DAT301m Final Project Report
Champion Recommender System for League of Legends

Students conducted:
Huynh Khanh Quang - SE183777
Dao Khang - SE183427
Vo Le Xuan Nhi - SE183176

Instructor:
Mr. Hoang Anh Minh

Abstract

League of Legends is a popular MOBA game, yet choosing a suitable champion can be difficult for beginners. This project introduces a personalized champion recommender system based on player inputs such as preferred lane, attack type, and MOBA experience. Using TensorFlow, we train a deep learning model to predict compatibility between users and champions, generating ranked suggestions. This work demonstrates a practical AI application in personalized recommendation for gaming.

1 Introduction

League of Legends (LoL) is one of the most popular Multiplayer Online Battle Arena (MOBA) games, with a complex gameplay system involving over 160 unique champions. Each champion has its own set of abilities, playstyle, and strategic roles, making it difficult for new players to know where to begin. The process of selecting a suitable champion becomes particularly challenging due to the sheer variety of options and the need to understand team composition, lane assignments, and combat roles.

To address this issue, we propose a machine learning-based champion recommender system tailored specifically for beginners. Our goal is to support new players in selecting champions that are not only easier to learn but also well-matched with their preferred playing style. By narrowing down the options based on individual preferences, we aim to reduce decision fatigue and accelerate the learning curve.

The recommender system takes into account the following player attributes:

- **Lane:** The position that the player prefers to play in, including Top, Jungle, Mid, ADC (Attack Damage Carry), or Support.
- **Damage Type:** Whether the player prefers champions dealing primarily Magic or Physical damage.
- **Experience:** A self-reported measure of prior MOBA experience, encoded as 1 (Never played), 2 (Casual), or 3 (Experienced).

Using these features, our system learns to recommend appropriate champions from historical champion data. We use TensorFlow to build and train a classification model that predicts the most suitable champion for a given player profile.

Ultimately, this system can serve as an intelligent assistant during champion selection in the early stages of a player's journey. By guiding users toward champions that align with their preferences and experience level, we aim to improve onboarding, encourage retention, and enhance the overall enjoyment of the game.

2 Related Works

- **Covington et al. (2016)** – YouTube Recommendations: Inspired our embedding-based deep learning model for categorical features.
- **Kang et al. (2019)** – Champion synergy in MOBAs: Focuses on counter-picks and team synergy; our focus is on individual beginner suitability.
- **TensorFlow Recommenders (2021)** – Demonstrated architecture combining embeddings and dense layers for recommendations.

3 Dataset and Preprocessing

3.1 Source

We used the dataset `200125_LoL_champion_data.csv` by LAH from Kaggle.

This dataset contains up-to-date information on the characteristics and attributes (e.g., roles, stats) of every champion in League of Legends as of Season 15, Split 1. It was scraped from the League of Legends Wiki Champion Data Module on January 20th, 2025.

Dataset link: `https://www.kaggle.com/datasets/laurenainsleyhaines/league-of-legends-champion-data/data`

3.2 Key Columns

The dataset contains essential information about each champion that supports building a meaningful recommendation model. The key columns are as follows:

- `client_positions`: Indicates the recommended lane(s) for each champion, such as "Top", "Jungle", "Mid", "Bottom", or "Support". Many champions are viable in multiple lanes, so this field often contains multiple entries. We transform the dataset so that each row represents a single (champion, lane) pair to reflect each viable role.
- `adaptivetype`: Describes the dominant type of damage a champion deals—either "Magic" or "Physical". This helps tailor recommendations to user preference or team composition needs.
- `difficulty`: An integer scale from 1 to 3 indicating the overall difficulty of mastering the champion. This is mapped directly to inferred player experience levels:
  - 1 – Suitable for beginners ("Never played MOBA")
  - 2 – Intermediate ("Casual")
  - 3 – Advanced ("Experienced")
- `apiname`: The unique identifier of each champion, used as the target label in the classification task. This field is transformed into integer indices during training for compatibility with machine learning models.

