Finsurvival Solution by Afinit - Hierarchical Feature Engineering

Jaehyeok Shin
Afinit
Seoul, South Korea
jace.shin@balancehero.com

Hyeonwoo Jeon
Afinit
Seoul, South Korea
bentley.jeon@balancehero.com

Abstract

We present a hierarchical feature engineering framework for survival modeling that addresses incomplete transaction data and data leakage in the FinSurvival Challenge [2]. Our approach introduces a multi-level indexing strategy that processes over 32 million records and generates more than 10,000 features across 10 tables. Key innovations include advanced missing transaction recovery, a hierarchical data architecture with eight core indices, and strict leakage prevention mechanisms. The framework uses the XGBoost Accelerated Failure Time (AFT) survival model with two-stage training and demonstrates significant improvements in the concordance index (C-index).

Keywords

FinSurvival, DeFi, Survival Analysis, XGBoost AFT, Feature Engineering, Missing Data Imputation, Data Leakage Prevention, Time-to-Event Prediction

1 Introduction

We present a hierarchical feature engineering framework for the FinSurvival Challenge [2] that processes over 32 million records and generates more than 10,000 features across 10 tables by means of a multi-level indexing strategy.

The primary contributions are: (1) hierarchical data architecture with 8 core indices enabling efficient joins across 12 data tables, (2) advanced missing data estimation using temporal and count-based inference, (3) strict data leakage prevention mechanisms ensuring temporal integrity, and (4) comprehensive feature engineering pipeline using XGBoost Accelerated Failure Time (AFT) Survival model with two-stage training.

The framework reconstructs complete transaction histories from incomplete data under strict temporal constraints and yields significant improvements in the concordance index (C-index).

2 Methodology

2.1 Data Architecture and Indexing Strategy

Our feature engineering approach employs a comprehensive data architecture that captures multiple dimensions of behavior and market dynamics through structured indices and aggregations.

2.1.1 Core Indexing Strategy. The feature engineering process transforms raw data into a structured, multi-dimensional dataset with hierarchical indices, including row_index, Index, user_index, trx_index, market_index, timestamp, liquidation_timestamp, liquidation_index, and liquidated_index (see Appendix A.1 for details).

ICAIF '25, Singapore
2025. ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

2.1.2 Data Tables Structure. The framework processes 12 interconnected data tables with over 32 million records including master table (32M rows), labels (5.4M rows), transactions (8.3M rows), market data (8.3M rows), liquidation events (11K rows), user-pool relationships (7.3M rows), and estimated data (see Appendix A.2 for details).

2.1.3 Key Design Principles.

(1) **Hierarchical Indexing**: Multi-level indices enable efficient joins and aggregations
(2) **Relational Integrity**: All tables can be joined using hierarchical indices to ensure temporal consistency and referential integrity
(3) **Normalization**: Left necessary columns for each table for efficient joins and aggregations
(4) **Scalable Architecture**: Modular design supports easy extension and modification

2.2 Missing Data and Log Imputation Strategy

Our approach handles missing data using sophisticated temporal and behavioral inference methods, identifying gaps in transaction sequences and liquidation events, then estimating missing information using multiple validation approaches.

2.2.1 Missing Data Types and Estimation Methods. The framework addresses six types of missing data with specialized estimation techniques (see Appendix A.3 for details).

2.2.2 Estimation Principles. The missing data estimation follows key principles:

(1) **Temporal Ordering**: All estimates maintain temporal ordering and logical sequence
(2) **Count-based Validation**: Uses transaction counts and amounts to validate estimates

2.2.3 Final Estimation Strategy. The final estimation process combines observed data with estimated missing data to create comprehensive datasets including estimated transactions (8.3M records), estimated liquidations (20.9K events), and estimated liquidated information (15.4K updates) (see Appendix A.3 for details). This approach provides a complete picture of user behavior for feature engineering.

2.3 Leakage Prevention Strategy

A critical aspect of our approach is the implementation of strict leakage prevention criteria to ensure that no future information is used in feature creation, which is essential for maintaining survival analysis integrity and preventing overoptimistic performance.

