4th solution: XGBoost at the Core of Flight Recommendation

comparing GBDT, neural rankers, and ensemble methods on large-scale flight data

Xiang Zheng

Vișan Ionuț

Abstract

In this work, we present our solution for the Aeroclub RecSys 2025 competition, which focused on ranking flight options in a real-world recommendation setting. Our approach is built on gradient boosting decision tree models (XGBoost and LightGBM), complemented by extensive feature engineering across price, time, route, company, cabin, and user-profile dimensions. We further explored deep learning-based ranking architectures, which provided useful insights but showed limited performance compared to tree-based methods. To maximize predictive power, we developed an ensemble strategy that combines model confidence signals through a hybrid weighting scheme based on uniqueness (via PCA–Spearman residuals) and informativeness (entropy of confidence distributions). This ensemble delivered a modest but consistent improvement over individual models, achieving a leaderboard score of ~0.537. Overall, our solution demonstrates the strength of carefully engineered features and principled ensembling in industrial-scale recommendation tasks.

1. Background

Special thanks to the organizers for the flawless management and smooth progress of this competition.

Our team was made up of two people:

Vișan Ionuț - Master's student in Artificial Intelligence (2nd year) at the Faculty of Automatic Control and Computers, National University of Science and Technology Politehnica Bucharest (https://www.linkedin.com/in/ionut-visan/).

Xiang Zheng - Incoming graduate student in Statistics at Fudan University in China, starting in Fall 2025, affiliated with the School of Data Science.

Competition results:

- 4th place, Insights Award (0.53672) – ensemble strategy
- 3rd place, Single Model Award (0.53552) – XGBoost

2. Introduction

Recommender systems have become a cornerstone of digital platforms, enabling users to quickly identify relevant items from a large pool of alternatives. In the travel industry, this challenge is especially complex: travelers face multiple flight options that differ in price, duration, routing, cabin class, and carrier policies. The Aeroclub RecSys 2025 competition hosted on Kaggle, addressed this problem by providing a large-scale industrial dataset of real flight search sessions. The task was framed as a learning-to-rank challenge, where models must order flight options within each search group so that the selected option is ranked highly.

The dataset includes millions of records enriched with user, company, and flight attributes, offering rich opportunities for feature engineering and modeling. The official evaluation metric, HitRate@3, reflects a realistic recommendation scenario in which only the top-ranked items matter for user satisfaction. This setting makes the competition both practically relevant and technically demanding, requiring solutions that balance accuracy, generalization, and scalability.

In this paper, we describe our final solution. It is built on gradient boosting decision tree models (XGBoost and LightGBM), supported by extensive domain-inspired feature engineering, and further enhanced by an ensemble strategy that integrates complementary predictors. We also explored deep learning ranking architectures, which, while less effective in this setting, provided useful insights. Overall, our pipeline demonstrates that feature-driven GBDT models combined with principled ensembling deliver strong and reliable performance for large-scale recommendation tasks.

3. Related Work

Learning-to-rank has been approached through two main families of methods: gradient boosting decision trees (GBDT) and deep neural networks.

“Which Tricks Are Important for Learning to Rank?” (Lyzhin et al., 2023) [1] systematically evaluates gradient boosting frameworks such as LightGBM LambdaMART, highlighting which training objectives, loss formulations, and optimization tricks most strongly affect ranking performance.

“Industry Insights from Comparing Deep Learning and GBDT Models for E-Commerce Learning-to-Rank” (Lutz et al., 2025) [2] contrasts neural approaches with production-grade LightGBM models in large-scale e-commerce, showing why GBDTs remain highly competitive baselines and reporting lessons from online A/B testing.

FiBiNET (2019) [3] introduces a Squeeze-and-Excitation mechanism to adaptively re-weight feature embeddings, together with a bilinear interaction layer that models fine-grained relations between features, achieving strong gains in CTR prediction.

FiBiNET++ (2022/23) [4] improves efficiency by replacing the bilinear layer with a low-rank interaction module, reducing model size by an order of magnitude while retaining or improving accuracy in large-scale recommendation.

FM-Pair (2018) [5] extends Factorization Machines with a Bayesian Personalized Ranking (BPR) pairwise loss, shifting the objective from score regression to direct preference learning, which is more suitable for implicit feedback ranking tasks.

Neural Collaborative Ranking (2018) [6] builds on NeuMF by combining embeddings and a deep MLP, but optimizes with a pairwise ranking objective, explicitly contrasting positive and negative items to better capture relative user preferences.

