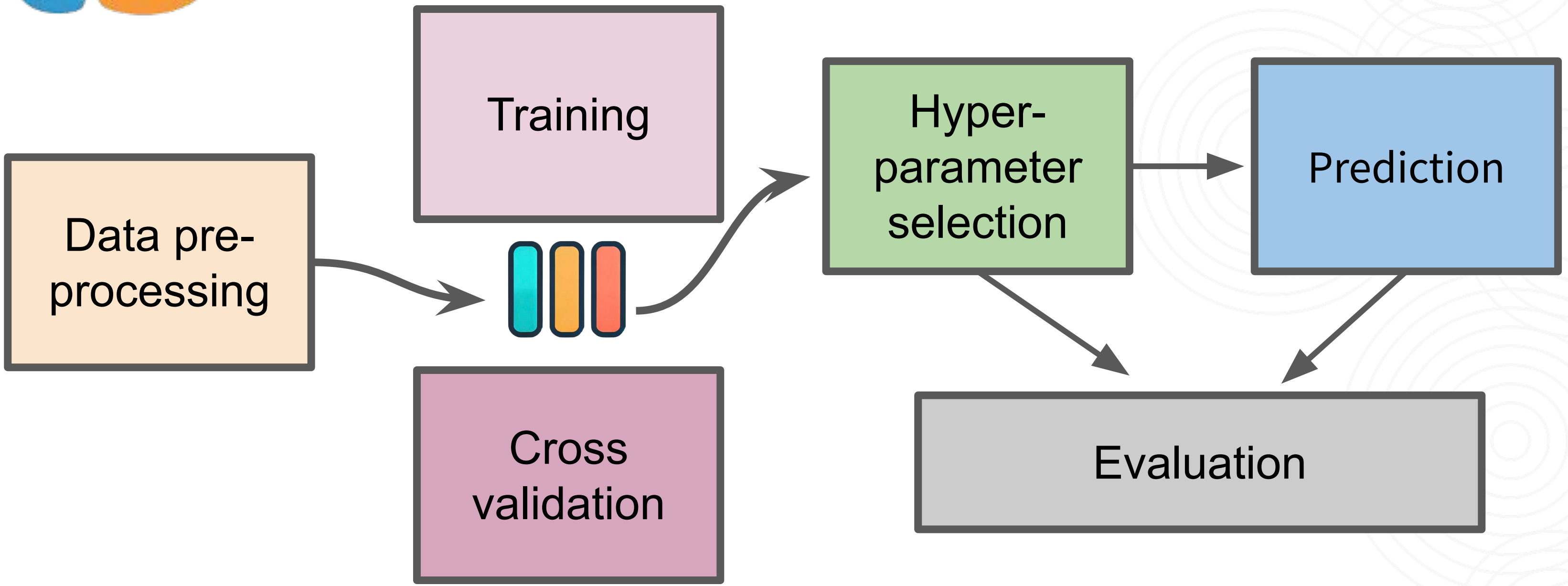
- Effectively handles non-linear relationships in data
- Well-suited for tabular data
- Resistance to overfitting
- Provides feature importance
- Easily parallelizable and scalable

NN

A powerful model that can capture complex and deep patterns through layers of interconnected neurons

- Learns complex and non-linear relationships in data
- Highly flexible and can integrate a variety of data types
- Adapts well to large and diverse datasets

RF Model Pipeline



RF Model Components

Cumulative years

Randomized Search CV

Time series Split

Regression

RF Model Components

Cumulative years	Randomized Search CV	Time series Split	Regression
Expanding training windows	Resource efficient	Temporal order preserved	Continuous numerical outputs