2.3.1 Leakage Prevention Criteria. The framework implements strict temporal constraints for different types of information (see Appendix A.4 for details) and follows key prevention principles:

(1) **Temporal Ordering**: All features use only information available at transaction time
(2) **Strict Boundaries**: Window functions use rangeBetween(-1 * period, -1) to exclude current transactions
(3) **Join Constraints**: All joins include temporal constraints to prevent future information leakage

2.3.2 Implementation Patterns. We adopt four implementation patterns to prevent data leakage: user transaction window functions, liquidation event joins, liquidated information joins, and market period features (see Appendix A.6 for code examples).

2.4 Feature Creation Strategy

Feature creation relies on two complementary approaches—temporal joins and window functions—for time-aware feature generation (see Appendix A.5 for details).

2.5 Model Training Strategy

Our model training strategy employs XGBoost Accelerated Failure Time (AFT) Survival model [1] with a two-stage training approach and comprehensive feature selection methodology.

2.5.1 Algorithm Selection. We adopt the XGBoost AFT survival model [1], which was reported as the best-performing approach in the FinSurvival Challenge [2].

2.5.2 Feature Selection Strategy. The feature selection process follows a two-step approach: (1) basic statistics selection to remove features with extreme sparsity, and (2) XGBoost importance-based selection with cumulative importance ratio threshold of 0.995.

The cumulative importance ratio is defined as follows:

- **Importance Ratio**: $r_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$ where f_i is the importance of feature i
- **Cumulative Importance Ratio**: $C_k = \sum_{i=1}^{k} r_i$ where features are ordered by descending importance

2.5.3 Two-Stage Training Process. Our training strategy employs a two-stage approach to capture both general and specific patterns:

Stage 1: Train by Same Outcome Event. Train the pattern for outcome event from other index event types to learn general patterns across different index events for the same outcome.

Stage 2: Incremental Training. Focus on training the pattern for exact index event - outcome event pair to specialize the model for specific index-outcome event combinations (see Appendix A.8 for hyperparameters).

This two-stage approach enables the model to first learn general survival patterns across different transaction types, then fine-tune for specific event combinations, resulting in improved concordance index performance.

3 Results

3.1 Feature Engineering Results

The feature engineering pipeline creates comprehensive feature sets across 10 feature tables, generating over 10,000 unique features with transaction features providing the largest feature set at 4,967 features per record (see Appendix A.8 for detailed breakdown).

3.2 Evaluation Results

As Table 1 shows, the final model improves the C-index over the baseline across all 16 event pairs on development, validation, and test splits, with the largest gains observed for liquidation outcomes on the test set.

4 Conclusion

We presented a hierarchical feature engineering framework for FinSurvival that achieves strong empirical performance.

4.1 Effectiveness

The framework achieves a higher concordance index (C-index) than the baseline. The proposed feature engineering methodology is straightforward and broadly applicable to financial settings.

4.2 Future Work

While this work establishes a solid foundation for survival analysis, several promising directions remain for future research:

4.2.1 Masked Multi-task Learning. Leveraging the multi-label nature of the dataset at the trx_index level, we can implement masked multi-task learning to predict multiple outcomes from the same index event, focusing on learning patterns for multiple outcome events rather than different index events for the same outcome.

4.2.2 Missing Transaction-based Data Augmentation. The estimated missing transactions could be incorporated as additional training samples to improve model robustness and generalization through data augmentation techniques.

4.2.3 Prediction Value as Input Variable for Other Tasks. A multi-task learning approach where prediction values from one task serve as input features for other related tasks, capturing complex interdependencies between different event types to improve overall prediction accuracy.

These future directions represent promising avenues for extending the current framework and improving survival analysis performance, though time constraints prevented their implementation in the current work.

