



CSCI-409 Senior Project II — Final Report

Spring 2026

**AI for Quantitative Finance:
Predicting Structural Breaks in Financial Time Series
Using Machine Learning**

Project Advisor: Jose Berengueres

Group Members

Adilkhan Orynbay, Bakdaulet Bolat, Nurzhan Tenelbayev, Yerzhan Tugralin

Department of Computer Science

Nazarbayev University

1. Executive Summary

Our two-semester senior project is discussed here. Our machine learning system is used to detect structural breaks in financial time series. Sudden, significant, abrupt transformation in the behavior of a time series is a structural break. The average, dispersion or form of the data may change at one point. These breaks are quick to spot, and this is vital to traders, risk teams and forecasters.

Our work was based on the ADIA Lab Structural Break Challenge of CrunchDAO. There were 10,001 time series in the dataset. There was a label to each series: break or no break at a marked point. We used ROC AUC as our main score.

During the first semester we constructed a rudimentary system using scikit-learn. We did the Logistic Regression, Random Forest, Gradient Boosting, SVM, and MLP. Random Forest worked the best with ROC AUC of about 0.64. We have included a Streamlit training, testing, and chart dashboard.

During the second semester we re-architected the system. We implemented a PyTorch pipeline of feature pipelines on the GPU. It pulls 810 series of features with 12 modules. We selected 200 features after filtering. We inputted them to a group of LightGBM and XGBoost. The overall score was ROC AUC 0.8111 and a standard deviation of 1.42%. It is an increase of 28% compared to the first-semester baseline.

2. Introduction

2.1 Problem Statement and Motivation

Most quant trading work today is motivated by financial time series. When a shock strikes markets can turn overnight, as policy changes, wars, or even sudden mood swings can turn the rules upside down. Such sudden changes are termed as structural breaks.

Breaking makes saving money. When the regime switches, mean-reversion traders may blow up. To update their Value-at-Risk models, risk teams must have early warning. Predication models that are based on consistent information collide when the trend changes.

The traditional approaches, such as the Chow Test and Bai-Perron test, require powerful conditions. They also make the assumption that the data is linear and non-varying. The break point is frequently known beforehand. The actual financial information violates these regulations. It is large, fat tailed and volatile. In our project, we are avoiding such assumptions and learning patterns directly based on the data using machine learning.

2.2 Project Scope and Goals

We developed a framework to categorize each time series as break or no break. The objectives were:

- Extract a broad range of characteristics of raw time series data.
- Compare multiple machine learning models.
- Optimize the optimal models that are more feature-rich and have an ensemble.
- Install an interactive Streamlit dashboard to use.
- ROC AUC, cross-validation, and feature rankings: Test system.

2.3 Report Organization

In section 3, we are provided with the constraints that we were bound to. Section 4 examines previous literature. In section 5, we discuss our system design and features we developed. Section 6 presents a step by step of what we had done on a semester by semester basis. The results are indicated in section 7. Section 8 concludes and refers to further work.

3. Identification of Constraints

3.1 Economic Constraints

We only used free compute. The work with GPUs was done on personal computers having CUDA cards. We considered cloud training, however, we did not do it because of the cost. This influenced the extent of tuning we could run. It also terminated large deep learning executions using recurrent neural networks, which require much compute time.

3.2 Technical and Data Constraints

CrunchDAO established the rules of ADIA Lab. We were not able to share the data or incorporate external data. These were univariate time series, organized in one series. That inhibited most cross-asset features. The length of each series was varied and was between 1,000 and 3,478 points. We did this using NaN-padding and masked math. The distribution of the classes was uneven, 71 percent no-break, and 29 percent break. We used class weights to fix this.

3.3 Ethical and Social Constraints

The system assists individuals in making decisions, but not making decisions on their behalf. We do not give a yes or no call but a chance score. The information does not contain any personal data. It is artificial financial information of the competition.

3.4 Sustainability Constraints

All work done with GPUs involves sensitive task management. We eliminate waste through use of batch operations. This dragged feature saves hours and cuts it down to some few minutes. We stored features and modeled trained models to avoid repeat runs.

3.5. Health, Safety, and Legal Constraints

Our activity is compatible with GDPR provisions on anonymity data. No real tickers or personal data of the contest participants. The system was never put into live trading. The EU AI Act has driven us towards comprehensible, legible models. This is why we preferred tree-based approaches to black box deep nets.