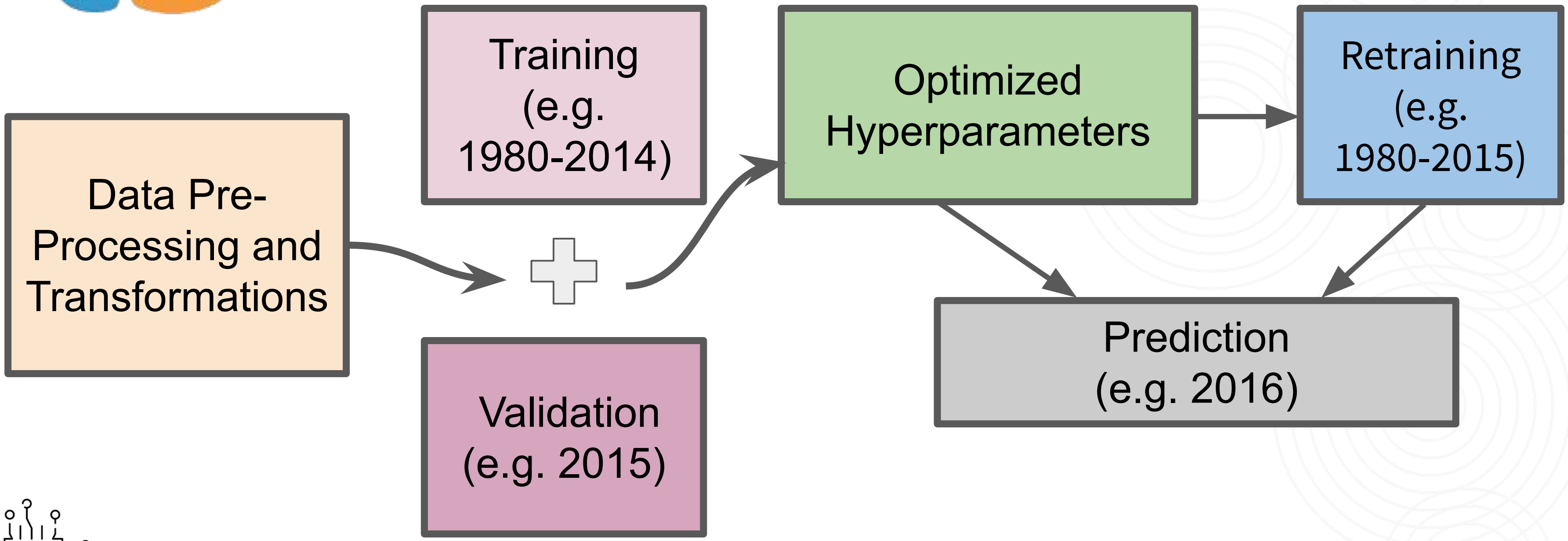
RF Hyperparameter Tuning

```
param_distributions = {
    'max_depth': list(range(10, 200)),          # Max depth of the trees
    'max_features': ['log2', 'sqrt'],           # Number of features to consider for best split
    'min_samples_leaf': [3, 5],                 # Min number of samples required to be a leaf node
    'min_samples_split': [3, 5],                # Min number of samples required to split an internal node
    'n_estimators': list(range(100, 1000)),     # Number of trees in the forest

}

# Create the RandomizedSearchCV instance
random_search = RandomizedSearchCV(estimator=rf,
                                   param_distributions=param_distributions,
                                   n_iter=10,  # Number of parameter settings to try
                                   scoring='neg_mean_squared_error',
                                   cv=tscv,
                                   error_score='raise',
                                   random_state=42)  # Ensure reproducibility
```

Neural Network Training Pipeline



Dynamic Neural Network Architecture

- Dynamic **batch size**
- Dynamic **learning rate** with optimized weight decay
- Dynamic **number of epochs** with early stopping mechanism to save computation time
- Dynamic **number of hidden layers**
- Dynamic **hidden layer dimensions**
- Dynamic **embedding dimensions**
- Regularization with optimized **dropout rate** to prevent overfitting
- **Batch normalizations** and **activations** used in all hidden layers
- kaiming_normal_ **weight initialization**

