International Institute of Information Technology, Bangalore

VLSI Architecture Design
VLS 731

Fair and Accurate Banking Predictions: A Multi-Objective Optimization Approach

Name	Roll No	Email
Niranjan Gopal	IMT2022543	niranjan.gopal@iiitb.ac.in
Akash Sridhar	IMT2022501	akash.sridhar@iiitb.ac.in

Summary

This report presents a comprehensive analysis of multi-objective optimization techniques applied to banking predictions, balancing the critical trade-off between **model accuracy** and **fairness metrics**. We leverage state-of-the-art optimization frameworks - **BoTorch** and **Optuna** - to generate Pareto-optimal solutions across three machine learning models: **Logistic Regression**, **Random Forest**, and **Neural Networks**. Our implementation addresses real-world challenges including GPU/CPU conflicts, precision requirements, and version compatibility, multi threading - delivering a robust framework for ethical AI deployment in financial services.

1 Introduction and Motivation

1.1 Problem Context

In the banking sector, predictive models for customer deposit behavior must satisfy two competing objectives:

- **High Accuracy**: Maximizing prediction correctness to ensure profitable marketing campaigns

- **Fairness**: Ensuring equitable treatment across demographic groups (measured via **Demographic Parity Difference - DPD**)

1.2 Dataset Overview

We utilize the **UCI Bank Marketing Dataset** from Kaggle, containing:

- **11,162 instances** of customer banking data

- Aim is to have **same number of positive labels** for all 3 classes of single, married, divorced thereby ensuring fairness

- We treat **marital status** as the sensitive feature and **single** as protected group.

- **17 features** including age, job, marital status, education, etc.

- Binary target variable: **Term deposit subscription (yes/no)**

1.3 Why Multi-Objective Optimization?

- **Real-world tradeoffs**: Perfect fairness often reduces accuracy and vice versa

- **Regulatory compliance**: Financial institutions must demonstrate non-discriminatory practices.

- **Strategic flexibility**: Enables selection from a frontier of optimal solutions based on business priorities

2 Algorithmic Framework

2.1 BoTorch Implementation

- **Advantages**:

  - State-of-the-art Bayesian optimization with Gaussian Processes
  - Supports parallel evaluation of candidates

– Advanced acquisition functions (qHVIM, qEI)

- **Disadvantages**:

  – Computationally intensive (15-20 mins for entire search)

  – Requires careful hyperparameter tuning

  – Float64 tensor requirement conflicts with PyTorch NN defaults

2.2 Optuna Implementation

- **Advantages**:

  – Lightweight and efficient (10-15 sec for entire search)

  – Automatic pruning of unpromising trials

  – Built-in visualization tools

- **Disadvantages**:

  – Less sophisticated than BoTorch for complex spaces

  – Limited multi-objective optimization capabilities

2.3 Parameter Spaces

- **Logistic Regression (LR)**

  – **Optuna LR:** Model space is 1D (C a.k.a inverse regularization strength)

  – **BoTorch LR:** Model space is 3D (C, penalty_type, class_weight)

- **Random Forest (RF)**

  – **Optuna RF:** Model space is 3D (min_samples_split, max_depth, n_estimators)

  – **BoTorch RF:** Model space is 4D (min_samples_leaf, min_samples_split, max_depth, n_estimators)

- **Neural Network (NN)**

  – **Optuna NN:** Model space is 3D (layers, units, learning rate)

  – **BoTorch NN:** Model space is 7D (layers, units, learning rate, activation functions, optimizers, batch_size)

3 Methodology and Observations

In this section we outline the exhaustive set of experiments we conducted during the multi-objective optimization of fairness and accuracy using Optuna and BoTorch. Our methodology involved exploring a wide variety of configurations, hyperparameter spaces, optimization settings, and numerical precision choices across three core models: Logistic Regression, Random Forest, and Neural Networks. Each experimental path provided unique insights into how these models behave under fairness constraints and how Bayesian optimization can be effectively tuned to improve Pareto front quality. We divide our analysis into six major avenues of experimentation.

3.1   Parameter Space Exploration

A significant portion of our effort was devoted to carefully defining and tuning the hyperparameter spaces for each model. Rather than exhaustively expanding the dimensionality of the parameter space, we focused on **compact yet expressive spaces** that strongly influenced both **accuracy** and **fairness**. Our choices were informed by empirical trials as well as theoretical intuitions about how different parameters interact with fairness-sensitive objectives.

Logistic Regression (LR):

For LR, the key parameter is the inverse regularization strength, `C`. In the `Optuna` setting, we limited the space to 1D (only `C`) because **regularization directly controls the complexity of the decision boundary**. We observed that varying `C` alone often resulted in markedly different fairness-accuracy trade-offs. While LR is a linear model, tuning `C` effectively allows us to interpolate between underfitting and overfitting regimes, which in turn governs group-wise error disparities. In `BoTorch`, we added `penalty` type (`l1` vs `l2`) and `class_weight` (`None` vs `balanced`), forming a 3D space. The inclusion of these discrete parameters improved the diversity of the Pareto front by letting the optimizer explore models that focus differently on minority class penalties. **L1 regularization combined with class reweighting** often gave **surprising boosts in fairness without large drops in accuracy**. This has been presented in the graph below :-
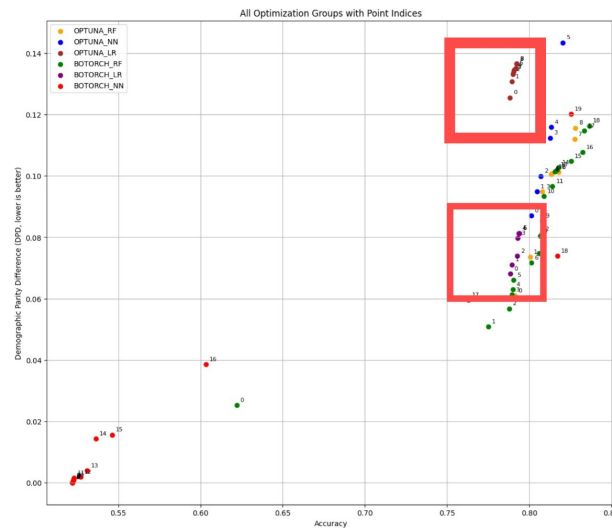
Random Forest (RF):

RF offered more architectural knobs. For `Optuna`, we stuck to a 3D space—`min_samples_split`, `max_depth`, and `n_estimators`—which are among the most critical in controlling the model's flexibility. **Shallow trees** with `high min_samples_split` were generally fairer but underfit the data, while **deeper trees** with `low thresholds` maximized accuracy at the cost of fairness. For `BoTorch`, we introduced `min_samples_leaf` as a fourth dimension. Empirically, this additional axis gave a small improvement in accuracy - it allowed us to isolate trees that were deep but had local regularization via leaf size, sometimes producing models with both high accuracy. This showed that **not all complexity is equally harmful**—**local smoothing** (via `min_samples_leaf`) can mitigate global overfitting effects.

