Titanic Data Analysis and Prediction

By Salome Ho

Data Cleaning

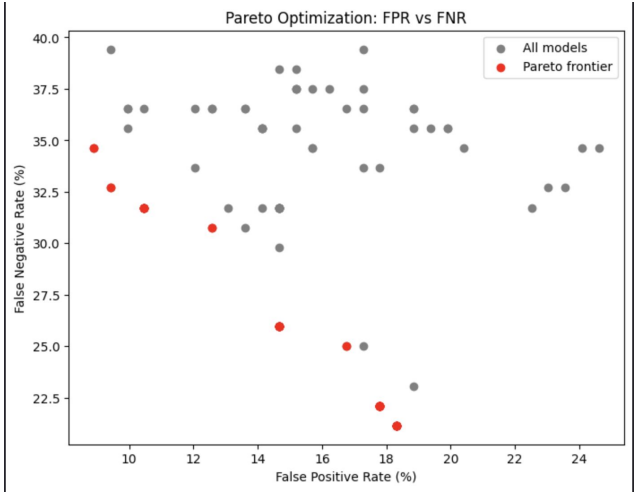
- The Kaggle dataset had the following fields (Class, Name, Age, Gender, Age, #siblings/spouses, #parents/children, Ticket, Fare, Cabin, and Port embarked.
- We decided to drop name, ticket, and the numeric part of the cabin.
- For missing data, if numeric we filled in with average, if not we used a probability distribution
- We split ⅘ of the train data into train, and the last ⅕ into validation data.

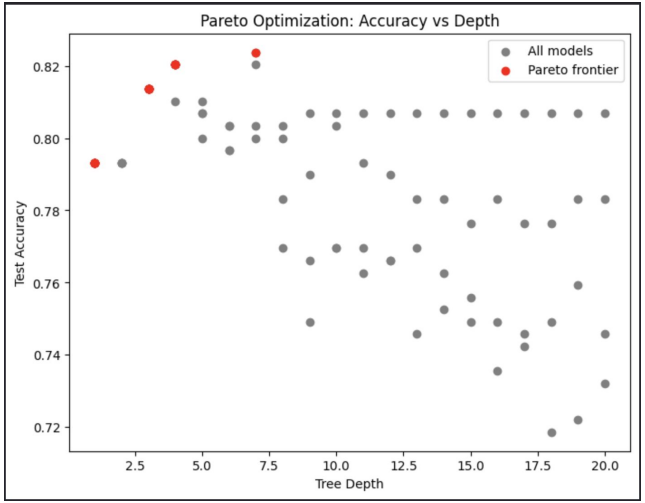
	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	2						

PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
464	2	0	48.000000	0	0	13.000	3	2
160	3	0	29.699118	8	2	69.550	4	2
48	3	1	29.699118	0	0	7.750	1	1
403	3	1	21.000000	1	0	9.825	7	2
619	2	1	4.000000	2	1	39.000	6	2

Decision Tree

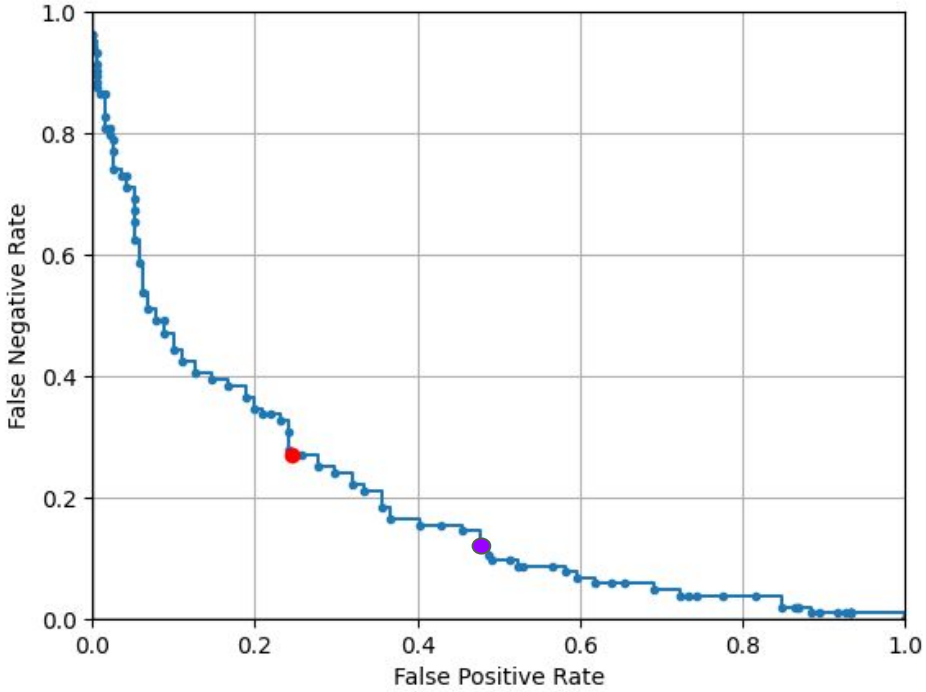
- Used scikit-learn DecisionTreeClassifier
- Dropped irrelevant features (Name, Ticket, Cabin #)
- Encoded categorical variables (Sex, Embarked, Cabin letter)
- Split 67% training / 33% test
- Tuned hyperparameters (max_depth, min_samples_split)
- Applied Pareto optimization for FNR vs. FPR