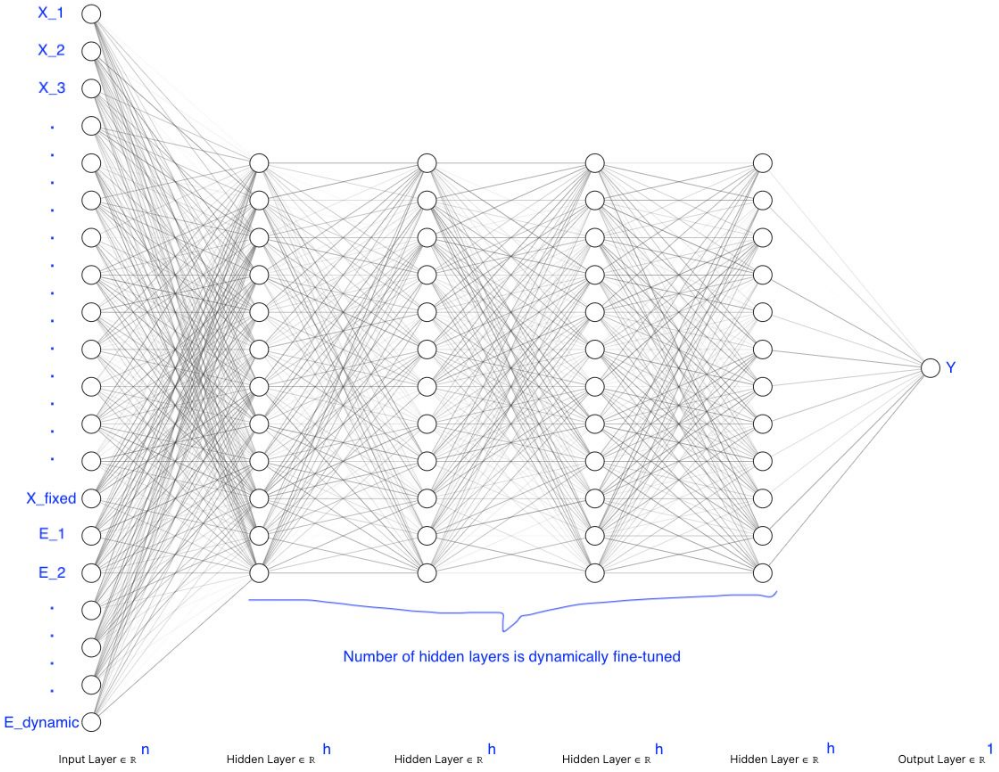
Hyperparameter Tuning - Neural Network

```
config = {
    "continuous_dim": continuous_dim,
    "hidden_dim": tune.choice([i for i in range(5, 200, 10)]),
    "num_layers": tune.choice([1, 2, 3, 4, 5]),
    "num_embeddings": num_embeddings,
    "embedding_dim": tune.choice([i for i in range(1, 11, 1)]),
    "dropout_rate": tune.choice([round(i * 0.01, 2) for i in range(1, 56)]), # uniform (0.01, 0.55)
    "lr": tune.loguniform(1e-6, 1e-2),
    "weight_decay": tune.loguniform(1e-6, 1e-3),
    "num_epochs": max_num_epochs,
    "num_gpus": num_gpus,
    "batch_size": tune.choice([8, 16, 32, 64, 128, 256]),
}

scheduler = ASHAScheduler(
    metric="avg_test_loss",
    mode="min",
    max_t=max_num_epochs,
    grace_period=1,
    reduction_factor=2
)

reporter = CLIReporter(
    metric_columns=["average_train_loss", "avg_test_loss", "training_iteration"])
```

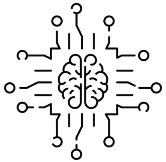
Dynamic Neural Network Architecture



Evaluations

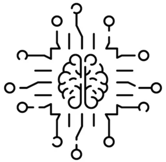
Evaluation Metrics

- **H-L (High-Low) metric**: A value-weighted return differential between the top and bottom portfolios
- **RMSE (Root Mean Square Error)**: A measure of the differences between predicted and actual values in a model
- **L1Loss:** Creates the criterion that measures the mean absolute error (MAE) between each element in the input x and target y (NN only)
- **Feature importance**: A ranking of input features (variables) based on their contribution to a model's predictions (RF only)



Evaluation Results

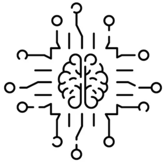
- **H-L (High-Low) metric**: Significant increases in both quarterly and monthly returns
- **RMSE (Root Mean Square Error)**: Some years with high values roughly coinciding with major economic events (e.g., dot-com bubble)
- **Feature importance**: Macroeconomic and investor sentiment variables having higher impacts on stock returns (RF only)