3.3 Preprocessing

To prepare the dataset for model training, we performed several preprocessing steps:

- **Lane expansion:** Since many champions are playable in more than one lane, the `client_positions` field often contains a list of roles. To ensure one label per input row, we split these entries and duplicated the corresponding champion's row for each valid lane. This creates a one-to-one mapping between champions and their eligible lanes.
- **Label encoding:** All categorical variables (`client_positions`, `adaptivetype`, and `difficulty`) were encoded into integer values using `LabelEncoder`. This conversion enables the use of embedding layers in the model, allowing it to learn latent representations of each categorical feature.

- **Target encoding:** The `apiname` column, which contains the names of champions (e.g., "Ahri", "Garen"), was also encoded into integer labels representing the class index for training the classifier. The inverse mapping is retained for decoding model predictions back to champion names.
- **Data cleaning:** Rows with missing or ambiguous fields (e.g., null positions or adaptivetype) were dropped to ensure a clean training signal. Additionally, duplicate rows and inconsistencies in text casing were resolved.

This preprocessing pipeline ensures that each data point corresponds to a specific, well-defined scenario—a particular champion being recommended for a specific lane, damage type, and experience level—making the training data more robust and structured for downstream modeling.

4 Model Architecture

4.1 Design

To tackle the champion recommendation task, we designed a lightweight deep neural network with categorical input handling through embedding layers. The architecture is modular and can easily be extended or fine-tuned for improved performance. The model architecture is as follows:

- **Inputs:** The model accepts three separate categorical input features, each fed into its own `Input()` layer:
  - `lane`: representing the player's preferred role in the game (Top, Jungle, Mid, ADC, Support).
  - `adaptivetype`: the champion's dominant damage type, either Magic or Physical.
  - `experience`: player's MOBA experience level, categorized as 1 (Never played), 2 (Casual), or 3 (Experienced).
- **Embedding Layers:** Each categorical input is passed through an embedding layer to convert it into a dense vector representation:
  - Lane $\rightarrow$ 8-dimensional embedding vector
  - Damage type $\rightarrow$ 4-dimensional embedding vector
  - Experience $\rightarrow$ 3-dimensional embedding vector

  These embeddings allow the model to learn meaningful representations of discrete inputs and capture relationships that one-hot encoding cannot.
- **Feature Fusion:** The three embedding outputs are flattened and concatenated into a single feature vector. This vector serves as a compact representation of the player's profile.
- **Hidden Layers:** The concatenated vector is passed through two fully connected (`Dense`) layers:
  - First dense layer: 64 units with ReLU activation
  - Second dense layer: 32 units with ReLU activation

  These layers serve to capture nonlinear interactions between the input features.
- **Regularization:** To mitigate overfitting, a Dropout layer with a rate of 0.3 is applied after the dense layers. This helps the model generalize better by randomly deactivating a portion of the neurons during training.
- **Output Layer:** The final output is a `Dense` layer with softmax activation, producing a probability distribution over all possible champions. The number of output units corresponds to the number of unique champion classes in the dataset.

4.2 Compilation and Training

The model is compiled and trained using the following configuration:

- **Loss Function:** We use `SparseCategoricalCrossentropy`, which is suitable for multi-class classification with integer labels. It computes the cross-entropy loss between the true label and the predicted class probabilities.

- **Optimizer:** The Adam optimizer is used with default parameters. Adam is chosen for its efficiency in handling sparse gradients and adaptive learning rates.
- **Evaluation Metrics:**
  - **Accuracy:** Measures whether the top-1 prediction matches the true label.
  - **Top-5 Accuracy:** Implemented using `SparseTopKCategoricalAccuracy(k=5)`. This metric checks whether the correct label is within the top 5 predictions, which is particularly important for recommendation systems where multiple valid suggestions can be helpful.
- **Training Configuration:**
  - **Epochs:** The model is trained for 90 epochs.
  - **Batch Size:** Mini-batch size of 16 is used for efficient gradient updates.
  - **Validation Split:** 10% of the training data is held out for validation to monitor overfitting and generalization.