A Appendix

A.1 Core Indexing Strategy Details

The feature engineering process transforms raw transaction data into a structured, multi-dimensional dataset with hierarchical indices:

- **row_index**: Sequential identifier for row number of each CSV file
- **Index**: Primary key for master table combining index_event, outcome_event, data_split, and row_index

Table 1: C-index comparison between baseline and final models across event pairs

index_event	outcome_event	baseline_dev	baseline_val	baseline_test	Final_dev	Final_val	Final_test
deposit	borrow	0.978	0.920	0.788	0.994	0.993	0.910
repay	borrow	0.926	0.864	0.743	0.986	0.984	0.970
withdraw	borrow	0.996	0.964	0.844	0.998	0.998	0.980
borrow	deposit	0.982	0.898	0.738	0.989	0.986	0.950
repay	deposit	0.984	0.915	0.741	0.991	0.989	0.940
withdraw	deposit	0.973	0.940	0.806	0.995	0.994	0.980
borrow	repay	0.946	0.872	0.811	0.987	0.984	0.910
deposit	repay	0.991	0.928	0.791	0.994	0.993	0.910
withdraw	repay	0.995	0.952	0.806	0.997	0.996	0.930
borrow	withdraw	0.982	0.913	0.687	0.988	0.985	0.920
deposit	withdraw	0.957	0.901	0.808	0.987	0.983	0.950
repay	withdraw	0.985	0.919	0.673	0.990	0.986	0.910
borrow	liquidation	0.986	0.925	0.565	0.997	0.993	0.810
deposit	liquidation	0.988	0.930	0.800	0.993	0.983	0.820
repay	liquidation	0.991	0.927	0.538	0.998	0.997	0.810
withdraw	liquidation	0.994	0.933	0.791	0.999	0.996	0.890
Total	**Score**	**0.978**	**0.919**	**0.745**	**0.993**	**0.990**	**0.912**

- **user_index**: User snapshot identifier at transaction time (sum of user transaction counts)
- **trx_index**: Primary key for transactions combining user and user_index
- **market_index**: Market snapshot identifier at transaction time (sum of market transaction counts)
- **timestamp**: Primary key for market_timestamp
- **liquidation_timestamp**: Timestamp for liquidation events (timestamp + timeDiff)
- **liquidation_index**: Primary key for liquidation events
- **liquidated_index**: Primary key for liquidated information updates

A.2 Data Tables Structure Details

The framework processes 12 interconnected data tables with over 32 million records:

- **Master Table (df)**: 32,028,004 rows with 99 columns containing the complete dataset
- **Labels**: 5,427,097 rows for training with survival analysis labels
- **Transactions**: 8,262,966 rows with normalized transaction data
- **Market Data**: 8,262,966 rows with market-level snapshots
- **Liquidation Events**: 10,966 observed liquidation events
- **User-Pool Relationships**: 7,333,392 user-pool combinations
- **Estimated Data**: Complete datasets combining observed and estimated missing data

A.3 Missing Data Types and Estimation Methods

The framework addresses six types of missing data through specialized estimation techniques:

- **Missing First Transaction**: Estimates user's missing first transaction using timestamp calculation when user_index = 2
- **Missing First Two Transactions**: Estimates user's missing first and second transactions when user_index = 3
- **Missing Middle Transaction**: Estimates gaps between observed transactions using temporal bounds
- **Missing Observed Transaction**: Detects missing observed transactions by comparing with actual transactions
- **Missing Liquidation**: Estimates missing liquidation events when liquidated information was updated but no liquidation event was recorded
- **Missing Liquidated Information**: Estimates missing liquidated information when liquidation events exist but no liquidated information update was recorded

A.3.1 Final Estimation Strategy Details. The final estimation process combines observed data with estimated missing data to create comprehensive datasets:

- **Estimated Transactions**: 8,271,294 complete transaction records (observed + estimated)
- **Estimated Liquidations**: 20,891 complete liquidation events (observed + estimated)
- **Estimated Liquidated Information**: 15,363 complete liquidated information updates (observed + estimated)

A.4 Leakage Prevention Criteria Details

The framework implements strict temporal constraints for different types of information:

- **User Transactions**: Use past user transaction information from first transaction to user_index transaction with window functions
- **Market Transactions**: Use past market information from the oldest info to market_index updated (inherently historical)
- **Liquidation Events**: Use past liquidation event information when liquidation_timestamp ≤ current_trx_timestamp
- **Liquidated Information**: Use liquidated information when liquidated updated timestamp_ub ≤ current_trx_timestamp
- **Any Information**: Use any kind of past information when information_timestamp < current_trx_timestamp

A.5 Feature Creation Strategy Details

A.5.1 Temporal Join Strategy. The first strategy uses temporal joins to attach historical information to each unique key:

- **Unique Key Identification**: Each feature creation process identifies appropriate unique keys (user, market, pool combinations)

- **Temporal Join Logic**: Joins current records with historical records where historical_timestamp < current_timestamp
- **Range Constraints**: Applies appropriate time windows (hour, day, week, month) to limit historical data scope
- **Aggregation**: Performs aggregations (count, sum, mean, max, min) on joined historical data

A.5.2 Window Function Strategy. The second strategy uses window functions with temporal partitioning and ordering to create features within defined time ranges:

- **Partitioning**: Groups data by right entity (user, pool, etc)
- **Temporal Ordering**: Orders data by timestamps within each partition
- **Range Definition**: Defines appropriate time ranges using rangeBetween(-1 * period, -1)
- **Window Aggregation**: Applies window aggregation functions for feature

A.6 Implementation Code Examples

A.6.1 User Transaction Window Functions.

```
window = (
  Window.partitionBy("user").orderBy("user_index")
)
month_window = window.rangeBetween(-1 * month, -1)
```

A.6.2 Liquidation Event Joins.

```
cond = liquidation["user"] == trx["user"]
cond &= (
 trx["timestamp"] >= liquidation["liquidation_timestamp"]
)
log = trx.join(liquidation, cond)
```

A.6.3 Liquidated Information Joins.

```
cond = liquidated["user"] == trx["user"]
cond &= (
  liquidated["timestamp_ub"] <= trx["timestamp"]
)
log = trx.join(liquidated, cond)
```

A.6.4 Market Period Features.

```
market_timestamp.rolling(
  index_column="log_at",
  period=period,
  closed="left",
)
```

A.7 Model Training Hyperparameters

Table 2 summarizes the hyperparameters used for feature selection and two-stage training process.

Table 2: Hyperparameters for Feature Selection and Two-Stage Training

Process	learning rate	earlyStopRounds
Feature Selection	0.35	2
Train by Same Outcome Event	0.2	2
Incremental Training	0.03	10

A.8 Feature Engineering Results Details

The feature engineering pipeline creates comprehensive feature sets across 10 feature tables:

- **Market Features**: 5,622,939 rows with 342 features for market trend analysis
- **Pool Features**: 6,983,281 rows with 540 features for pool-specific characteristics
- **User-Pool Features**: 7,333,392 rows with 520 features for user-pool relationships
- **Liquidation Features**: 506,818 rows with 434 features for liquidation events
- **Estimated Liquidation Features**: 616,085 rows with 290 features for estimated liquidation events
- **Liquidated Features**: 616,537 rows with 24 features for liquidated information
- **Estimated Liquidated Features**: 616,537 rows with 19 features for estimated liquidated information
- **Transaction Features**: 8,262,966 rows with 4,967 features for comprehensive transaction analysis
- **Estimated Transaction Features**: 8,271,294 rows with 4,967 features for estimated transactions
- **Liquidation Transaction Features**: 506,818 rows with 4,960 features for liquidation-transaction relationships

References

[1] Avinash Barnwal, Hyunsu Cho, and Toby Dylan Hocking. 2020. Survival regression with accelerated failure time model in XGBoost. *arXiv preprint arXiv:2006.04920* (2020). https://arxiv.org/abs/2006.04920

[2] Aaron Green, Zihan Nie, Hanzhen Qin, Oshani Seneviratne, and Kristin P Bennett. 2025. FinSurvival: A Suite of Large Scale Survival Modeling Tasks from Finance. *arXiv preprint arXiv:2507.14160* (2025). https://arxiv.org/abs/2507.14160

Received 20 October 2025