4. Proposed Method

Our approach begins with a careful inspection of the raw flight search logs, ensuring data integrity and a clear understanding of group-level structures (ranker_id sessions). We then apply a systematic preprocessing pipeline that handles missing values, normalizes numerical variables, and encodes categorical attributes into compact integer indices suitable for embeddings. Building on this foundation, we design extensive domain-driven feature engineering, creating new attributes that capture temporal patterns (e.g., booking lead time, trip duration), economic indicators (e.g., price ratios, competitiveness signals), and contextual priors (e.g., carrier, route, and user-level statistics). To avoid redundancy and overfitting, we subsequently apply feature selection techniques that retain only the most informative signals.

Finally, we construct training pipelines tailored for different model families - tree-based learners such as XGBoost/LightGBM and neural ranking architectures. Each pipeline integrates preprocessing, feature handling, and model training in a reproducible workflow, enabling consistent evaluation across methods and facilitating ensemble strategies that combine their complementary strengths.

5. Dataset

The dataset used in the Aeroclub RecSys 2025 challenge comprises over 18 million real-world flight search records, grouped by the ranker_id field to represent individual user queries. Each record provides detailed information on price, duration, route, cabin class, airline, and user profile attributes, offering a rich and challenging environment for learning-to-rank. The training set is complemented by a test split containing several million additional instances, making this one of the largest and most realistic public benchmarks for travel recommendation. To enable the development of effective recommendation models at this scale, the dataset must undergo a series of preprocessing steps designed to ensure data quality, handle missing values, normalize heterogeneous features, and create model-ready representations.

6. Data Preprocessing

To ensure consistency and comparability across queries, the dataset was chronologically divided into training and validation partitions, with the last 10% portion of the training data reserved for validation. Continuous features such as price and duration exhibited heavy-tailed distributions and were stabilized through logarithmic transformation. Records corresponding to multiple pricing options for the same flight were aggregated at the option level using statistical descriptors (minimum, maximum, mean, rank), thereby reducing redundancy.

Additional metadata extracted from raw sources, including categorical attributes and binary indicators, was incorporated after alignment through surrogate keys. Noisy or misaligned entries were retained only when they contributed measurable improvements under validation. The preprocessing stage thus produced a structured representation that facilitated subsequent feature engineering and model training.

The feature.py files serve as the main reference point, encompassing both feature engineering and feature selection. While their core structure remains consistent, each model relies on a slightly adapted version, allowing us to introduce variation and broaden the exploration of both modeling strategies and data representations.

6.1. Feature Engingeering

We construct features across complementary semantic layers to capture behavioral, operational, and contextual signals that drive flight selection. Below we outline the main families, how they are computed, and why they matter for learning-to-rank.

- **Rank-based features** = relative price and duration ranks within each search, plus simple interactions between them to reflect trade-offs.

- **Time and itinerary features** = departure/arrival hours, business-friendly or red-eye flags, one-way vs. round-trip, direct-flight indicators, and stay duration bins.

- **Price context features** = flags for cheapest or outlier options, distance from group medians, and stabilized log transforms for skewed variables.

- **Cabin class features** = indicators of economy vs. premium cabins, presence of business class, and consistency across flight segments.

- **Carrier and flight features** = popularity and selection ratios of marketing carriers, diversity of companies and users per carrier, and cabin-specific preferences.

- **Company features** = aggregate preferences at company level, such as average accepted price, share of direct flights, or typical travel times.

- **Frequent-flyer features** = matches between offers and the traveler's loyalty programs, including major-carrier flags.

- **User profile features** = historical averages and counts of past selections, with differences to the current offer to model personal tendencies.

- **Airport and geography features** = international vs. domestic flag, time-zone differences, and contextual metadata from airport databases.

- **Corporate tariff features** = frequencies and averages associated with corporate tariff codes, plus policy-compliance indicators.

- **Group and route features** = candidate set size, route popularity, and aggregated statistics at the search-query level.

- **Robustness handling** = missing-value flags, log transforms, and winsorization for extreme distributions, ensuring stability and preventing leakage.

6.2. Feature Selection

An iterative selection process was applied to reduce redundancy and mitigate overfitting. Preliminary gradient-boosted models were used to estimate feature importance, with consistently negligible variables removed. For high-cardinality attributes, only occurrence-based statistics were preserved, while complex aggregations were discarded due to instability. Highly correlated variables were simplified by retaining only the more effective transformation, such as the logarithmic variant of continuous attributes. Features that did not demonstrate predictive value, such as certain metadata fields, were excluded. The resulting feature set comprised approximately 160 variables, balancing expressiveness and efficiency.