3.6. Manufacturability Constraints

The system should be practical to be developed within the time, tools and technical knowledge of the team. This limitation influences the choice of the technology (e.g., Python, FastAPI, PostgreSQL) and promotes the modularity of the system architecture to guarantee its maintainability and scalability.

3.7. Sustainability Constraints

The maintainability, scalability and ability to adapt to new data and models will determine the long-term usability of the system. The project deals with this through modular design, version controlled models and extensible architecture. Nevertheless, this dependency on external APIs could impact sustainability in the long term should data access be modified.

3.8. Impact on System Design and Operation

All these constraints have an impact on important design and implementation decisions. Financial and environmental constraints promote the optimal choice of models and less computational costs. The use of probabilistic outputs and transparent visualizations are driven by ethical and social considerations. Data acquisition strategies are influenced by regulatory constraints whereas the system architecture and scope are influenced by technical feasibility. Complexity versus interpretability, performance versus cost and scalability versus development time Trade-offs are necessary between model complexity and interpretability, performance versus cost and scalability versus development time.

4. Background and Related Work

4.1 Classical Structural Break Detection

The first official test of breaking was the Chow Test of 1960. It also tests whether there are different coefficients of a linear model in two halves. It assumes the set up of the break point beforehand and assumes a linear connection. Multiple unknown break points were allowed in a Bai-Perron method of 2002. It still makes the assumption of the piecewise-linear data. The battles of both tests are with nonlinear and messy financial series.

CUSUM was an off-shoot of factory quality control. It comprises running totals of variation between a target value. It is suitable in the mean changes. It is not as accurate at identifying change of variance or shape.

4.2 Machine Learning Approaches

Change point work has been taken over by machine learning. The review was done on a variety of methods by Aminikhanghahi and Cook (2016). They demonstrated that ML is more adaptable compared to parametric tests.

Random Forest (Breiman, 2001) are tree ensembles that are effective in regime detections. They support feature mixes which are not linear and prioritize features by importance. The majority of tabular data competitions are now dominated by gradient boosting models such as LightGBM and XGBoost. They are fast, handle missing values and are regularized. LightGBM builds trees leaf-by-leaf, using loss as a criterion. They are boosted by XGBoost. The errors in these two paths are different and hence a good pair in an ensemble.

RNNs such as deep learning and LSTMs have been experimented with sequential change work. These involve more information and require much computations. Their tabular boosting models with well-tuned returns are mottled.

4.3 Feature Engineering for Time Series

Good features have a tendency to be more important than model choice. Truong, Oudre, and Vayatis (2020) conducted a review of change point methods. The comparison of segments is the key, and they presented statistical features. Interesting groups include the moments (mean, variance, skew, kurtosis), distance measures (entropy, KL, KS), autocorrelation and local features at the break point. Our project extends this with a 12-module pipeline which is based on GPU.

5. Project Approach

5.1 System Design

We have five layers in our system. The data is loaded by the first layer. The second runs feature extract of GPU. The third executes feature extraction of CPU. The fourth is used to filter and pick features. The fifth is the trainee, and the dashboard operator.

Parquet files are read by the data layer. It fetches the break position out of the period column (0 before, 1 after). It transforms each series into a padded tensor of gpus.

There are 11 modules in PyTorch in the GPU layer. A feature group is covered by each of the modules. Each of the 10,001 series is simultaneously run as batched tensors on CUDA. That drops the time off hours to a rough of five minutes.

A single CPU module supports features that require scipy. These are approximate entropy, Hurst, ADF and sample entropy. We paralleled them using joblib.

Cross-features are made with a post-merge module. It matches the features of GPUs and CPUs. An example of this is sample entropy times the dispersion index.

5.2 Feature Engineering Pipeline

Feature work was the core of the second semester. The pipeline builds 810 raw features across 12 modules. The table below shows the groups.

Module	Category	Features	Compute	Description
1	Statistical Basics	~110	GPU	Mean, variance, skewness, kurtosis, dispersion index, CV, quantiles
2	Distribution	~130	GPU	Entropy, KL divergence, Bhattacharyya distance, Hellinger distance, tail ratios
3	Time Series	~120	GPU	Autocorrelation, spectral centroid, rolling stats, zero-crossing rate
4	Boundary	~120	GPU	Boundary jump, local variance ratios, CUSUM, gradient breaks
5	Financial	~120	GPU	Volatility, Value-at-Risk, Sharpe ratio, max drawdown, vol clustering
6	Interactions	~80	GPU	Cross-module products, polynomial features, stability tests
7	Advanced	~230	GPU	Scan stats, MA crossover, complexity measures, residual analysis
8	CPU Features	~90	CPU	Approximate entropy, Hurst exponent, DFA, ADF, sample entropy
9	Pattern/Regime	~80	GPU	Phase changes, regime change intensity, change point scores
10	Transition	~50	GPU	Jerk coefficient, curvature jump, CUSUM recovery, gradient decay
11	Priority	~60	GPU	Compound mixes of the top-performing features
12	Post-Merge	~50	NumPy	Cross-features pairing GPU and CPU features