Neural Networks (NN):

NNs naturally support rich parameterizations, and this was the domain where the **design of parameter space had the greatest effect**. `Optuna`'s 3D space (`num_layers`, `hidden_units`, `learning_rate`) already gave us substantial variety in model performance. We observed that shallow networks with low learning rates tended to yield fairer models, likely due to smoother decision boundaries. In `BoTorch`, we expanded to 7D: `activation`, `optimizer`, `batch_size`, and `dropout` were added. The exploration here was extremely insightful—certain configurations (e.g., `LeakyReLU + SGD + small batch size`) consistently led to very fair models, suggesting over-responsiveness to dominant-class

patterns. `Dropout` had a mixed effect: it helped generalization in some cases but sometimes increased variance in group-level errors. The most interesting finding was that while **deeper networks usually achieved higher accuracy**, the **fairer networks almost always had simpler structures**—implying that **fairness constraints inherently act as an implicit regularizer**.

General Observations:
Across all models, one key trend emerged: **increasing the dimensionality of the parameter space beyond a certain point did not always improve the Pareto front**. In fact, bloated spaces often led to unstable or redundant optimization paths. The most impactful parameters were those that directly shaped the decision boundary (like `regularization`, `depth`, or `learning_rate`), while others (like `dropout` or `activation`) had **marginal effects unless in very specific combinations**. Furthermore, fairness-accuracy trade-offs were often controlled by a small subset of parameters, and identifying these early led to **faster convergence and better exploration efficiency**. In essence, a well-constructed, low-to-moderate dimensional space provided better optimization landscapes—reducing noise, improving interpretability, and leading to more diverse solutions along the Pareto frontier.

3.2 BoTorch Configuration and Acquisition Functions

We experimented extensively with various configurations of BoTorch. Specifically, we evaluated different acquisition functions: Expected Improvement (EI), Hypervolume Improvement (HVI), and its batched variant qHVI.

While EI is effective for single-objective optimization, it did not translate well in our bi-objective scenario. HVI and qHVI, designed for multi-objective settings, provided significantly better Pareto front quality. qHVI was especially useful when dealing with batch evaluation of configurations, allowing us to speed up the optimization process.

We also experimented with different initial sampling strategies. Using Sobol sequences, Latin Hypercube Sampling, and random initialization affected early-stage model exploration. Sobol sequences led to better exploration of the input space, particularly in higher dimensions (e.g., neural network hyperparameters), and often yielded better global optima.

Moreover, we observed that the **choice of surrogate model and the fidelity of its inputs critically impacted the stability** of BoTorch's acquisition functions. BoTorch requires inputs to be within the unit cube and in 'torch.float64' precision for stable GP modeling. Failure to min-max scale inputs or use double precision led to frequent issues with matrix non-positive definiteness during covariance estimation.

Our experiments showed that **qLogEHVI coupled with Sobol initialization and double-precision tensors** produced the best results in terms of both convergence speed and Pareto quality.

3.3 Logistic Regression Model Configurations

For Logistic Regression, we evaluated several configurations to understand how regularization, solver type, and feature scaling affected the accuracy-fairness trade-off. We primarily used the '`liblinear`' and '`lbfgs`' solvers and experimented with varying the 'max_iter' parameter (from 100 to 1000).

Early warnings such as **ConvergenceWarning** prompted us to either increase 'max_iter' or scale features using `StandardScaler`. We also observed that a high value of 'C' (low regularization) led to models that were accurate but unfair — often favoring the majority group (in our case this was marital_status as Divorced+Married) in the protected attribute (marital_status as single). Conversely, very small values of 'C' overly regularized the model, flattening decision boundaries and thus decreasing accuracy.

We also explored one-vs-rest (`OvR`) vs `multinomial classification settings`, especially when dealing with multi-class protected groups (since marital status = single, married, divorced). Multinomial settings were found to better model the interactions among classes and led to improved fairness

metrics at a slight cost to accuracy.

Interestingly, the most fair and reasonably accurate logistic model emerged from using moderate regularization ($C \approx 0.1$), with standardized features and the multinomial loss.

3.4   Random Forest Configurations

Our Random Forest experiments included tuning the number of estimators, tree depth, splitting criterion ('gini' vs 'entropy'), and minimum samples per split/leaf. We also compared performance with and without bootstrapping.

We observed that **shallow trees (depth $\leq$ 5)** tended to **favor fairness**, as they produced simpler, more generalizable decision boundaries. However, this came at the cost of reduced accuracy. Conversely, **deeper forests (depth $\geq$ 15)** often **led to overfitting and severe fairness violations**, especially when the protected attribute was correlated with the target label.

The number of estimators also played a role in the variance of predictions. We noticed that increasing trees from 100 to 500 marginally improved accuracy, it did not significantly impact fairness metrics. Thus, we settled on around 150 estimators as a good trade-off point.