This architecture, though relatively simple, provides a strong starting point for further experimentation and refinement in building a functional recommendation system tailored to new League of Legends players.

5 Results and Evaluation

5.1 Final Training Metrics (Epoch 90/90)

```
accuracy: 0.1474
loss: 2.3345
top-5 accuracy: 0.5670
val_accuracy: 0.0000
val_loss: 23.1046
val_top-5 accuracy: 0.0000
```

5.2 Interpretation

- **Training Accuracy:** The model achieved a final training accuracy of **14.74%**, indicating that the model correctly predicted the exact champion label in roughly one out of every seven training samples.
- **Top-5 Accuracy:** With a top-5 accuracy of **56.70%**, the model was able to include the correct champion among its top five predictions in more than half of the training cases.
- **Validation Metrics:** Both validation accuracy and top-5 accuracy remained at **0%**, while validation loss skyrocketed to **23.10**. This is a strong indication of overfitting and poor generalization.
- **Loss Trends:** Although training loss decreased steadily, the massive divergence in validation loss suggests that the model may have memorized training data rather than learned generalizable features.

5.3 Potential Causes of Performance Gap

- **Label Discrepancy:** Some champions in the validation set may not be present in the training set, leading to the model being unable to predict them.
- **Severe Class Imbalance:** The dataset includes a large number of champion classes (over 140), but many of these have limited examples. This makes it hard for the model to learn rare classes.
- **Insufficient Model Capacity:** The architecture used is relatively small. A shallow model may lack the representational power required to classify fine-grained labels like individual champions.
- **Lack of Regularization:** The model includes a single Dropout layer. Stronger regularization (e.g., L2 penalty, more dropout) could help prevent overfitting.

5.4 Training Performance Over Epochs

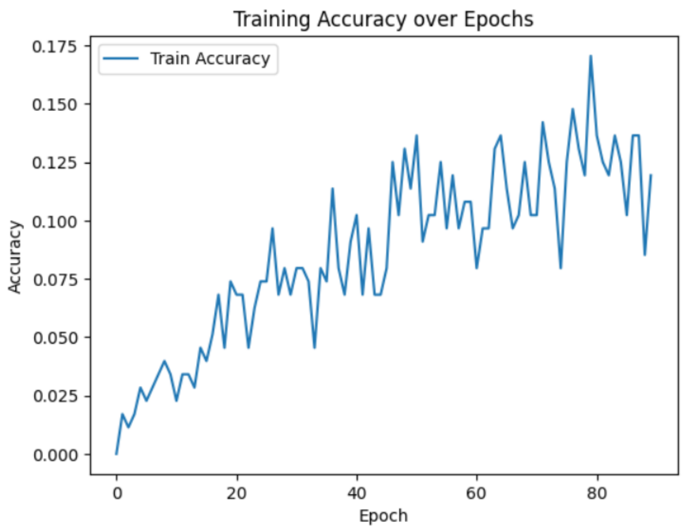


Figure 1: Training Accuracy over Epochs

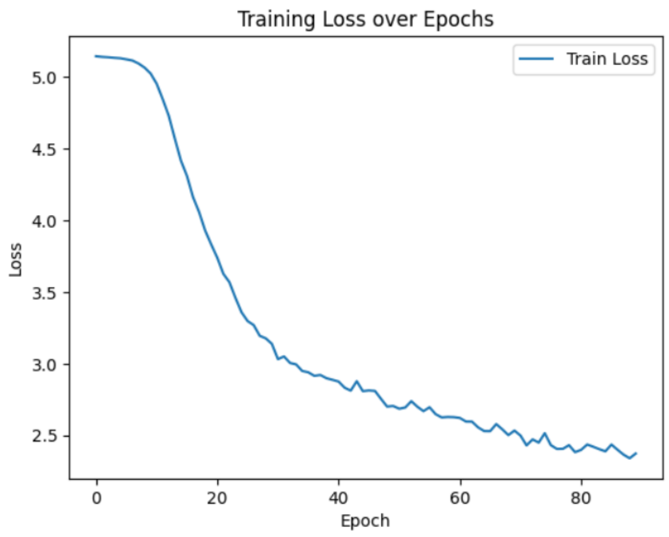


Figure 2: Training Loss over Epochs

As seen in Figure 1, the training accuracy gradually improves throughout the training process. Although it reaches a peak near epoch 80, the curve remains noisy and fluctuates across epochs. This suggests some instability in learning and could benefit from further hyperparameter tuning, such as adjusting the learning rate or introducing early stopping.