K - Nearest Neighbors Strategy

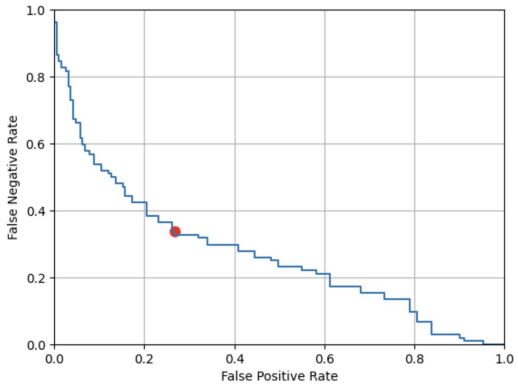
- Used gridsearch w/ regression to quickly tune parameters one at a time (in a greedy fashion)
- Mainly number of nodes and distance formula caused a difference.

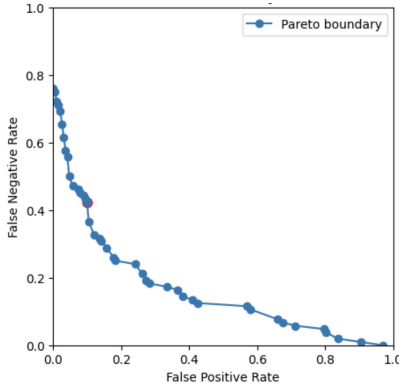


Neural Network

- Used scikit-learn MLPClassifier
- Scaled data with StandardScaler
- Hidden layers set to (50, 20) gave best trade-off between performance and training time
- Early stopping enabled to prevent overfitting
- Tuned threshold α to balance FPR and FNR
- Optimized using Pareto trade-off to minimize both false positives and false negatives

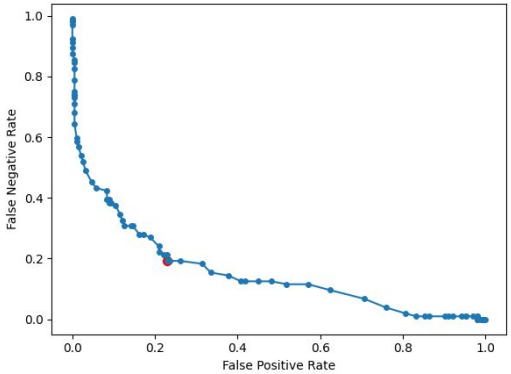
Type of Optimization	Best Val	Result
Hidden layers	(50, 20)	Stable convergence, accuracy = 81%
Early Stopping	Enabled	Prevented overfitting
Threshold tuning	α = 0.5	Balance FPR/FNR
Pareto trade-off	FPR vs. FNR	Reduced Pareto score from 0.44 -> 0.19