Interestingly, using **entropy** as the splitting criterion rather than **gini** slightly improved fairness by favoring more balanced splits, especially when the data distribution was skewed across protected groups.
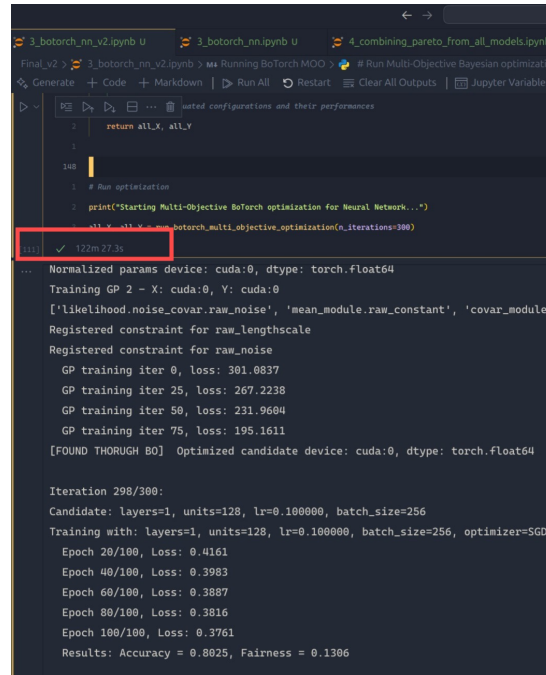
3.5   Neural Network Architectures and Training Dynamics

Neural Networks introduced the highest degree of flexibility — and complexity — into our optimization pipeline. We experimented with 1 to 4 hidden layers, layer widths from 16 to 256 neurons, and learning rates, activation functions (ReLU, LeakyReLU, Tanh), and dropout layers (rates between 0 and 0.5).

We tested different optimizers ('SGD', 'Adam', and 'RMSprop') and found that 'SGD' was consistently more stable and converged faster across optimization trials. We also found learning rates that were below 10^{-3} led to stagnation, while those above 10^{-1} often resulted in unstable training.

What we did not notice before was :- **Batch size and epoch tuning also impacted model quality**. Smaller batch sizes (32 or 64) improved generalization and fairness, while large batches (256+) led to sharp loss valleys and high disparity. We found this when we ran the optimizer with only batch size as the model space (1D). Where we did not expect to find a non-dominating point, surprisingly, we did - just like the case of Logistic Regression producing the good pareto front with just 1 model parameter (C).

We came up with a optimal GPU - CPU workflow (since we wanted NN to be trained on GPU) and for metrics calculations, we had to move the required tensors back to CPU and calculate the metrics there. We also had the acquisition function converge on next candidate point in CPU. So essentially GP convergence loop in CPU, NN learning loop in GPU. " Optimization took 2 GPU hours to explore the 7D model search space.

3.6 Numerical Stability, Precision, and Optimization Heuristics

During our experimentation with BoTorch, we encountered several numerical stability issues that significantly influenced the optimization trajectory. One of the most persistent issues arose from the numerical precision used in modeling. Initially, we employed `torch.float32` (single precision) for the GP models. However, multiple warnings and degradations in performance highlighted the inadequacy of single precision. After switching to `torch.float64` (double precision), we observed a marked improvement in the quality of the Pareto fronts. This is consistent with BoTorch documentation, which strongly recommends double precision for enhanced numerical stability and reduced susceptibility to ill-conditioned covariance matrices. **Switching to 'torch.float64' — as recommended by BoTorch — significantly improved stability**. This was especially true for high-dimensional spaces like those from neural network hyperparameters. Additionally, scaling all input features to the unit cube became essential, as BoTorch's internal assumptions require bounded inputs.

In addition, BoTorch raised the following warning during our use of the `qEHVI` acquisition function:

```
NumericsWarning: qExpectedHypervolumeImprovement has known numerical issues that
lead to suboptimal optimization performance. It is strongly recommended to simply
replace qExpectedHypervolumeImprovement --> qLogExpectedHypervolumeImprovement instead,
which fixes the issues and has the same API. . See https://arxiv.org/abs/2310.20708
for details.
```

This was a interesting insight. After replacing `qEHVI` with `qLogExpectedHypervolumeImprovement`, we observed smoother acquisition surfaces and fewer cases of ill-conditioned GP posteriors. This change is backed by the recent NeurIPS 2023 spotlight paper that the BoTorch linked in the output:

- Sebastian Ament et al., *Unexpected Improvements to Expected Improvement for Bayesian Optimization*, NeurIPS 2023. Available at: https://arxiv.org/abs/2310.20708

Their analysis revealed that applying the logarithmic transformation to the Expected Hypervolume Improvement (EHVI) criterion mitigates over-exploration of poorly estimated hypervolumes and THIS is what improves both numerical stability and optimization convergence. This switch proved especially effective when working with neural network-based surrogates or high-dimensional objective spaces.

We also noticed that there were instances where optimizer tried adding small jitter values (e.g., 10^{-9}, 10^{-3}) to the diagonal of kernel matrices to **stabilize Cholesky decomposition**. However,

```
Iteration 60/90:
Candidate: layers=2, units=59, lr=0.003347, batch_size=40
Candidate device before evaluation: cuda:0
/home/nira/Documents/code/aiml/MultiObjective_Machine_Learning/.conda/lib/python3.11/site-packages/linear_operator/utils/cholesky.py:40: NumericalWarning: A not p.d., added jitter of 1.0e-08 to the di
    warnings.warn(
/home/nira/Documents/code/aiml/MultiObjective_Machine_Learning/.conda/lib/python3.11/site-packages/linear_operator/utils/cholesky.py:40: NumericalWarning: A not p.d., added jitter of 1.0e-07 to the di
    warnings.warn(
/home/nira/Documents/code/aiml/MultiObjective_Machine_Learning/.conda/lib/python3.11/site-packages/linear_operator/utils/cholesky.py:40: NumericalWarning: A not p.d., added jitter of 1.0e-06 to the di
    warnings.warn(
/home/nira/Documents/code/aiml/MultiObjective_Machine_Learning/.conda/lib/python3.11/site-packages/linear_operator/utils/cholesky.py:40: NumericalWarning: A not p.d., added jitter of 1.0e-05 to the di
    warnings.warn(
/home/nira/Documents/code/aiml/MultiObjective_Machine_Learning/.conda/lib/python3.11/site-packages/linear_operator/utils/cholesky.py:40: NumericalWarning: A not p.d., added jitter of 1.0e-04 to the di
    warnings.warn(
/home/nira/Documents/code/aiml/MultiObjective_Machine_Learning/.conda/lib/python3.11/site-packages/linear_operator/utils/cholesky.py:40: NumericalWarning: A not p.d., added jitter of 1.0e-03 to the di
    warnings.warn(
 Epoch 20/100, Loss: 0.3270
 Epoch 40/100, Loss: 0.2510
 Epoch 60/100, Loss: 0.2010
 Epoch 80/100, Loss: 0.1588
 Epoch 100/100, Loss: 0.1185
 Results: Accuracy = 0.7882, Fairness = 0.1161
Return tensor device: cuda:0, dtype: torch.float64
Results after evaluation: cuda:0, dtype: torch.float64
Before concatenation — X_train: cuda:0, new_X: cuda:0
Before concatenation — Y_train: cuda:0, new_Y: cuda:0
After concatenation — X_train: cuda:0, Y_train: cuda:0
```