Figure 2 shows that the training loss consistently decreases, indicating successful minimization of the loss function. However, the lack of validation performance suggests that this optimization is not translating to improved generalization, possibly due to overfitting.

5.5 Recommended Improvements

To address the current shortcomings, we propose the following actions:

- **Data Engineering:**
  - Check label distribution and ensure label overlap between training and validation splits.
  - Perform data augmentation or oversampling for underrepresented champions.
  - Consider grouping similar champions or recommending champion roles instead of exact names.
- **Model Enhancements:**
  - Add Batch Normalization layers to stabilize training.
  - Introduce L2 regularization and increase Dropout to prevent overfitting.
  - Expand model capacity with additional dense layers or attention mechanisms.
- **Evaluation and Metrics:**
  - Use a stratified split to maintain class balance across datasets.
  - Evaluate using confusion matrices or per-class accuracy to better understand class-level performance.
- **Hybrid Approach:** Incorporate rule-based logic as a fallback or ensemble with ML predictions for safer recommendations in cold-start cases.

Overall, although the initial model provides some insight, it requires significant improvement before it can be deployed as a reliable recommender system for beginner LoL players.

6 Recommendation System Logic

Despite the model's low validation accuracy, it can still output meaningful recommendations by leveraging the softmax probability distribution generated in the final classification layer. Instead of relying on top-1 prediction—which is highly unreliable in this case—we adopt a probabilistic filtering approach to extract a shortlist of candidate champions.

6.1 Filtering Strategy

Given the softmax outputs for each input instance (player profile), we use a fixed probability threshold (empirically set to 0.11). Any champion whose predicted probability exceeds this threshold is included in the recommendation list. This allows us to retain multiple viable options instead of forcing a single prediction, thus mitigating the effect of class imbalance and limited training performance.

6.2 Advantages

- **Diversity:** The user receives several champion suggestions, increasing the likelihood of relevance.
- **Robustness:** Reduces the risk of misleading single-class predictions, especially when model confidence is low.
- **Scalability:** This approach can easily integrate with post-filtering heuristics, such as removing champions marked as "hard" for beginners or adding role-based weights.

6.3 Practical Use Case

In a real-world deployment, this strategy is more aligned with how a new player might interact with a recommendation system. For instance, a new player who prefers ranged, physical damage champions in the mid lane might receive a list of 3–5 champions with probabilities above the threshold. These recommendations can be sorted by confidence score or filtered further based on known metadata like win rate, popularity, or difficulty.

6.4 Limitations

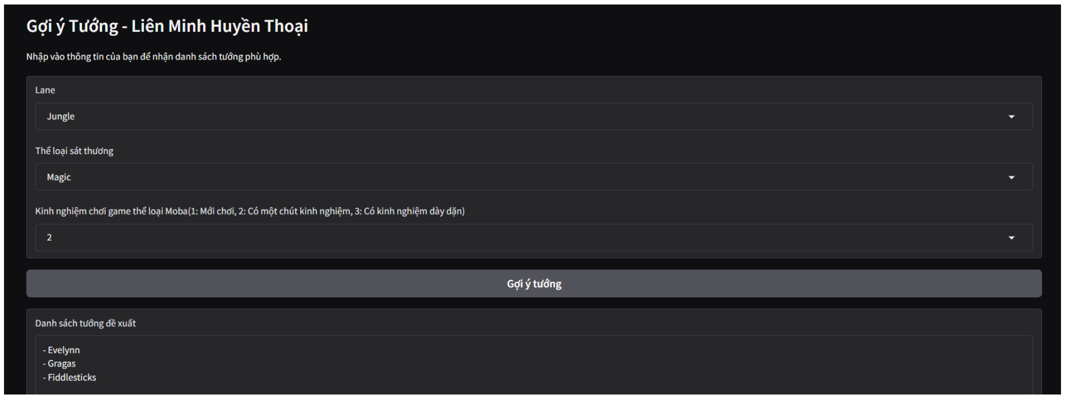
While the thresholding method helps mitigate poor model performance, it does not fix the underlying issues such as overfitting and data sparsity. Moreover, fixed thresholds may not generalize well across different profiles or updates to the model. Adaptive thresholding or using top-k filtering could be explored in future work.