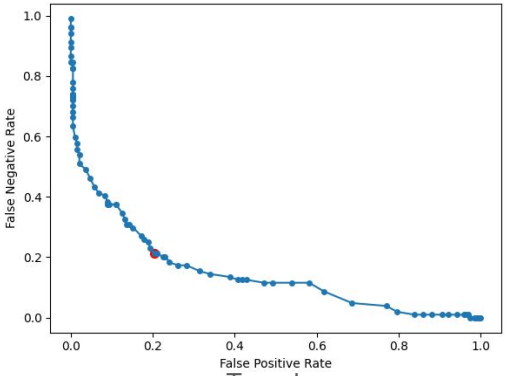


Linear SVC

- Preprocessing (group):
  - Removed irrelevant features (Name, Ticket, numeric Cabin)
  - Filled missing values and encoded categorical variables
- Scaling:
  - Applied StandardScaler to normalize features → improved solver stability and balanced feature influence
- Tuning:
  - Swept regularization parameter C (0.001 → 1000)
  - Accuracy remained consistently around ~80% across all values
- Thresholding:
  - Ran α-sweep across decision scores to tune between prioritizing survivor detection vs minimizing false alarms
  - Generated Pareto frontier (FPR vs FNR)
- Challenges:
  - Accuracy plateaued at ~0.80 regardless of scaling or C
  - Performance capped by dataset features rather than hyperparameters