Figure 1: NumericalWarning: A not p.d., added jitter of 1.0e-08 to the diagonal. Optimizer kept on adding upto 1.0e-03 jitter for matrix to finally be p.d

in some cases, even increased jitter didn't resolve positive definiteness issues, especially when the acquisition function tried to sample near singularities. What we found out from online sources was that **failures often corresponded to highly clustered initial points or repeated evaluations due to poor Sobol spacing**.

To mitigate this, we tried implementing strategies like:

- Increasing the number of initial design points (e.g., 70 instead of 20)

- Disabling duplicate evaluations in early Sobol rounds

- Regularizing the output targets with standard normalization

We expected to have improvements like surrogate surface to be smooth and valid for acquisition functions to explore, but it didn't happen so.

Throughout the project, we did try to solve all the warnings and suggestions that interpreter threw at us, but we ultimately had to suppress 3 warnings like these where there was little we could do - $InputDataWarning$ (since we were adamant on having a GPU/CPU workflow for the best optimization possible and the best possible workflow) , $DataConversionWarning$ (instances were we could not afford to keep max_iter to be as high as 2000).

In summary, leveraging high-precision types, implementing jitter when necessary, and choosing more stable acquisition functions (like `qLogExpectedHypervolumeImprovement`) played a crucial role in resolving numerical issues and improving the robustness and consistency of our optimization pipeline.

Overall Summary

Across all configurations, we found that **no single model universally dominated across all fairness-accuracy trade-offs**. Logistic Regression offered interpretability and predictable fairness trends, Random Forests captured non-linear patterns at the risk of group bias, and Neural Networks offered high expressiveness but posed challenges in fairness control.

BoTorch, when configured with the right precision, input scaling, and acquisition strategies, proved to be a powerful tool in uncovering Pareto-optimal trade-offs. Our experiments highlight that optimizing for fairness is not merely a post-processing task, but something deeply embedded in model architecture, optimization pipeline, and surrogate modeling strategy.

4 Numerical Results

4.1 Combined Pareto Frontier
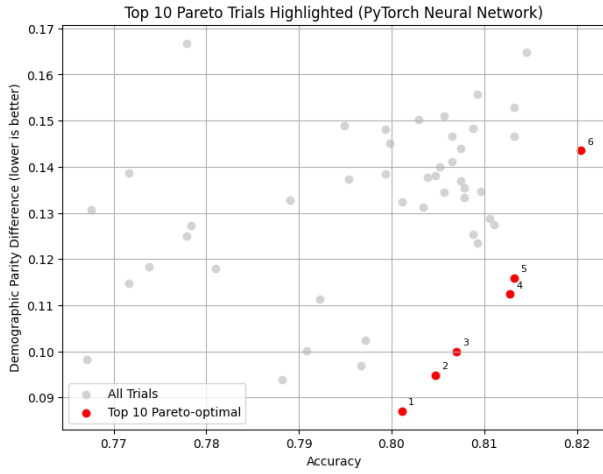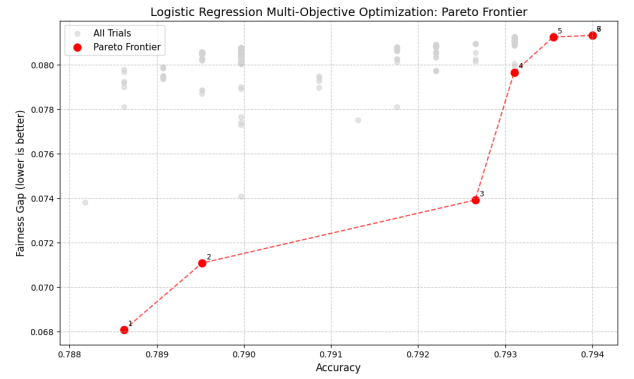

(a) Optuna - Neural Network
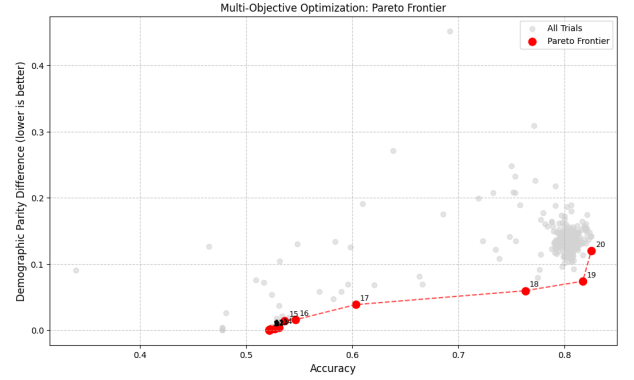
(b) BoTorch - Neural Network
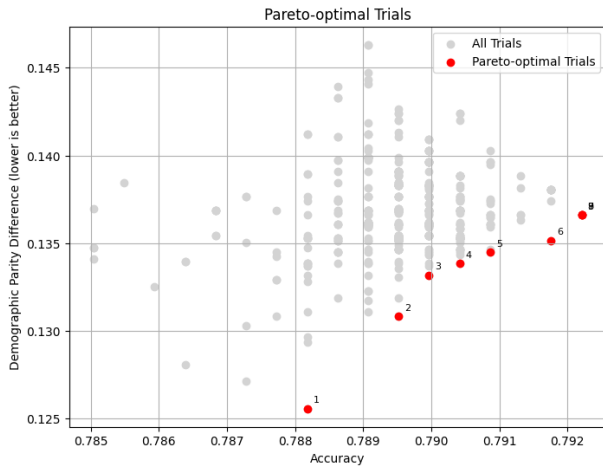
(c) Optuna - Logistic Regression