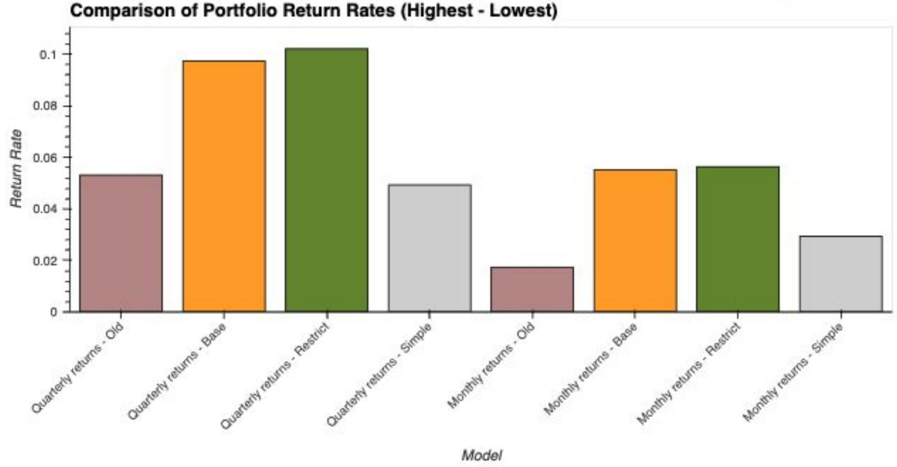
Evaluation Metrics

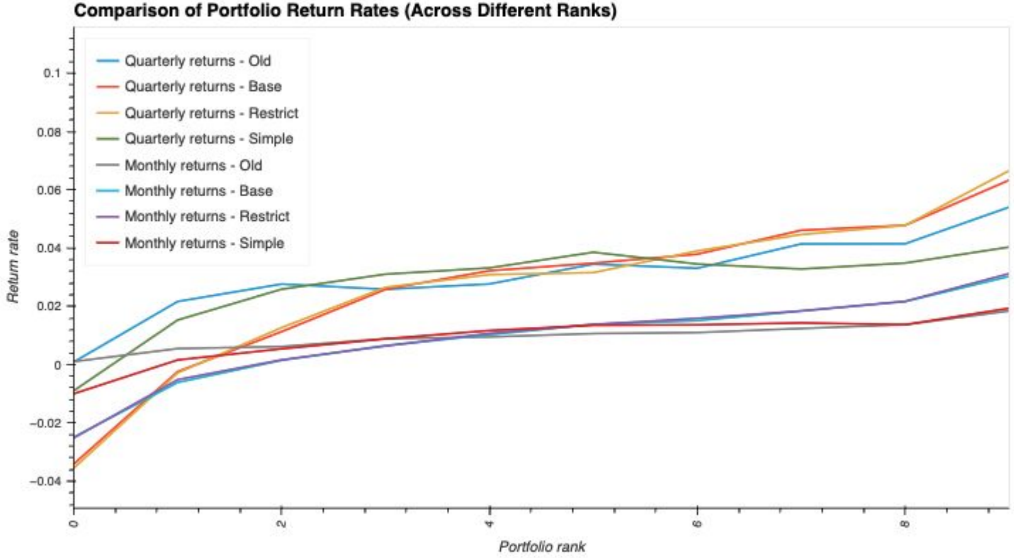
	rank	Average of port_ret
0	0.0	-0.035549
1	1.0	-0.002851
2	2.0	0.012681
3	3.0	0.026528
4	4.0	0.030880
5	5.0	0.031569
6	6.0	0.039013
7	7.0	0.044643
8	8.0	0.047770
9	9.0	0.066568
Return rate		0.102117