7. XGBoost

XGBoost (xgboost folder) was employed with two ranking objectives, namely rank:ndcg and rank:pairwise. Both settings were evaluated under cross-validation, with rank:ndcg generally yielding slightly superior results. Hyperparameters such as maximum tree depth, minimum child weight, and subsampling ratios were tuned to control model complexity and mitigate overfitting. Training on the full dataset required approximately ten minutes on a single GPU. The model achieved private leaderboard scores in the range of 0.532–0.536, representing the best results among the methods evaluated.

8. LightGBM

LightGBM was trained using the lambdarank objective, which directly optimizes ranking-based metrics. Its efficient implementation for categorical variables and histogram-based training enabled rapid convergence. Parameter tuning focused on leaf number, minimum data per leaf, and feature sampling ratios. Training time was also within ten minutes, with validation scores slightly lower than XGBoost. On the private leaderboard, LightGBM achieved scores in the range of 0.515–0.520. The distribution of feature importance differed between the two frameworks, indicating complementary perspectives on the data.

9. Deep Learning

To complement our gradient-boosted models, we designed a hybrid neural ranking architecture tailored for group-wise recommendation. The goal was to combine the strengths of linear models (capturing simple additive effects), embedding layers (capturing categorical semantics), and a deep nonlinear tower (learning higher-order feature interactions).

This modular design allowed us to test both shallow and deep representations while remaining scalable to millions of training samples.

The files from the archive that we refer to (deeprec_validate.py, deeprec_full_training.py) can be found in the dl_ranker folder.

9.1. Loss Function

The loss function used to train our model was chosen after extensive experimentation with different ranking objectives. We ultimately adopted a pairwise ranking loss inspired by RankNet, but tailored to the competition's needs. This loss compares items within the same query group and penalizes the model whenever a less relevant option is ranked above a more relevant one. Unlike a standard formulation, our variant places additional emphasis on the top-k most relevant items, ensuring the model learns to prioritize the flights that matter most in practice. It also incorporates a margin to encourage clearer separation between relevant and irrelevant choices, and a temperature factor to balance sharpness and stability. This design proved to be the most effective, driving the model to consistently improve recommendation quality at the very top of the ranked list.

9.2. Performance Metrics

All metrics are computed within each query group (ranker_id) and then averaged across the batch. We ignore very small groups during validation (min_group_size=10) to avoid noisy/unstable estimates. Padding added by the dataloader is masked via the true lengths array, so only real items are evaluated.

HitRate@k:

Measures the probability of "at least one correct item" appearing in the top-k of the model's ranking for each group. It's a coarse but highly practical signal: if any positive label is in the top-k, that group counts as a hit (1), otherwise 0. We average over valid groups. This aligns with user-centric goals ("did we surface something relevant near the top?"), though it is insensitive to exact position within the top-k.

NDCG@k:

Normalized Discounted Cumulative Gain captures both relevance and position. Highly relevant items contribute more, and earlier positions are discounted less (so pushing the right item to rank 1 matters more than rank 3). We normalize by the ideal DCG for the same group to make scores comparable across groups. This is a good proxy for ranking quality near the top and is more position-sensitive than HitRate.

MAP@k:

Mean Average Precision focuses on precision across all relevant items in the top-k. For each relevant hit at position *j*, it measures precision up to *j* and then averages across all relevant hits. MAP@k thus rewards early and consistent placement of relevant items, complementing HitRate (which only asks for "at least one") and NDCG (which discounts by rank).

9.3. Architecture

Among the different neural ranking architectures we explored - including transformer layers, attention mechanisms, SENet-style modules, and FiBiRanker variants - the simplest design turned out to be the most effective. Despite its straightforward structure, this architecture consistently outperformed more complex alternatives, striking a balance between expressiveness and training stability.

The model takes as input both categorical and numerical features. Categorical variables are encoded using two parallel embeddings: one-dimensional embeddings that serve as linear terms, and 64-dimensional dense embeddings that capture richer representations. Numerical features are standardized and processed through both a linear projection and concatenation with embeddings for deeper modeling.