Feature Selection

We filtered the 810 features with a number of filters. A sanity check eliminated 8 drop dead features. The next features cut were ten near-constant. We eliminated 324 features that had a Pearson correlation greater than 0.93. The other rankings were based on a composite score: Mann-Whitney U-test effect, point-biserial correlation, and LightGBM importance. We selected the 200 best.

5.3 Model Architecture

First Semester: Baseline Models

We constructed six scikit-learn models; Logistic Regression, Random Forest, Gradient Boosting, SVM, MLP, and a Voting Ensemble. We trained them on basic stats from each series. Random Forest was the winner with ROC AUC of approximately 0.643.

Second Semester: LightGBM + XGBoost Ensemble

We reduced the list to two boosting models. LightGBM was trained with 2,000 estimators, learning rate 0.01, 80 leaf, max depth 12, feature fraction 0.6, and L1/L2 of 1.0. XGBoost parameters were estimators=1,000, learning rate= 0.01, max depth=14, column sample= 0.7 and subsample= 0.9. Both fixed the imbalance of classes with scale_pos_weight of 2.44. Both were GPU supported.

We applied an average of the two model probabilities. This was successful since the leaf-wise growth of LightGBM and level-wise growth of XGBoost commit different errors. The weak points of each one are cancelled by simple averaging. We also tested stacking and blending as well that neither beat the simple average on our data.

5.4 Tools and Tech

- PyTorch to extract features using Gpus.
- Gradient boosting using LightGBM and XGBoost that are GPU enabled.
- scikit-learn to test models, cross-validate and metrics.
- scipy and statsmodels to perform statistical tests such as ADF, Hurst and entropy.
- Pandas and NumPy to handle data and do numerical tasks.
- Streamlit and Plotly to the dashboard and interactive charts.
- Matplotlib and seaborn to make reports with static plots.
- Git and GitHub version control, branches and code reviews.
- Python notebooks to develop quick prototypes and tests.

5.5 Team Roles

Team Member	Main Tasks
Nurzhan Tenelbayev	GPU feature pipeline architecture and implementation; feature extraction 12 modules; NaN-mask logic and variable length series processing; feature verification and selection (810 -200 features)
Adilkhan Orynbay	Implementation of baseline models and benchmarking; setting up evaluation framework (ROC AUC, cross-validation, threshold tuning); ablation studies and model comparison in both semesters.
Bakdaulet Bolat	LightGBM and XGBoost ensemble design; hyperparameter tuning; class imbalance handling (scale_pos_weight, stratified folds); LSTM experiments and deep learning comparisons
Yerzhan Tugralin	Streamlit dashboard development (series browser, feature distribution tab, model comparison tab); data visualization and reporting charts, system integration and end-to-end testing.

6. Project Execution

6.1 First Semester (Fall 2025)

The base was established in the first semester. We read the contest of the ADIA Lab and analyzed the data. We constructed a complete pipeline.

The data stream loaded Parquet files, sanitized values and normalized features. We created 6 scikit-learn models. We created a Streamlit app to load data, train models, and display charts.

The key issue was that the data size (more than 10,000 series) had to be managed with great attention to the memory. We employed selective preprocessing and batching. Tuning Random Forest and Gradient Boosting took a lot of time. The default configurations tend to overfitting or slow running. We also saw that ROC AUC and accuracy alone missed a lot. Next we were going to add precision, recall, and F1.

Random Forest achieved ROC AUC 0.643 and accuracy 0.654 at the end of the semester. The prototype was successful, but weak results were shown. We were aware that we required superior features, not superior models.

6.2 Second Semester (Spring 2026)

Rethinking Feature Engineering

Scrapping the basic feature set was our largest call. Instead we constructed a 12-module GPU pipeline. It draws 810 characteristics per series. We did so because our first-semester models were starved to good features, but not model power.

Working on GPU features needed a more targeted approach because all of us had no extensive experience with PyTorch batches. Early implementations contained bugs in the NaN-mask code, they returned incorrect values to short series. We did this by hand debugging, line by line, and comparing each feature to scipy and NumPy, which took us about three weeks.