Baseline

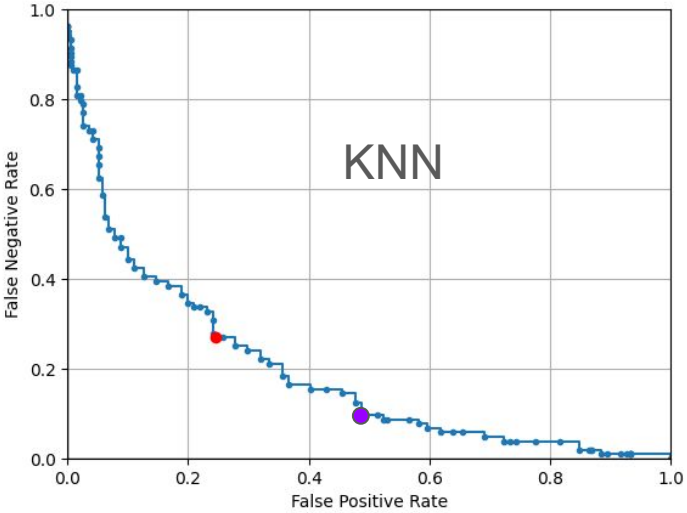


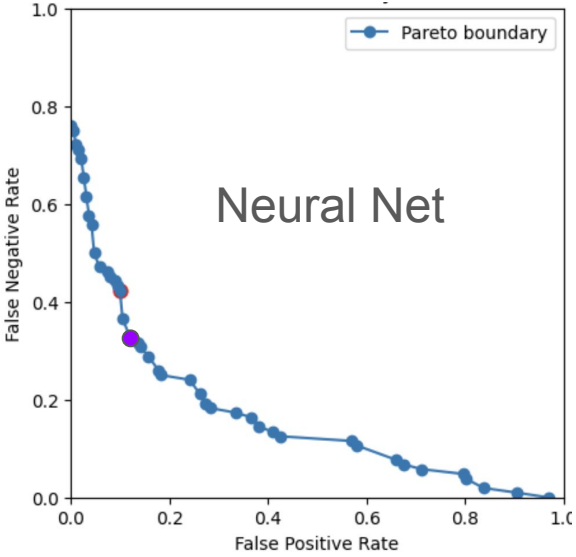
Tuned

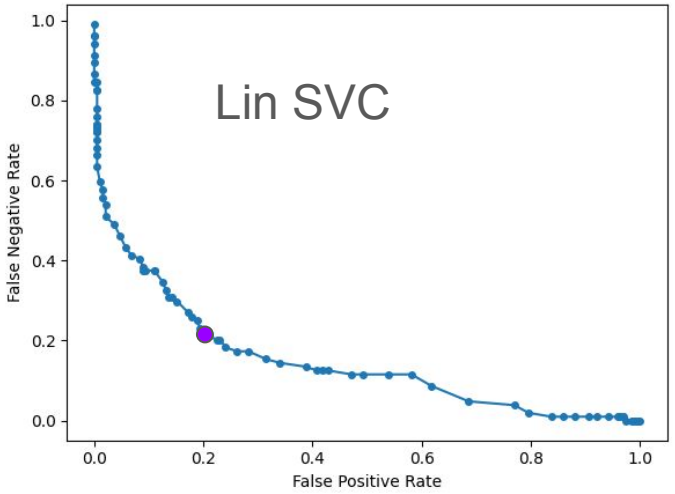
```
C=0.01  acc=0.800
C=0.01778279410038923  acc=0.797
C=0.03162277660168379  acc=0.797
C=0.05623413251903491  acc=0.797
C=0.1   acc=0.800
C=0.1778279410038923  acc=0.800
C=0.31622776601683794  acc=0.800
C=0.5623413251903491  acc=0.800
C=1.0   acc=0.800
C=1.7782794100389228  acc=0.800
C=3.1622776601683795  acc=0.800
C=5.623413251903491  acc=0.800
C=10.0  acc=0.800
C=17.78279410038923  acc=0.800
C=31.622776601683793  acc=0.800
C=56.23413251903491  acc=0.800
C=100.0  acc=0.800
C=177.82794100389228  acc=0.800
C=316.22776601683796  acc=0.800
C=562.341325190349  acc=0.800
C=1000.0  acc=0.800

Selected C=0.005623413251903491  (accuracy=0.803)
```

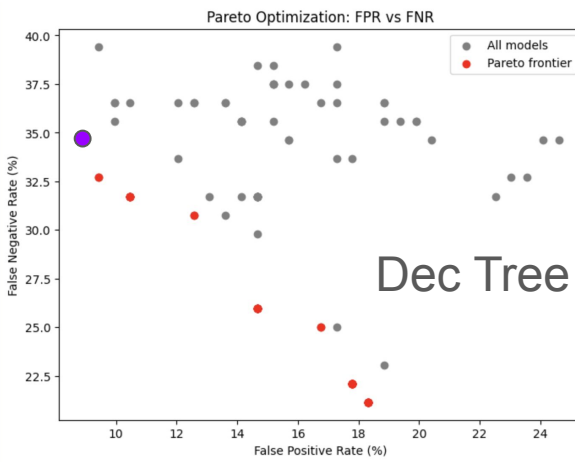
Mutual Pareto Optimality

The purple points are points on the pareto front of their respective algorithm that are also mutually pareto optimal with the other purple points



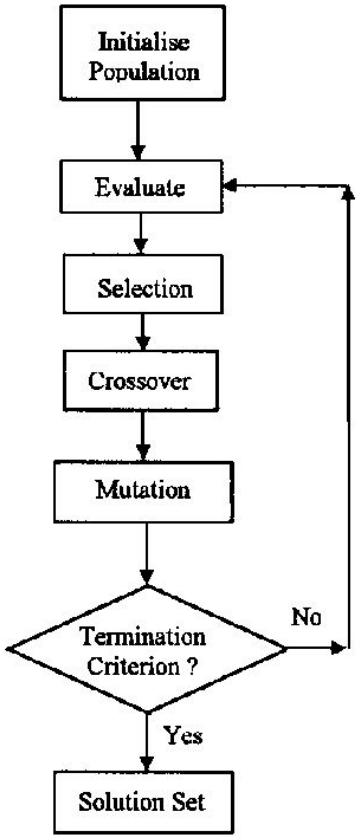






MOGP

- Extension of Genetic Programming to handle multiple conflicting objectives (minimize FPR and FNR)
- Produces a Pareto set of classifiers instead of one “best” model
- Explores trade-offs between survivor detection vs. false alarms
- We used DEAP (Distributed Evolutionary Algorithms in Python) to implement this, coding our own evolutionary loop with DEAP primitives



MOGP Design and considerations

- Preprocessing: Same as before
- Representation:
- Functions: +, -, *, tanh, custom div1000
- Terminals: Pclass, Sex, Age, SibSp, Parch, Fare, Cabin, Embarked,
- Ephemeral constant: random constant r (-1,1)
- Loosely typed (only floats). If >0 then survived else died.
- Fitness: minimize False Positive Rate (FPR) and False Negative Rate (FNR)
- Evolution process: half-and-half init → selection → crossover → mutation → evaluation
- Constraints: max tree depth = 17 to avoid bloat
- Used multi-objective setup (FPR vs FNR trade-off)
- Z-score normalization applied in evaluation to prevent clustering of FPR near 1
- Custom evolutionary loop (avoided DEAP built-in algos), also custom sampling
- Designed to balance interpretability (symbolic expressions) with performance