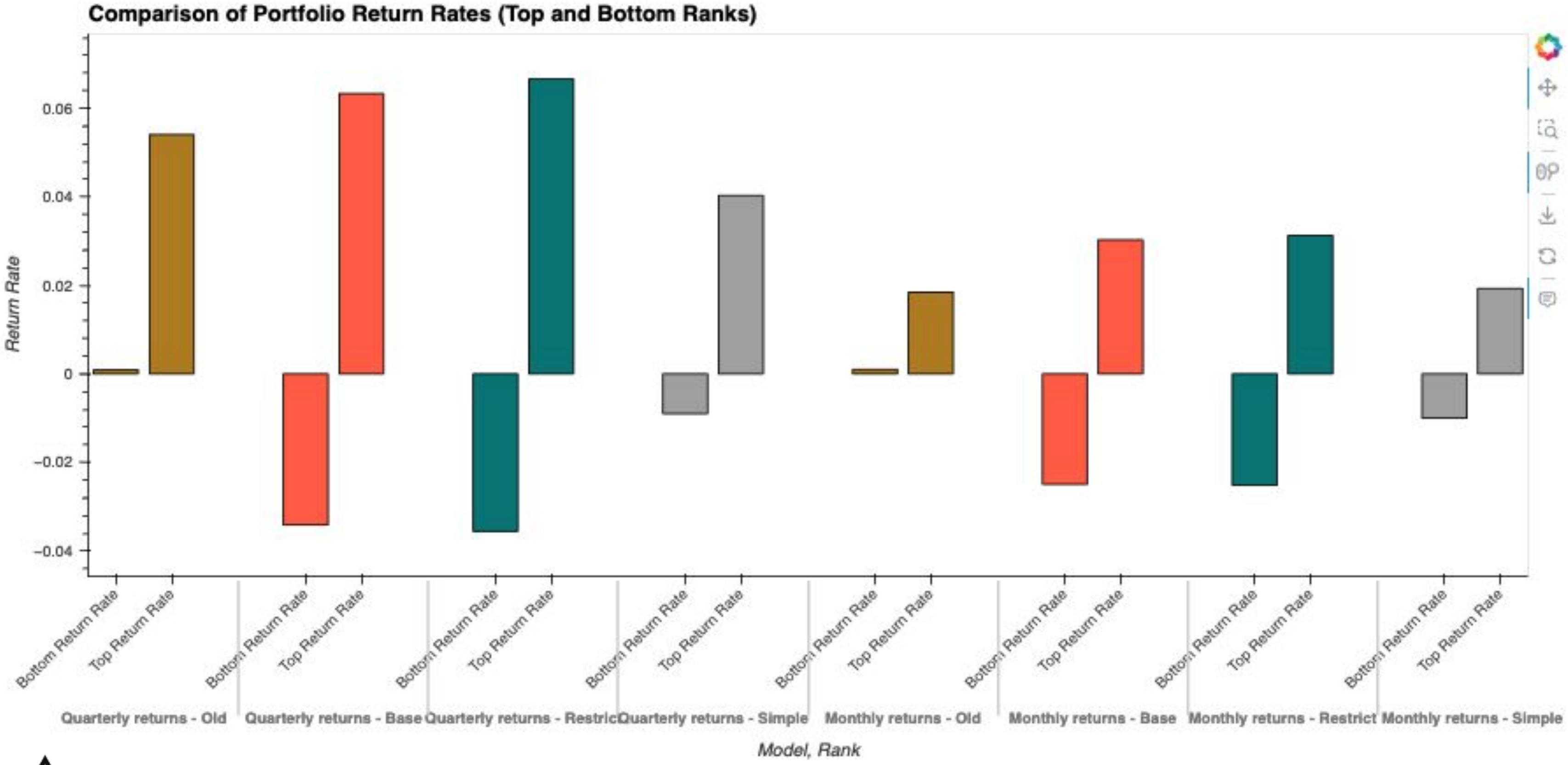
A comparison of **quarterly H-L metric** using quarterly restricted dataset between RF and NN



	rank	Average of port_ret
0	0.0	0.013289
1	1.0	0.025109
2	2.0	0.029242
3	3.0	0.031388
4	4.0	0.033673
5	5.0	0.030429
6	6.0	0.034697
7	7.0	0.034552
8	8.0	0.046154
9	9.0	0.054191
Return rate		0.040902


Comparison of Portfolio Return Rates (Highest - Lowest)
Return Rate
0.1
0.08
0.06
0.04
0.02
0
Quarterly returns - Old
Quarterly returns - Base
Quarterly returns - Restrict
Quarterly returns - Simple
Monthly returns - Old
Monthly returns - Base
Monthly returns - Restrict
Monthly returns - Simple
Model


Comparison of Portfolio Return Rates (Across Different Ranks)
Quarterly returns - Old
Quarterly returns - Base
Quarterly returns - Restrict
Quarterly returns - Simple
Monthly returns - Old
Monthly returns - Base
Monthly returns - Restrict
Monthly returns - Simple
Return rate
0.1
0.08
0.06
0.04
0.02
0
−0.02
−0.04
0
2
4
6
8
Portfolio rank


Comparison of Portfolio Return Rates (Top and Bottom Ranks)
Return Rate
0.06
0.04
0.02
0
−0.02
−0.04
Bottom Return Rate
Top Return Rate
Quarterly returns - Old
Quarterly returns - Base
Quarterly returns - Restric
Quarterly returns - Simple
Monthly returns - Old
Monthly returns - Base
Monthly returns - Restrict
Monthly returns - Simple
Model, Rank

RMSE Scores for Training and Test Datasets - New Data, Restrict, Monthly

Are the results too good to be true?