Handling Variable-Length Series

The series ranged between 1000 and 3478 points. We padded each series with NaN to the maximum length. All PyTorch ops needed to omit the NaN slots. We replaced regular torch calls with torch.nanmean, torch.nansum and our own versions masked, but we initially experimented with zero-padding, which was also biased on short series.

Finding the Real Boundary

The most minor victory was a minor repair. The first thing we did was to cut up each series in half. We switched to period column to real boundary. ROC AUC sprinted over by 8 points. This one alteration showed that data structuring influences are significant.

Model Choice and Ensemble

After feature stability had been achieved, we ran numerous model tests. The new features enabled even the Logistic Regression hit ROC AUC to be above 0.72. It was a tremendous advancement over the first semester. Gradient boosting prevailed though.

LightGBM and XGBoost were chosen as the ensemble members because of the head-to-head test. LightGBM hit 0.8097 (± 0.0125) in 10-fold CV. XGBoost hit 0.8081 (± 0.0156). The simple average ensemble got 0.8111 (± 0.0142).

We experimented with stacking a meta-model, weighted averaging and blending. No one was a true advantage. To reduce risk of overfitting, we used the simple average.

Feature Selection Rounds

The feature selection was done numerous times. The fact that the training on all 810 features was a bit worse than with a smaller set indicates that. We tested correlation cutoffs of 0.85, 0.90, 0.93, and 0.95. We tried 100, 150, 200, 250, and 300 features. Relatively, a 0.93 cutoff with 200 features was the best.

Difficulties We Hit

- GPU memory: Running all 10,001 series simultaneously in CUDA out-of-memory errors in Module 7. Our solution was to divide the work into sub-batches of 50 series.
- CPU bottleneck: Module 8 was slow. It was approximately 40 minutes even with joblib. We experimented with GPU ports of Hurst and ApEn. They did not work as the algorithms are too sequential.
- CV noise: Early 5-fold CV had a standard deviation above 3%. It was reduced to 1.42 when moving to 10-fold stratified CV.
- Class imbalance: 2.44:1 break made models underweight the break class. This was fixed with the use of `scale_pos_weight` and stratified folds.
- REST API was on our spring plan. We constructed one-half of it, but left it unfinished. Time became a factor as we stretched model scores. The primary interface was the Streamlit app.
- Tests on LSTMs: We tested LSTMs at the beginning of spring. They required a lot more time to train and failed to beat the boosting ensemble on our features. We reduced them in order to concentrate on the primary pipeline.

Teamwork

We did weekly sprints using GitHub pull requests. We monitored actions using GitHub Projects and Issues. Daily discussion occurred on Telegram. We spent time together once a week and planned and gave suggestions to one another. As the feature pivot increased in size, we rearranged positions. Nurzhan made the GPU pipeline full-time. Adilkhan and Bakdaulet paired up on model tests. Yerzhan dealt with the dashboard and kept things in touch.

7. Evaluation

7.1 Evaluation Method

In order to carry out the evaluation, we applied 10-fold stratified cross-validation. The ROC AUC was the main score since it is used to determine the performance of the model in terms of ranking break versus no-break. And we had other backup scores, such as accuracy at F1-best 0.37, precision, recall and F1. Every fold had the same proportion of the classes. We mean-folded, and provided standard deviation.

7.2 Model Comparison

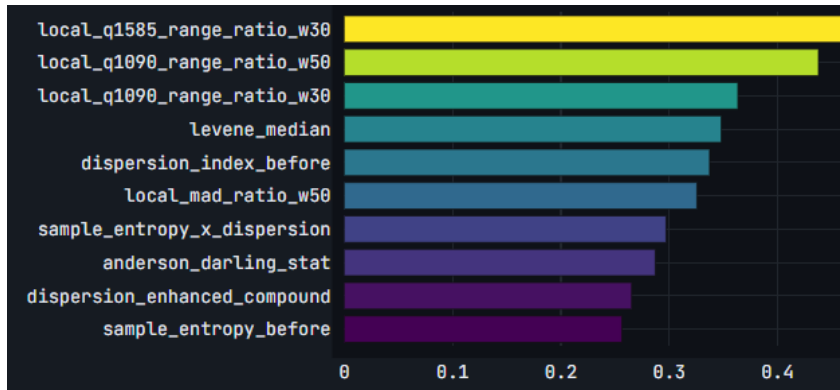
The table below shows every model we tested throughout 2 semesters