The architecture consists of two main pathways. A linear pathway aggregates first-order signals from categorical and numeric features, ensuring that simple relationships are directly modeled. In parallel, a deep pathway concatenates all embeddings and numeric features and processes them through a multi-layer perceptron with hidden layers of sizes 512, 256, 128, and 64. Each layer uses Batch Normalization, ReLU activations, and Dropout (0.1) to enhance generalization.

The outputs of both pathways are combined into a single relevance score per item. Training is performed with a pairwise ranking loss (xRankNet variant) that emphasizes correctly ordering the top-k items within each group, aligning optimization with evaluation metrics such as HitRate@3 and NDCG@3.

9.4. Data Preprocessing and Feature Preparation

Effective preprocessing is essential in recommendation and ranking tasks, since raw booking logs combine heterogeneous feature types, missing values, long-tailed distributions, and group-relative dynamics. Our pipeline transforms the raw parquet files into a compact, leakage-safe, and model-ready dataset by performing targeted feature engineering, standardizing categorical and numerical variables, caching intermediate results for reproducibility and efficiency, and ensuring that training statistics are isolated from validation and test data.

What the pipeline engineers:

- Temporal & trip structure: converts durations into minutes, derives time bins (business hours, red-eye flights), stay length, and booking lead time.
- Economic & route features: price/tax ratios, duration ratios, one-way vs. direct flags, codeshares, and route popularity.
- Group-relative signals: within-query (ranker_id) ranks and percentiles for price/duration, as well as smooth interactions capturing competitiveness between options.
- Contextual priors: per-company, per-carrier, and per-user aggregates (selection counts, ratios, average prices/durations), stabilized with log transforms and KDE-based priors for sparse users.
- Cabin, loyalty, and geography: cabin-class indicators, frequent flyer matches, corporate tariff effects, and airport metadata (country, timezone).
- Robustness: winsorization, log transforms, and imputation to handle outliers and missing data.

All of these transformations are implemented in feature_engineering and later filtered by feature_selection. Both defined in our feature.py from dl_ranker folder.

The function load_or_prepare_data handles the orchestration of raw data ingestion and feature engineering. It either loads a cached, preprocessed parquet file for efficiency or, if unavailable, processes the raw train and test splits by concatenating them, generating engineered features, imputing missing values, and saving the results for reproducibility. Alongside the transformed dataset, it also extracts test identifiers and group labels required for later submission.

Building on this, prepare_data finalizes the dataset for model training. It applies feature selection, encodes categorical variables into dense integer indices suitable for embeddings, and standardizes numerical features using statistics computed only from the training set to avoid leakage. The function then returns all components needed by ranking models - feature matrices, labels, groups, feature metadata, and test references - ensuring a clean and consistent input pipeline.

9.5. Training Configuration

Initially, the train and test files were concatenated to ensure consistent feature engineering across all samples. After processing, the dataset was split back into three parts according to known group boundaries: the first 16,487,352 rows were used for training, the next segment up to 18,145,372 rows served as validation, and the remaining rows corresponded to the official test set. To generate a final submission, we adopted a two-stage approach: first, training and validating models using the train/validation split to verify generalization and avoid overfitting (deeprec_validate.py), and then retraining the best configuration on the full training set of 18,145,372 rows before applying it to the test set (deeprec_full_training.py).

This strategy combines the reliability of validation-based model selection with the advantage of exploiting the entire dataset for maximum learning capacity in the final model.

Training was carried out for 10 epochs with an initial learning rate of 1e-3, using the AdamW optimizer (with weight decay=0.01) and a linear scheduler that included a 10% warm-up phase before decaying the learning rate steadily toward zero. This setup allowed the embeddings and MLP layers to stabilize early, while the decay phase encouraged smooth convergence.

Throughout training/validation, the model was monitored on multiple ranking metrics - HitRate@3, NDCG@3, and MAP@3 - in addition to the pairwise ranking loss, to ensure both score calibration and top-k ranking quality. An EarlyStopping mechanism (patience=3, min_delta=0.001) was also employed, halting training when validation improvements plateaued.

The checkpointing strategy required two conditions to be met before saving the model:

1. the validation loss decreased
2. the HitRate@3 improved

By enforcing this dual criterion, we ensured that the saved model was not only better calibrated (lower loss) but also more effective at ranking relevant items at the top positions. This way, the best checkpoint combined robustness with ranking performance and was later reloaded for final validation predictions and test-time inference.

By contrast, when training on the full dataset (18,145,372 rows) to produce the final submission, the model is saved after each epoch whenever the training loss improves. This approach serves as a continuation of the earlier stage, where model stability was first validated on the held-out validation set.