(d) BoTorch - Logistic Regression
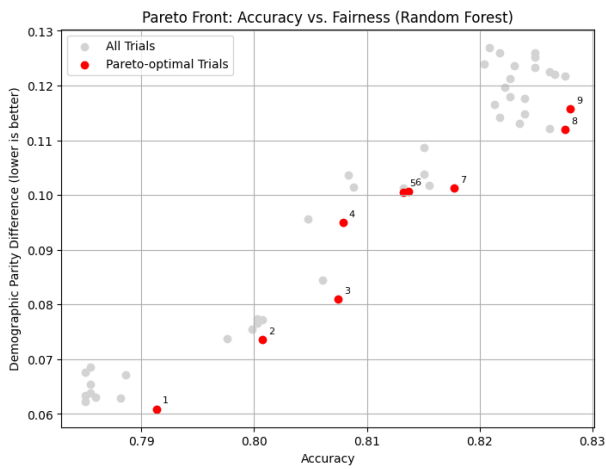
(e) Optuna - Random Forest
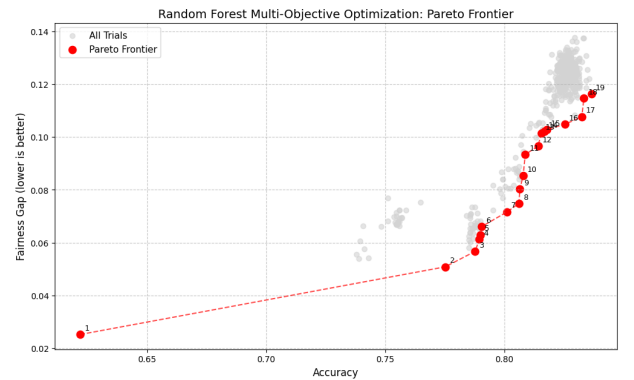
(f) BoTorch - Random Forest

Figure 2: Comparison of Pareto frontiers obtained via Optuna and BoTorch optimization for each model type

4.2 Combined Pareto Frontier
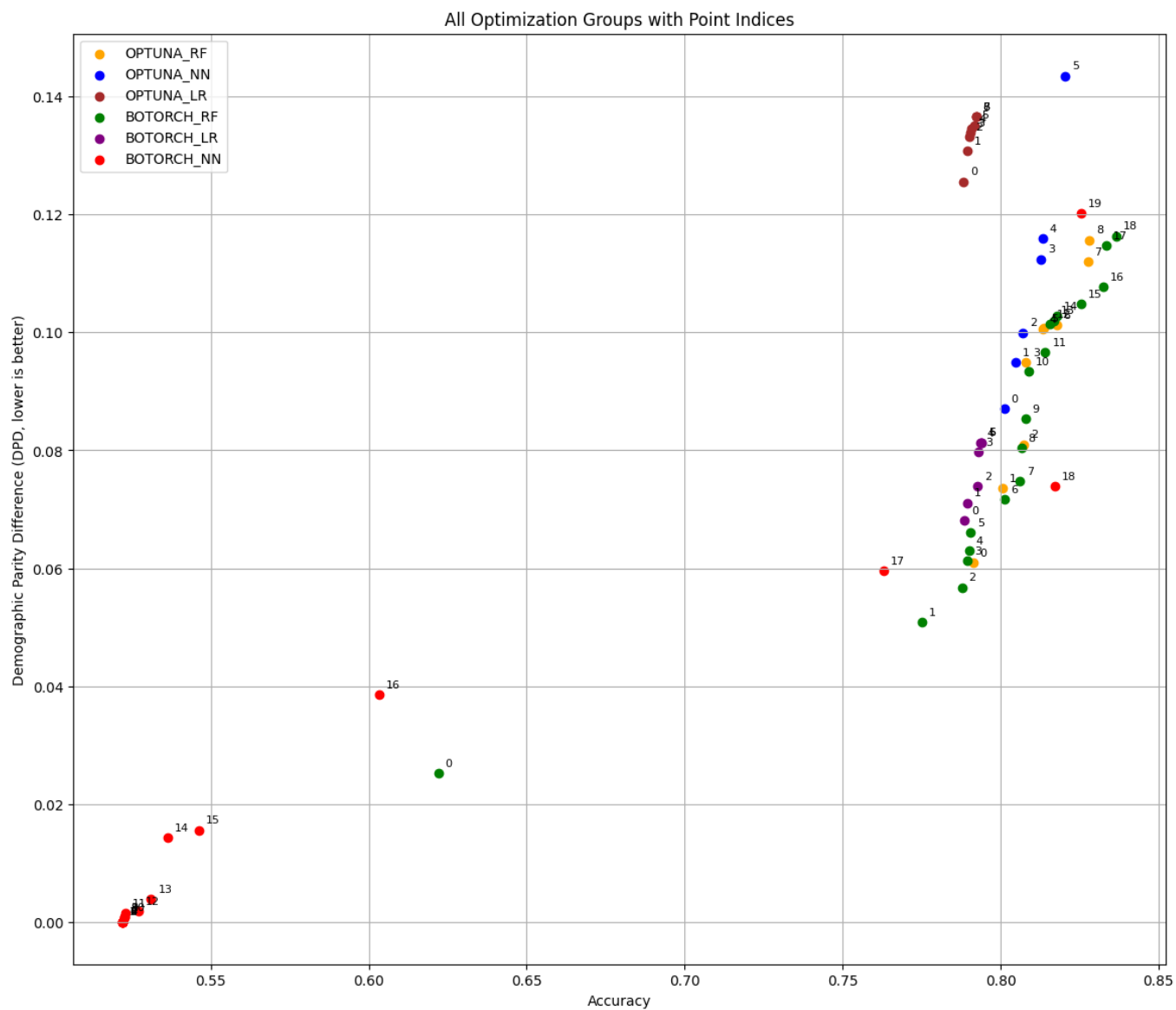
Figure 3: Composite Pareto frontier showing all optimized models
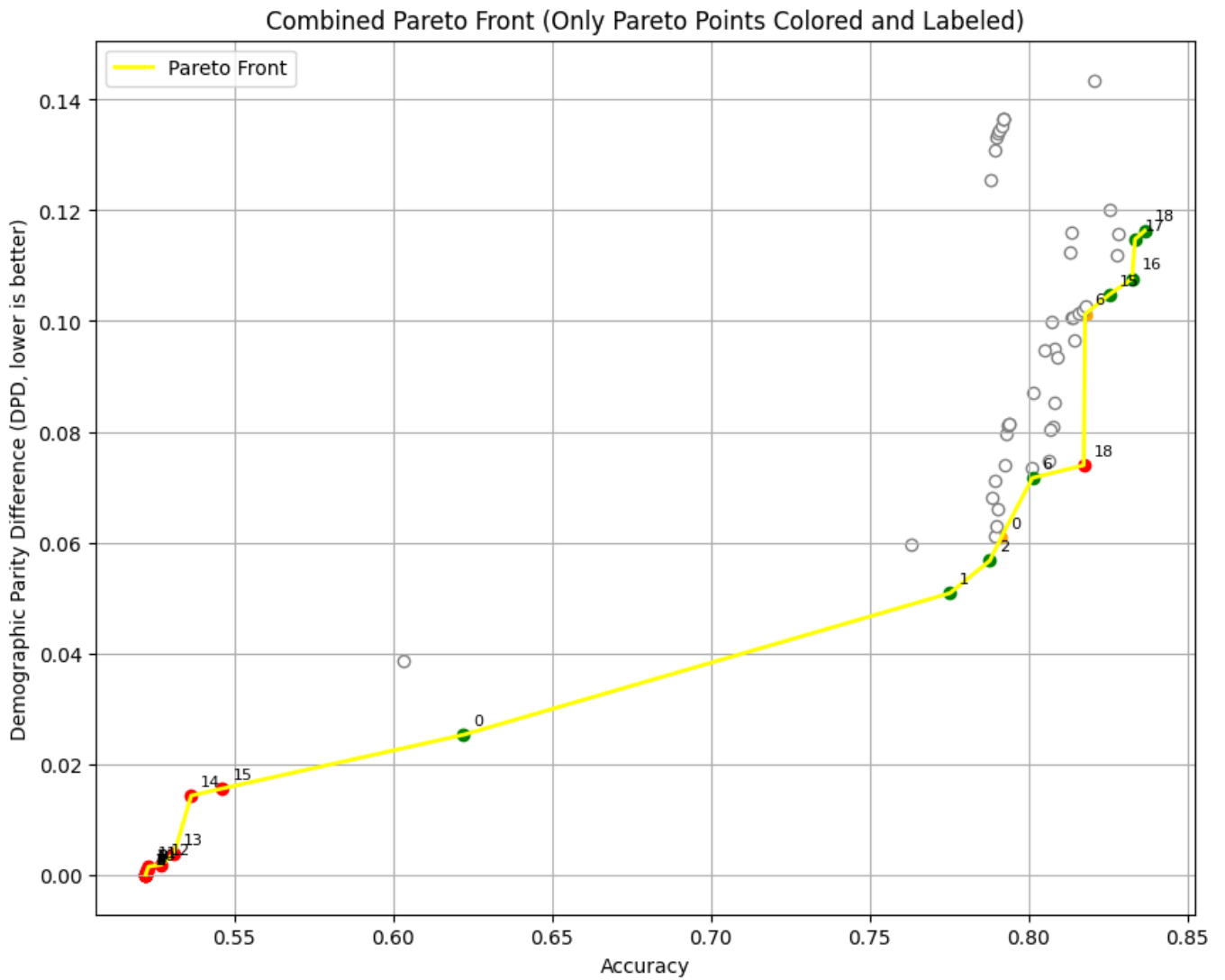
Figure 4: Final Pareto Frontier