Code

Z score

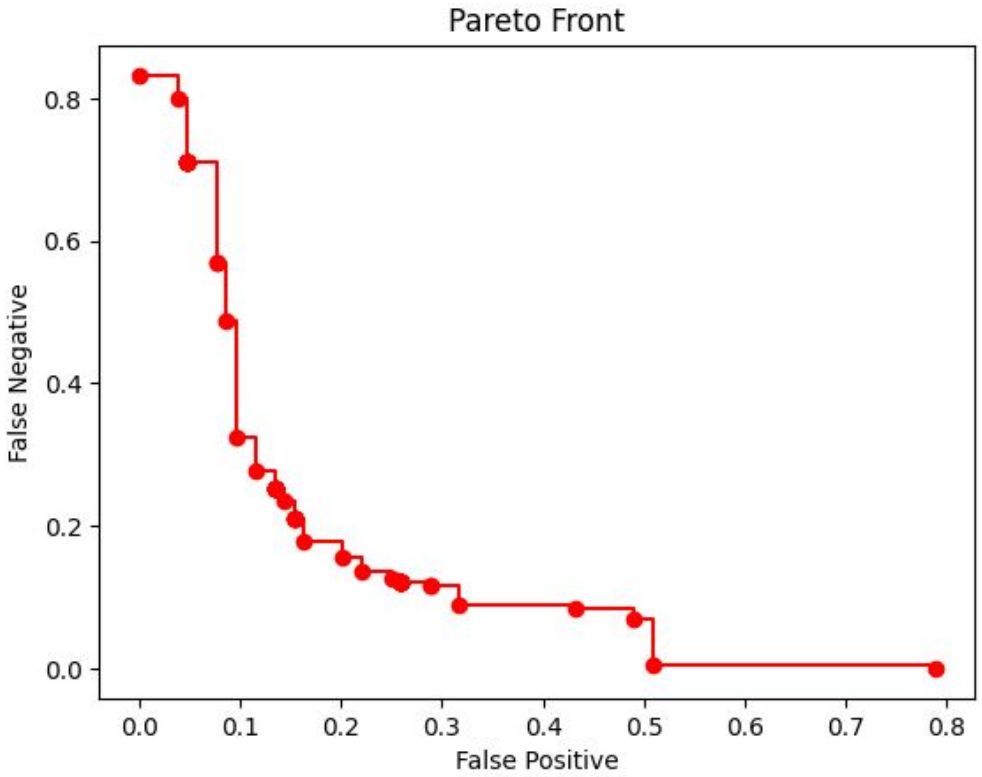
```
for i in X_test.index:
    X = X_test.loc[i]
    # Calculate Z-scores for each feature using pre-calculated stats
    Pclass_z = Zscore(feature_stats['Pclass']['mean'], feature_stats['Pclass']['std'], X['Pclass'])
    Sex_z = Zscore(feature_stats['Sex']['mean'], feature_stats['Sex']['std'], X['Sex'])
    Age_z = Zscore(feature_stats['Age']['mean'], feature_stats['Age']['std'], X['Age'])
    SibSp_z = Zscore(feature_stats['SibSp']['mean'], feature_stats['SibSp']['std'], X['SibSp'])
    Parch_z = Zscore(feature_stats['Parch']['mean'], feature_stats['Parch']['std'], X['Parch'])
    Fare_z = Zscore(feature_stats['Fare']['mean'], feature_stats['Fare']['std'], X['Fare'])
    Cabin_z = Zscore(feature_stats['Cabin']['mean'], feature_stats['Cabin']['std'], X['Cabin'])
    Embarked_z = Zscore(feature_stats['Embarked']['mean'], feature_stats['Embarked']['std'], X['Embar
```

Custom Select

```
def customSelect(pop,tournsize):
  chosen = []
  aspirants=[]
  for i in range(tournsize):
      aspirants.append(random.choice(pop))
  for a,i in enumerate(aspirants):
    for b,j in enumerate(aspirants):
      fi,fj = i.fitness.values, j.fitness.values
      if(a < b and fi[0] >= fj[0] and fi[1]>=fj[1] or fi[0] > fj[0] and fi[1]>=fj[1] ):
        break
    else:
      chosen.append(i)

  return random.choice(chosen)
pop = toolbox.population(n=100)
customSelect(pop,15)
```

Results

- We got to a Area under Curve of roughly 0.12.
- Some difficulties were that initially there was clustering around FN = 1.0, which we fixed by implementing a new custom tournament algorithm.
- Then we decided to be more harsh towards duplicates (which typically clustered around (1,0) and (0,1) and thus the edges were trimmed.