- Our RF model outperforms mutual funds by delivering approximately 10% higher annual return.
- We met with Suzie and Miao to ensure that there is no **look-ahead-bias** In the input data and that our results are robust and valid

DARC

- Doubled H-L rate for quarterly realized returns and tripled it for monthly realized returns using the updated data
- Higher monthly H-L rate for RF compared to the best NN rate in the benchmark Gu et al. paper

Faculty

- PIT data used in our analysis vs stale, outdated data used in the benchmark paper to be more timely and price relevant
- Higher increases in monthly returns as a result

Scalability

Scalability

Processing Time in Hours by Model Type

	RF (48 CPU cores in parallel on interactive Yens, 10 iterations)		**NN** (24-26 CPU cores in parallel using Long Partition on Yens)	
	Base dataset	**Restricted dataset**	**Base dataset** (150 iterations)	**Restricted dataset** (100 iterations)
Quarterly	26	20	65	55
Monthly	73	40	211	153

Scalability - Random Forest

Library used- Sklearn

Monthly/Quarterly Mean Imputation

Winsorize and Scale

Training and Cross Validation

Fitting

sequential

Parallelize over X cores

Column Parallelization

Model Parallelization

Scalability - Random Forest

Scikit-learn lacks native GPU support

- Switching libraries (e.g., CuML) would increase development overhead

Pipeline scaling limitations

- Full pipeline requires both CPU and GPU resources; GPU use would require significant code changes, overhead of loading and unloading data onto the GPU.

Limited performance gains

- Random Forest’s decision tree structure isn’t as GPU-friendly as deep learning models

Scalability - Neural Network

Neural Network training pipeline can run on multiple CPUs and GPUs

- Switching between CPUs and GPUs only requires changing a few parameters on the Slurm script such as ‘partition’ **#SBATCH -p** and number of GPUs **#SBATCH -G**
- Using **24 CPUs**, the quarterly data took about **2 days and 17 hours** to complete, which included **150 hyperparameter searches** per prediction year

DataParallel processing is enabled on multiple GPUs during training

```
# Wrap the model with DataParallel
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(model)
model.to(device)
```

Scalability - Neural Network

RayTune offers cutting edge hyperparameter optimization algorithms

- Reducing the cost of tuning by **terminating bad runs early** with state of the art ASHA scheduler
- Choosing **better parameters to evaluate**
- **Parallel hyperparameter tuning** across multiple CPUs or GPUs

Object-oriented programming was used to develop robust and maintainable code

- **Modularity and Reusability**: Code was organized in **data preprocessing, data transformation, modeling,** and **prediction post-processing** to reduce redundancy
- **Easier Maintenance**: Encapsulation simplifies debugging and updates without affecting other parts

Git version control was used to track the history of code changes

- **Change Tracking**: Easily revert to previous versions and track changes over time

Scalability - Neural Network

Custom detailed logging was used throughout the entire pipeline

- **Debugging:** Provides clear insights into errors and issues at each step of the process
- **Performance Monitoring:** Helps track execution times and system behavior for optimization

Custom early stopping was implemented to save training time

```
# Early stopping mechanism
if avg_test_loss < best_loss:
    best_loss = avg_test_loss
    epochs_without_improvement = 0
else:
    epochs_without_improvement += 1

if epochs_without_improvement >= patience:
    if not ray_tuning: # only display this message when not using Ray Tune
        logger.info(f"Early stopping at epoch {epoch + 1}")
    break
```

Challenges

Challenges

- **Changing scope of work**:
  - Shift from exploratory proof-of-concept to full model development
  - Transition from research computing support to comprehensive model refinement and implementation
  - Expansion from RF only to include NN
- **Lack of collaboration** in model development and implementation
- **Learning a new domain** — stock price prediction
- **Inability to use GPU efficiently** due to scikit-learn pipeline transformation not natively supporting GPU acceleration

Challenges

- The **transformation step in the Sklearn pipeline took hours** for each prediction year for both the RF and NN models
- **Training the models**, both RF and NN, **took days** to complete
- Building a flexible neural network training pipeline from scratch is not trivial, and there were **several difficult bugs to resolve**
- A fixed-number prediction issue occurred in the neural network for some of the more **challenging years to predict** (e.g., 2000, 2009)
- Doing **cross validation in the NN is inefficient**

Discussions