4.2.1 Optuna - Logistic Regression

Index	Accuracy	DPD	Model Parameters
1	0.78817733	0.125571326	{'C':1.7008021695866808}
2	0.78952082	0.130821847	{'C': 0.6889367620281636}
3	0.78996865	0.133182192	{'C':9.362048338011848}
4	0.79041648	0.133840497	{'C':0.2774540519206283}
5	0.7908643	0.134498801	{'C':8.34431753095483}
6	0.79175996	0.135157107	{'C':0.06381656061873184}
7	0.79220779	0.136602193	{'C':0.059729469545183066}
8	0.79220779	0.136602193	{'C':0.058856928331692905}
9	0.79220779	0.136602193	{'C':0.06225026047297803}

4.2.2 Optuna - Random Forest

Index	Accuracy	DPD	Model Parameters
1	0.791312	0.06089	{'n_estimators': 82, 'max_depth': 4, 'min_samples_split': 3}
2	0.800716	0.07351	{'n_estimators': 163, 'max_depth': 5, 'min_samples_split': 9}
3	0.807433	0.08099	{'n_estimators': 179, 'max_depth': 6, 'min_samples_split': 9}
4	0.807881	0.09493	{'n_estimators': 96, 'max_depth': 6, 'min_samples_split': 5}
5	0.813255	0.10057	{'n_estimators': 152, 'max_depth': 7, 'min_samples_split': 7}
6	0.813703	0.10069	{'n_estimators': 86, 'max_depth': 7, 'min_samples_split': 6}
7	0.817733	0.10122	{'n_estimators': 138, 'max_depth': 7, 'min_samples_split': 4}
8	0.827586	0.112	{'n_estimators': 82, 'max_depth': 8, 'min_samples_split': 4}
9	0.828034	0.11567	{'n_estimators': 98, 'max_depth': 8, 'min_samples_split': 4}

4.2.3 Optuna - Nueral Network

Index	Accuracy	DPD	Model Parameters
1	0.801164,	0.08706,	{'layers':1,'units':53,'learning_rate':0.0998842936532}
2	0.804746,	0.09485,	{'layers':2,'units':85,'learning_rate':0.000469962442648}
3	0.806986,	0.09997,	{'layers':2,'units':96,'learning_rate':0.0186193334077}
4	0.812807,	.112435,	{'layers':4,'units':32,'learning_rate':0.000533930135863}
5	0.813255,	0.11593,	{'layers':1,'units':59,'learning_rate':0.0613403735347}
6	0.820420,	0.14350,	{'layers':2,'units':74,'learning_rate':0.0803358147025}

