## 1 General Description of the Idea

The main goal of the contest was to predict, for each order active at the end of a 15-minute test feed, the probability that the order will remain unmodified and uncancelled by its creator for the next 180, 300, and 600 seconds (denoted as $T_1, T_2, T_3$).

My approach consisted of:

1. Generating synthetic 15-minute test cases with ground truth labels using the provided long-duration training datasets.
2. Extracting features from each order based on its activity, market context, and relative behavior within the order book.
3. Training three independent LightGBM models (one per prediction horizon), tuned via Optuna, and binarizing outputs at a 0.5 threshold for final predictions.

## 2 Test Case Generation

Due to hardware constraints, I only utilized two of the four training archives. I randomly selected files from these datasets and extracted valid 15-minute intervals by choosing random offsets. Each generated test case included both the feed and its ground truth, resulting in approximately 400 test cases. The script `generate_data.py` was used for this process.

## 3 Feature Engineering

Each order in a test case was converted into a row in a DataFrame, where features captured the order's characteristics, activity, and book-relative position. The following features were used:

1. **final_price**: The final quoted price of the order. Essential as price level reflects aggressiveness and survival likelihood.
2. **final_quantity**: The final size of the order, as larger orders may attract execution or modification.
3. **offset**: Time (in seconds) since the start of the session at which the order was created. Early vs. late orders tend to exhibit different behaviors.
4. **start_time_sec**: Encodes the absolute time of the session (in seconds since midnight) to capture intraday market patterns.

5. **session_weekday**: Encodes the day of the week; markets can behave differently depending on the day.
6. **num_events**: Total events associated with the order, as more active orders are often modified or cancelled.
7. **modification_count**: Number of modifications (event type 2) applied to the order, as previously modified orders are more likely to change again.
8. **transaction_count**: Number of executions (partial fills) linked to the order, capturing its level of market interaction.
9. **avg_time_between_events**: Mean time between consecutive events for the order, as orders with frequent events behave differently.
10. **was_partially_executed**: Indicator (1/0) if the order has been partially executed, as execution can influence future cancellations.
11. **mean_exec_size**: Average executed quantity per partial execution, giving insight into execution intensity.
12. **modification_ratio**: Ratio between modifications and total events, measuring how ”dynamic” the order is.
13. **execution_ratio**: Ratio between transactions and total events, as highly executed orders tend to persist less.
14. **is_bid**: Boolean indicating whether the order is a BID; buy and sell sides behave differently.
15. **price_above_median_bid**: For BID orders, difference between price and median bid; aggressive bids are more prone to execution.
16. **price_below_median_ask**: For ASK orders, difference between median ask and price; similar rationale as above.
17. **price_distance_best**: Distance to the best price (bid or ask), as proximity to the top of book influences survival.
18. **time_since_last_mod**: Time since the last modification, as recently updated orders may be more stable.
19. **time_since_last_exec**: Time since the last execution, capturing the ”cooldown” behavior of orders.
20. **bid_proportion**: Proportion of bid volume relative to total (bid + ask), representing market pressure.

21. **p1, p2, p3**: Target labels for each horizon ($T_1, T_2, T_3$).

All features were computed using the `get_features.py` script, generating a consolidated DataFrame used for model training.

## 4 Modeling Approach

I trained separate LightGBM models for $p_1, p_2, p_3$:

- Models were deliberately trained with **low iteration counts and shallow depths**, preventing convergence to avoid overfitting.
- Outputs were **binarized at 0.5** using `binarize_output.py`, which yielded better competition scores despite less precise probability calibration.
- Hyperparameters were optimized using Optuna, with the scoring function applied to validation data after binarization.

This setup produced my best contest score.

## 5 Future Improvements

To enhance results, I identify several potential improvements:

1. Fix inconsistencies in synthetic test case generation, as some intervals produced unusual ground truth distributions.
2. Utilize all provided training archives (beyond the first two) for greater coverage.
3. Apply cross-validation rather than a single validation split, to maximize data efficiency.
4. Explore alternative regularization methods in LightGBM (e.g., early stopping, stronger L1/L2 penalties) instead of relying solely on low-iteration training.