9.6. GPU Usage

The presented model and configuration are relatively lightweight, running on an NVIDIA RTX 4500 Ada Generation GPU, and occupying - depending on the batch size - up to around 12 GB of GPU memory with a typical utilization of about 22%, power draw of ~48 W out of 210 W, and maintaining a low temperature of 39 °C.

9.7. DL Score

Upon completing both stages and producing the submission file, the model attains the following performance scores:

Public score: 0.48755

Private score: 0.49391

While its standalone performance is lower than that of GBDT models, it adds meaningful value in ensemble settings by offering a complementary perspective on the data and enriching the final prediction distribution.

10. Ensemble

To further enhance performance and improve the overall leaderboard score, I experimented with several ensemble strategies that combine the three previously trained models: XGBoost, LightGBM, and the Deep Learning ranker (DL). Each of these models has different strengths - GBDT models are highly effective at capturing structured feature interactions, while the DL model excels at learning dense representations and exploring feature spaces in a more flexible way.

10.1. Compute Confidence Score

Frist script (compute_confidence_score.py) takes model submission files (CSV or Parquet) and enriches them with a confidence score designed to quantify how reliable each prediction is within its ranking group. For every ranker_id, it first computes the group size, then assigns a base confidence proportional to the item's relative position in the ranking (items at the top get values near 1, those at the bottom near 0). To capture the intuition that top-ranked items deserve extra weight, it also computes a Reciprocal Rank Fusion (RRF) score, which decays nonlinearly with the rank according to the formula 1/(k + rank)1, where in this case k = 5 determines how quickly the score decreases for lower ranks. The two signals are then blended with a weight parameter alpha = 0.7, meaning the final confidence is 70% driven by the linear rank-based confidence and 30% by the RRF adjustment. As a result, the final confidence reflects both the overall position in the group and the top-heavy emphasis from RRF. The script processes multiple submission files in sequence, saving new versions suffixed with _with_confidence, thus producing enriched predictions that later will be exploited in ensemble strategy.

The submission files used to compute confidence scores and subsequently to derive the weights for the final blended submission are stored in the submissions folder. This directory contains both the implementation details and the actual submission files, whose names include not only the score achieved on the public leaderboard but also a time tag automatically generated at creation, ensuring clear traceability of each experiment.

The files are:

```
files = [
    "submission_20250721083807_0.52244.parquet",
    "submission_20250724032338_0.51345.parquet",
    "submission_20250725083055_0.52391.parquet",
    "submission_20250727084025_0.52795.parquet",
    "submission_20250802074816_0.52603.parquet",
    "submission_20250804001151_0.51244.parquet",
    "submission_20250807032439_0.52538.parquet",
    "submission_dl_ranker_0.48755.parquet"
]
```

10.2. Ensemble Strategy

The script constructs the final ensemble submission through a multi-step process that combines the outputs of several individual models. It begins by collecting all enriched submission files (those suffixed with _with_confidence.csv), extracting the public score from each filename, and renaming their confidence columns so that each model's contribution is uniquely identifiable. These files are then merged on ["Id", "ranker_id"], aligned row by row across models, and transformed into a standardized confidence matrix for further analysis.

Next, it runs PCA via SVD keeping k_pca = 5 components and, in the default "residual" mode, computes Spearman correlations on the PCA residuals (original standardized data minus its rank-k reconstruction). This measures how much unique signal each model contributes beyond the common shared structure. It then converts each model's average correlation into a uniqueness weight: uniqueness = 1 − mean_corr, normalized to sum to 1.

In parallel, it computes entropy weights: for each model's confidence column, it normalizes values to a probability vector (divide by the column sum, with safe fallbacks) and computes Shannon entropy; these entropies are normalized to sum to 1.

Finally, the script blends the two weightings with alpha = 0.75: final_weight = 0.75 * uniqueness_weight + 0.25 * entropy_weight, and prints the resulting per-model weights.

Final ensemble weights (uniqueness + entropy):

confidence_0.51345: 0.1270

confidence_0.52603: 0.1215

confidence_0.52391: 0.1242

confidence_0.51244: 0.1209

confidence_0.48755: 0.1251

confidence_0.52795: 0.1303

confidence_0.52538: 0.1235

confidence_0.52244: 0.1275

With the weights set, it computes an ensemble_confidence as the weighted sum of the (unnormalized) per-model confidence columns (missing values treated as 0), then ranks items within each ranker_id by descending ensemble confidence using an ordinal rank to produce the selected positions. The script finishes by selecting ["Id", "ranker_id", "selected"] and saving the final blended submission to submission_ensemble.parquet, also printing a small sample to verify the output.