LLM Guided Evolution setup

- Modified preprocess.py to match previous algorithms' solutions
- Updated eval.py to calculate and display FPR/FNR metrics
- Changed constants in constants_titanic.py
  - PORT = 4444 (trial 1), 4446 (trial 2)
  - start_population_size = 32 (↓ from 128)
  - population_size = 32 (↓ from 128)
  - num_generations = 100 (unchanged)
- Chained inference servers: Server1 → Server2 → Server3
- Chained runs: Run1 → Run2 → Run3 (sequential with checkpoints)
- Model changes (only in trial 2 mentioned in next slide)
- Kicked off 24-hour (not fully) evolution run

```
3354758   ice-gpu LLMGE01_   swani8  R    4:55:34      1 atl1-1-03-012-23-0
3356171  pace-cpu  llm_opt   swani8  R    1:58:32      1 atl1-1-01-005-1-2
3356173 pace-cpu,  llm_opt   swani8 PD       0:00      1 (Dependency)
```

```
num_generations = 100           # was 100
start_population_size = 32      # was 128
population_size = 32            # was 128
```

```
# Confusion matrix and rates
tn, fp, fn, tp = sklearn.metrics.confusion_matrix(y_val, predictions).ravel()
fpr = (fp / (fp + tn)) if (fp + tn) > 0 else 0.0
fnr = (fn / (fn + tp)) if (fn + tp) > 0 else 0.0

# Print rates
print(f"False Positive Rate (FPR): {fpr:.6f}")
print(f"False Negative Rate (FNR): {fnr:.6f}")

# Rates shown in results files
results_text = f"{fpr:.6f},{fnr:.6f}"
```

LLM GE trials

- 2 trials
  - We ran 2 separate trials of LLM Guided Evolution
  - The main differences between these two was the LLM model and the seed data
  - Our first trial used a simpler seed (with just a Random Forest Classifier) and Llama 3.3
  - Our second trial used a more complex seed (with many classifiers) and Deepseek
  - Both trials had the same population size (32 individuals) and generations (100)

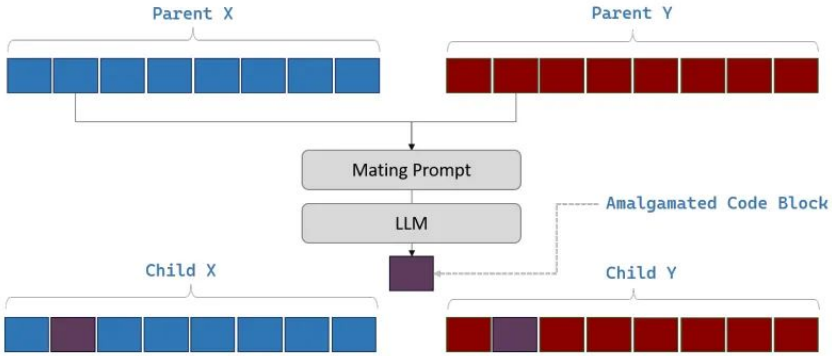


Figure 1: LLM Driven Code Block Mating

Monitoring during the evolution

```python
def quick_status(self):
    """Print a quick one-line status."""
    slurm_status = self.get_slurm_status()
    gen_info = self.analyze_current_generation()

    server_status = "🟢" if slurm_status.get('server_running') else "🔴"
    orchestrator_status = "🟢" if slurm_status.get('orchestrator_running') else "🔴"

    self.analyzer.load_fitness_data()
    fitness_count = len(self.analyzer.fitness_data)

    print(f"{server_status} Server | {orchestrator_status} Orchestrator | "
          f"Gen {gen_info['generation']} | "
          f"{gen_info['individuals']} individuals | "
          f"{fitness_count} evaluated | "
          f"{slurm_status.get('running_jobs', 0)} running | "
          f"{slurm_status.get('pending_jobs', 0)} pending")
```