4.2.4 Botorch - Logistic Regression

Index	Accuracy	DPD	Model Parameters
1	0.7883	0.06807895	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}
2	0.789	0.07108396	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}
3	0.7925	0.07391786	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}
4	0.7936	0.07966131	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}
5	0.7937	0.08125039	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}
6	0.7932	0.08132192	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}
7	0.7932	0.08132192	{'C': 8.427191734313965 'penalty': 'l2' 'class_weight': 'balanced'}

4.2.5 Botorch - Random Forest

Index	Accuracy	DPD	Model Parameters
1	0.622035185	0.025240848	{'n_estimators': 21, 'max_depth': 1, 'min_samples_split': 14.032552719116211}
2	0.775194649	0.050843605	{'n_estimators': 22, 'max_depth': 3, 'min_samples_split': 5.235397458076477}
3	0.787724429	0.05670402	{'n_estimators': 27, 'max_depth': 4, 'min_samples_split': 9.547285556793213}
4	0.789525826	0.061296235	{'n_estimators': 136, 'max_depth': 4, 'min_samples_split': 16.88043963909149}
5	0.789966176	0.062998276	{'n_estimators': 87, 'max_depth': 4, 'min_samples_split': 3.92435085773468}
6	0.790416524	0.066016927	{'n_estimators': 185, 'max_depth': 4, 'min_samples_split': 11.718171119689941}
7	0.801164909	0.071668797	{'n_estimators': 207, 'max_depth': 5, 'min_samples_split': 8.665801167488098}
8	0.806097231	0.074800007	{'n_estimators': 21, 'max_depth': 6, 'min_samples_split': 3.7175471782684326}
9	0.8065391	0.080339317	{'n_estimators': 213, 'max_depth': 6, 'min_samples_split': 9.048012733459473}
10	0.807880148	0.085332882	{'n_estimators': 251, 'max_depth': 6, 'min_samples_split': 15.708385109901428}
11	0.808770846	0.093345097	{'n_estimators': 180, 'max_depth': 6, 'min_samples_split': 9.752291083335876}
12	0.814155039	0.096604785	{'n_estimators': 24, 'max_depth': 7, 'min_samples_split': 10.109917521476746}
13	0.815496086	0.101357313	{'n_estimators': 190, 'max_depth': 7, 'min_samples_split': 2.1884552240371704}
14	0.816837134	0.102015617	{'n_estimators': 176, 'max_depth': 7, 'min_samples_split': 9.972015500068665}
15	0.817737833	0.102802399	{'n_estimators': 142, 'max_depth': 8, 'min_samples_split': 19.63646948337555}
16	0.825343775	0.104777313	{'n_estimators': 22, 'max_depth': 19, 'min_samples_split': 18.678343653678894}
17	0.832519822	0.107667483	{'n_estimators': 17, 'max_depth': 22, 'min_samples_split': 8.001313924789429}
18	0.833400058	0.114764443	{'n_estimators': 86, 'max_depth': 25, 'min_samples_split': 7.866268992424011}
19	0.836542503	0.116338007	{'n_estimators': 39, 'max_depth': 29, 'min_samples_split': 5.505961537361145}

4.2.6 Botorch - Nueral Network

Index	Accuracy	DPD	Model Parameters
1	0.522167488	0	{'layers': 3 'units': 72 'learning_rate': 0.09575703180698278 'batch_size': 156 'optimizer': 'Adam' 'activation': 'LeakyReLU' 'dropout': 1}
2	0.522167488	0	{'layers': 4 'units': 32 'learning_rate': 3.0847611469672955e05 'batch_size' 16 'optimizer' 'SGD' 'activation': 'ReLU' 'dropout': 0}
3	0.522167488	0	{'layers': 4 'units': 32 'learning_rate': 0.00025475806243148517 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
4	0.522167488	0	{'layers': 4 'units': 32 'learning_rate': 0.0005642859851861557 'batch_size': 16 'optimizer': 'SGD' 'activation': 'LeakyReLU' 'dropout': 1}
5	0.522167488	0	{'layers': 4 'units': 128 'learning_rate': 0.1 'batch_size': 256 'optimizer': 'RMSprop' 'activation': 'ReLU' 'dropout': 0}
6	0.522167488	0	{'layers': 3 'units': 32 'learning_rate': 0.00010770610593330191 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
7	0.522167488	0	{'layers': 4 'units': 32 'learning_rate': 0.0003674564975397127 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
8	0.522167488	0	{'layers': 3 'units': 32 'learning_rate': 8.777040908429805e05 'batch_size' 16 'optimizer' 'SGD' 'activation': 'ReLU' 'dropout': 0}
9	0.522615316	0.000786782	{'layers': 4 'units': 32 'learning_rate': 0.0005385497474460577 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
10	0.522615316	0.000786782	{'layers': 4 'units': 32 'learning_rate': 0.0003051284186367862 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
11	0.522615316	0.000786782	{'layers': 4 'units': 32 'learning_rate': 0.0003213920484489167 'batch_size': 16 'optimizer': 'SGD' 'activation': 'LeakyReLU' 'dropout': 1}
12	0.523063144	0.001445087	{'layers': 3 'units': 32 'learning_rate': 4.921760713545689e05 'batch_size' 16 'optimizer' 'SGD' 'activation': 'ReLU' 'dropout': 0}
13	0.527093596	0.001803771	{'layers': 3 'units': 61 'learning_rate': 0.0003651536514240078 'batch_size': 155 'optimizer': 'SGD' 'activation': 'LeakyReLU' 'dropout': 1}
14	0.531124048	0.003799866	{'layers': 3 'units': 32 'learning_rate': 0.00013014560366329914 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
15	0.536497985	0.014268893	{'layers': 3 'units': 32 'learning_rate': 0.00014661252374876767 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
16	0.546350202	0.015558273	{'layers': 3 'units': 32 'learning_rate': 5.397437258420407e05 'batch_size' 16 'optimizer' 'SGD' 'activation': 'ReLU' 'dropout': 0}
17	0.603224362	0.038523197	{'layers': 2 'units': 32 'learning_rate': 4.3758345900998806e05 'batch_size' 16 'optimizer' 'SGD' 'activation': 'ReLU' 'dropout': 0}
18	0.76309897	0.059573978	{'layers': 1 'units': 128 'learning_rate': 0.1 'batch_size': 16 'optimizer': 'Adam' 'activation': 'Tanh' 'dropout': 2}
19	0.817286162	0.073948383	{'layers': 1 'units': 32 'learning_rate': 0.1 'batch_size': 16 'optimizer': 'SGD' 'activation': 'Tanh' 'dropout': 2}
20	0.825347067	0.120159358	{'layers': 3 'units': 105 'learning_rate': 0.0018172027927374488 'batch_size': 61 'optimizer': 'Adam' 'activation': 'LeakyReLU' 'dropout': 1}