Model	ROC AUC	Std Dev	Semester
Logistic Regression (baseline)	0.623	± 0.021	Fall 2025
Random Forest (baseline)	0.643	± 0.019	Fall 2025
Gradient Boosting (baseline)	0.640	± 0.020	Fall 2025
SVM (baseline)	0.632	± 0.022	Fall 2025
MLP (baseline)	0.567	± 0.025	Fall 2025
Voting Ensemble (baseline)	0.639	± 0.018	Fall 2025
LightGBM (200 features)	0.8097	± 0.0125	Spring 2026
XGBoost (200 features)	0.8081	± 0.0156	Spring 2026
LightGBM + XGBoost Ensemble	0.8111	± 0.0142	Spring 2026

Our progress in this semester was significant: the ROC AUC score rose to 0.8111, which is a 0.168 gain in ROC AUC. The bulk of this was in the form of feature work rather than the model swap.

7.3 Feature Importance

This semester, we determined the top 10 best features by combining score:



It is evident that local boundary features dominate the list and that the breaks are most evident around the break point. Moreover, such distribution properties as entropy, KL, and KS were very useful, and other complex characteristics such as ApEn and sample entropy were included. Actual gains were seen in the cross-features such as sample_entropy_x_dispersion (ranked 7th).

7.4 Ablation Studies

We ran several teardowns to see what each piece added:

- Boundary position: The addition of the real period boundary vs. the midpoint added approximately 8 AUC points (0.73 to 0.81).
- Feature count: 100, 200, and 300 features scored 0.804, 0.811, and 0.808. 200 was about the optimal position.
- Ensemble: The average beat per single model of 0.14 to 0.30 AUC.
- GPU-only vs. GPU+CPU: 0.798 was received by the GPU-only. The addition of CPU features took it to 0.811.

7.5 Dashboard Review

We monitored our Streamlit dashboard throughout the project, refining it as we went. It's organized into three main tabs: the first tracks individual series against their predicted labels, the second breaks down feature distributions and correlations, and the third handles our performance metrics like AUC scores and CV statistics. Because we had to work quickly, the dashboard relies on pre-loaded data rather than live processing. While this makes it great for rapid testing, it does mean it isn't set up for real-time retraining just yet.

8. Conclusion and Future Work

8.1 Summary of Contributions

Our project went from a basic Random Forest (ROC AUC 0.643) to a GPU ensemble at 0.8111.

Key wins:

- A 12-module pipeline of feature puller that pulls 810 features in less than 5 minutes.
- Filter and rank that reduces 810 features to 200.
- An ensemble of LightGBM + XGBoost that combines leaf-wise with level-wise growth.
- A Streamlit dashboard to review, analyze features, and compare models per-series.
- A complete test run with a relative increase of 26.1 percent above a baseline.

8.2 Lessons Learned

- Attributes are important, not model selection. The 26-point increase was mostly due to improved features, rather than the model change.
- Data structure matters. Achieving an improvement of the boundary split was on its own sufficient to add 8 AUC points.
- Plain groups outdo decorative groups. Beat stacking and blending averaging on our data.
- GPU batch processing is scaled quickly. It was much faster to run all the series simultaneously than a per-series loop.

8.3 Future Work

Several paths forward stand out:

- Live monitoring: feed streaming market data and refresh models as the feeds arrive.
- Deep learning: repeat LSTMs and TCNs with an increased compute budget.
- Feature extraction: add more reliable features and diversify the methods to extract the most useful ones and in the right amount
- Multi-asset data: introduce cross-asset signals to extensive market shocks.
- Deploy to the cloud: deploy the system on Docker on AWS or Google Cloud.

- Expansive measures: include calibration curves, precision-recall plots and simulated trading P&L.

9. References

ADIA Lab. (2025). Structural Break Challenge. CrunchDAO Competition Platform.

<https://hub.crunchdao.com/competitions/structural-break>

Aminikhanghahi, S., & Cook, D. J. (2016). A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51(2), 339–367.

Bai, J., & Perron, P. (2002). Computation and analysis of multiple structural change models. *Journal of Applied Econometrics*, 18(1), 1–22.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Fawcett, T. (2005). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861–874.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

Hastie, T., Tibshirani, R. J., & Friedman, J. (2013). *The elements of statistical learning: Data mining, inference, and prediction*. Springer.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30.

McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 56–61.

Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32.

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Streamlit Inc. (2025). Streamlit documentation. <https://docs.streamlit.io/>

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, 167, Article 107299.

VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data*. O'Reilly Media.