We Needed a Monitoring Script

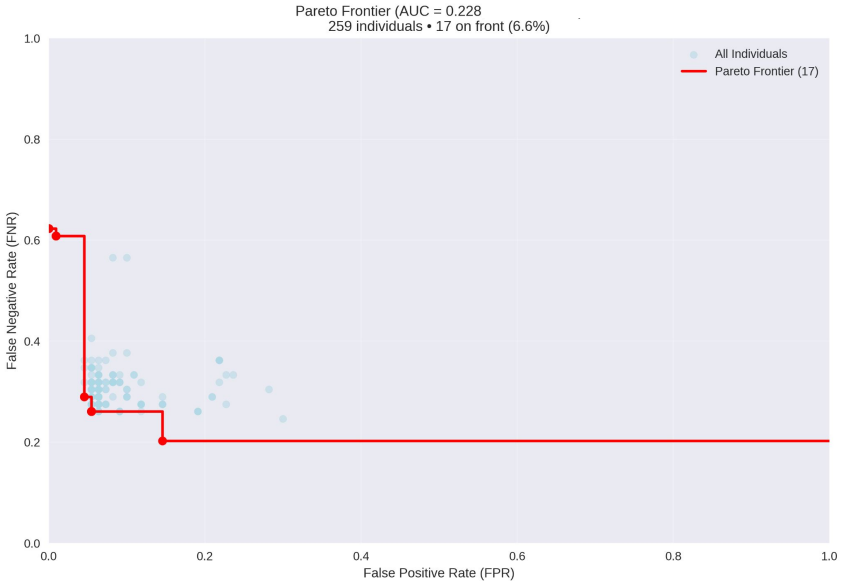
- Long Runtime: Evolution runs up to 24 hours across multiple generations
- Complex Job Dependencies: Multiple SLURM jobs (servers + runs) must coordinate
- Resource Bottlenecks: GPU queues can delay evaluation jobs for hours
- Data Scattered: Results spread across checkpoints, SLURM logs, and result files
- Bug Detection: Need to catch issues like infinite population growth early

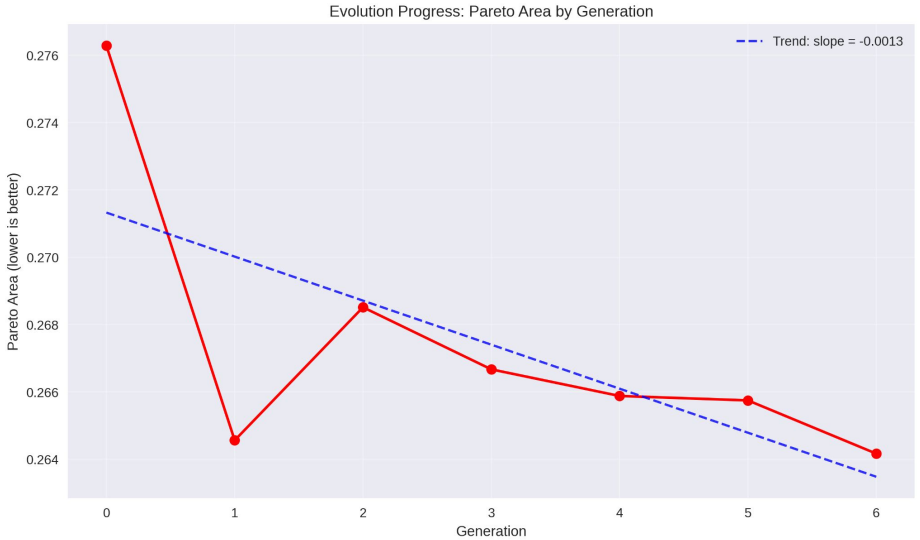
Our Solution: monitor_evolution.py

- **Real-time status tracking** - Current generation, population size, evaluation progress
- **Job queue monitoring** - SLURM job status and resource allocation
- **Automatic data aggregation** - Reads checkpoints, results, and logs automatically

```
[swani8@login-ice-3 llm-guided-evolution-fork]$ python monitor_evolution.py --quick
Loaded fitness data for 197 individuals
🟢 Server | 🟢 Orchestrator | Gen 4 | 305 individuals | 197 evaluated | 5 running | 20 pending
```