4.3 Final Pareto Frontier

Accuracy	DPD	Group	Index	Color	Model Parameters
0.836542768	0.116338007	botorch_rf	18	green	{'n_estimators': 39 'max_depth': 29 'min_samples_split': 5.505961537361145}
0.833407971	0.114764443	botorch_rf	17	green	{'n_estimators': 86 'max_depth': 25 'min_samples_split': 7.866268992424011}
0.832512319	0.107667483	botorch_rf	16	green	{'n_estimators': 17 'max_depth': 22 'min_samples_split': 8.001313924789429}
0.825347067	0.104777313	botorch_rf	15	green	{'n_estimators': 22 'max_depth': 19 'min_samples_split': 18.678343653678894}
0.81773399	0.101228835	optuna_rf	6	orange	{'n_estimators': 138 'max_depth': 7 'min_samples_split': 4}
0.817286162	0.073948383	botorch_nn	18	red	{'layers': 1 'units': 32 'learning_rate': 0.1 'batch_size': 16 'optimizer': 'SGD' 'activation': 'Tanh' 'dropout': 2}
0.801164353	0.071668797	botorch_rf	6	green	{'n_estimators': 207 'max_depth': 5 'min_samples_split': 8.665801167488098}
0.791312136	0.060894885	optuna_rf	0	orange	{'n_estimators': 82 'max_depth': 4 'min_samples_split': 3}
0.787729512	0.05670402	botorch_rf	2	green	{'n_estimators': 27 'max_depth': 4 'min_samples_split': 9.547285556793213}
0.775190327	0.050843605	botorch_rf	1	green	{'n_estimators': 22 'max_depth': 3 'min_samples_split': 5.235397458076477}
0.622033139	0.025240848	botorch_rf	0	green	{'n_estimators': 21 'max_depth': 1 'min_samples_split': 14.032552719116211}
0.546350202	0.015558273	botorch_nn	15	red	{'layers': 3 'units': 32 'learning_rate': 5.397437258420407e-05 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.536497985	0.014268893	botorch_nn	14	red	{'layers': 3 'units': 32 'learning_rate': 0.00014661252374876767 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.531124048	0.003799866	botorch_nn	13	red	{'layers': 3 'units': 32 'learning_rate': 0.00013014560366329914 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.527093596	0.001803771	botorch_nn	12	red	{'layers': 3 'units': 61 'learning_rate': 0.0003651536514240078 'batch_size': 155 'optimizer': 'SGD' 'activation': 'LeakyReLU' 'dropout': 1}
0.523063144	0.001445087	botorch_nn	11	red	{'layers': 3 'units': 32 'learning_rate': 4.921760713545689e-05 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522615316	0.000786782	botorch_nn	10	red	{'layers': 4 'units': 32 'learning_rate': 0.0003213920484489167 'batch_size': 16 'optimizer': 'SGD' 'activation': 'LeakyReLU' 'dropout': 1}
0.522615316	0.000786782	botorch_nn	8	red	{'layers': 4 'units': 32 'learning_rate': 0.0005385497474460577 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522615316	0.000786782	botorch_nn	9	red	{'layers': 4 'units': 32 'learning_rate': 0.0003051284186367862 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522167488	0	botorch_nn	0	red	{'layers': 3 'units': 72 'learning_rate': 0.09575703180698278 'batch_size': 156 'optimizer': 'Adam' 'activation': 'LeakyReLU' 'dropout': 1}
0.522167488	0	botorch_nn	7	red	{'layers': 3 'units': 32 'learning_rate': 8.777040908429805e-05 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522167488	0	botorch_nn	5	red	{'layers': 3 'units': 32 'learning_rate': 0.00010770610593330191 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522167488	0	botorch_nn	6	red	{'layers': 4 'units': 32 'learning_rate': 0.0003674564975397127 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522167488	0	botorch_nn	2	red	{'layers': 4 'units': 32 'learning_rate': 0.00025475806243148517 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522167488	0	botorch_nn	3	red	{'layers': 4 'units': 32 'learning_rate': 0.0005642859851861557 'batch_size': 16 'optimizer': 'SGD' 'activation': 'LeakyReLU' 'dropout': 1}
0.522167488	0	botorch_nn	1	red	{'layers': 4 'units': 32 'learning_rate': 3.0847611469672955e-05 'batch_size': 16 'optimizer': 'SGD' 'activation': 'ReLU' 'dropout': 0}
0.522167488	0	botorch_nn	4	red	{'layers': 4 'units': 128 'learning_rate': 0.1 'batch_size': 256 'optimizer': 'RMSprop' 'activation': 'ReLU' 'dropout': 0}

5 Future Work

Our framework successfully generates **27 high-quality Pareto points** across all model types, enabling banks to select optimal operating points based on their accuracy-fairness preferences.

Future Directions:

- Incorporate additional fairness metrics (Equalized Odds, Opportunity)

- Run the optimizer for days, we were only able to do for at max 2 hours (NueralNets for both Botorch and Optuna).

- Modify the Loss function with an α that business can according ot the importance they give :-

$$\mathcal{L}_{\text{total}} = \alpha \cdot \text{CrossEntropy} + (1 - \alpha) \cdot \text{DPD}$$

where α is a tunable parameter controlling the accuracy-fairness tradeoff.

- Extend to three-objective optimization