10.3. Ensemble Score

Through the application of the previously outlined ensemble strategy, the final submission file achieved the following performance scores:

Public score: 0.53438

Private score: 0.53672

Additionally, a combination that performed weaker on the public leaderboard but yielded better results on the private leaderboard is:

compute confidence: alpha = 0.7, k = 5

k_pca = 5

spearman_mode = "loadings"

alpha(spearman/entropy)=0.75

Final ensemble weights (uniqueness + entropy):

confidence_0.51244: 0.1828

confidence_0.52795: 0.1057

confidence_0.52603: 0.1021

confidence_0.52538: 0.1075

confidence_0.48755: 0.1272

confidence_0.51345: 0.1703

confidence_0.52244: 0.1021

confidence_0.52391: 0.1021

Public score: 0.53209

Private score: 0.53726

11. Interesting Findings and Suggestions

During experimentation, several observations stood out. First, tree-based models such as XGBoost remained surprisingly dominant despite extensive efforts with deep neural rankers. While neural architectures offered complementary signals for ensembling, their standalone performance lagged, reinforcing the importance of feature-rich GBDT pipelines in industrial recommendation.

Second, the ensemble analysis revealed that models with lower public scores could still contribute valuable unique signal on the private leaderboard. This suggests that leaderboard-driven model selection can be misleading, and diversity of predictors is often more important than individual strength.

Third, route- and company-level statistics consistently ranked among the most informative features, highlighting the role of domain-specific priors. By contrast, some highly engineered combinations of ranks or cabin indicators proved less stable, underscoring the need to balance complexity with robustness.

Based on these findings, we suggest that future work explore hybrid strategies that integrate lightweight neural modules with GBDT backbones, rather than treating them as competing alternatives. Additionally, methods for uncertainty estimation and calibration could improve the ensemble weighting process, leading to more reliable gains. Finally, extending the dataset with richer temporal signals (e.g., seasonality, holidays) or external knowledge sources (e.g., historical delay patterns, macroeconomic indicators) may open new avenues for boosting predictive performance in real-world deployment.

12. Conclusion

This solution demonstrated that carefully engineered features combined with robust tree-based models can achieve strong performance in the AeroClub RecSys 2025 challenge. By systematically exploring company, route, user, and temporal signals, we were able to capture diverse aspects of flight choice and translate them into effective predictors. Ensemble strategies further highlighted the value of model diversity, showing that even weaker individual models can strengthen the overall ranking system when combined thoughtfully.

Looking forward, we see potential in hybrid approaches that merge the interpretability and stability of gradient-boosted trees with the representational flexibility of neural models. Beyond algorithmic choices, enriching the dataset with external contextual information may provide the next leap in performance. Overall, the competition underlined the importance of domain-aware feature design, careful model selection, and ensemble learning as key ingredients for building high-quality recommendation systems in practice.

13. References

[1] - „Which Tricks Are Important for Learning to Rank?” by Ivan Lyzhin, Aleksei Ustimenko, Andrey Gulin, Liudmila Prokhorenkova - https://arxiv.org/pdf/2204.01500

[2] - „Industry Insights from Comparing Deep Learning and GBDT Models for E-Commerce Learning-to-Rank” by Yunus Lutz, Timo Wilm, Philipp Duew - https://arxiv.org/pdf/2507.20753

[3] - „FiBiNET: Combining Feature Importance and Bilinear feature Interaction for Click-Through Rate Prediction” by Tongwen Huang, Zhiqi Zhang, Junlin Zhang - https://arxiv.org/pdf/1905.09433

[4] - „FiBiNet++: Reducing Model Size by Low Rank Feature Interaction Layer for CTR Prediction” by Pengtao Zhang, Zheng Zheng, Junlin Zhang - https://arxiv.org/pdf/2209.05016

[5] - „Factorization Machines for Datasets with Implicit Feedback” by Babak Loni, Martha Larson, Alan Hanjalic - https://arxiv.org/pdf/1812.08254

[6] - „Neural Collaborative Ranking” by Bo Song, Xin Yang, Yi Cao, Congfu Xu - https://arxiv.org/pdf/1808.04957