Results (Trial 1)



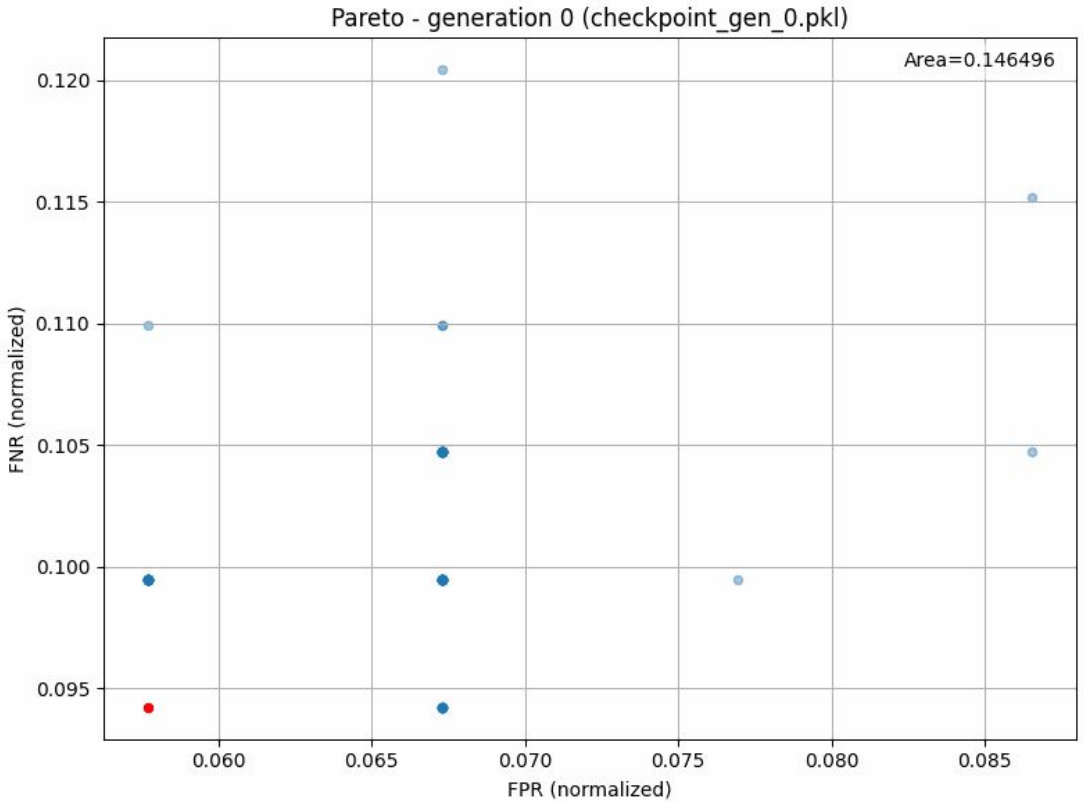


- 17 optimal solutions identified out of 259 individuals (6.6% on frontier)
- Best balanced accuracy: 84.23 % (FPR = 0.055, FNR = 0.261)
- Pareto area = 0.228
- Trade-off curve is clear: low FNR solutions come with moderate FPR, and no individuals strongly optimize for ultra-low FPR — likely due to dataset imbalance or lack of evolved “conservative” classifiers so far.

- 7 generations fully completed (Gen 0 → Gen 6)
- General trend towards improvement: 0.276 → 0.264 Pareto area
- Negative trend slope (-0.0013) = gradual refinement over time

Results (Trial 2)

- Deepseek-based
- Seed file has many scikit learn algorithms with some amount of default parameter tuning (generated by chatGPT)
- Results (final) on the right: area under curve of 0.132 though only one pareto optimal individual
- Current individuals generated
- Ran for 27 generations
- However, minimal significant evolution since the algorithm struggled to improve on the best individual, and the seed was optimized
- Common LLM optimizations: Gridsearch, model averaging.



All Algorithms Comparison

In general the MOGP algorithms outperformed the ML algorithms, we also saw that the LLM algorithms were near or inside the pareto boundary but did similar with Area Under Curve due to less individuals and therefore less points to be on such a boundary.

Red - MOGP
Purple - ML
Green - Trial 1 LLM
Yellow - Trial 2 LLM