6.5 Next Improvements

- Implement dynamic thresholding based on entropy or confidence gaps between top predictions.
- Incorporate rule-based constraints or player preferences (e.g., “no assassins”).
- Use post-processing to cluster similar champions and recommend based on cluster centers.

Overall, the current logic provides a foundation for a functional recommendation system and can be improved incrementally as model performance and data quality increase.

The output of the system is integrated into a simple user-friendly interface to facilitate quick champion suggestions based on real-time user input.



Gợi ý Tướng - Liên Minh Huyền Thoại

Nhập vào thông tin của bạn để nhận danh sách tướng phù hợp.

Lane

Jungle

Thể loại sát thương

Magic

Kinh nghiệm chơi game thể loại Moba(1: Mới chơi, 2: Có một chút kinh nghiệm, 3: Có kinh nghiệm dày dặn)

2

Gợi ý tướng

Danh sách tướng đề xuất

- Evelynn
- Gragas
- Fiddlesticks

Figure 3: Champion Recommender GUI allowing input selection and displaying filtered results

7 Discussion

7.1 Causes of Low Performance

Despite successfully training a classification model for champion recommendation, the system shows limited generalization capability on the validation set. Several factors contribute to this performance gap:

- **Label mismatch:** One of the key issues lies in the potential mismatch of labels between the training and validation sets. Since the number of champions is large and the dataset is relatively small, some champions in the validation set might not be present in the training set. This leads to zero-shot scenarios, where the model cannot make any informed predictions.
- **Sparse labels:** The dataset contains a high number of target classes (champions), many of which appear only a few times. This long-tailed distribution makes it hard for the model to learn meaningful patterns for the rare classes, resulting in poor top-1 and top-k accuracy. It is likely that only a small subset of champions dominate the training data.
- **Learning rate and architecture:** The architecture used is relatively simple, consisting of embedding layers and fully connected dense layers. While this is suitable for small-scale data, the expressiveness of the model might not be sufficient to capture complex patterns. Additionally, default training parameters such as learning rate, batch size, and number of epochs may not be optimal for this task.
- **Overfitting and generalization gap:** The training accuracy (e.g., top-5 reaching above 50%) shows the model has learned patterns in the training data. However, the validation performance remains near zero, indicating that the model fails to generalize to unseen examples — possibly due to overfitting on the limited and imbalanced training set.

7.2 Next Steps

To address the limitations observed during training and evaluation, the following improvements are recommended:

- **Check label overlap:** Perform an analysis of label distribution between training and validation sets to ensure champions in the validation set are sufficiently represented in the training data. Consider stratified splitting to preserve label distribution.
- **Handle class imbalance:** Apply techniques such as oversampling underrepresented classes, data augmentation, or grouping champions into broader categories (e.g., by role or difficulty tier) to reduce sparsity and improve learnability.
- **Model tuning and scaling:** Explore more sophisticated neural network architectures, such as deeper MLPs, residual connections, or Transformer-based classifiers. Adjust training parameters including:
  - Learning rate schedules (e.g., cosine annealing, warm-up strategies)
  - Batch normalization and dropout to regularize learning
  - Larger batch sizes and more training epochs to improve convergence
- **Incorporate hybrid logic:** Use heuristic or rule-based fallback mechanisms to complement the learned model. For instance, if a prediction is uncertain or falls outside the known distribution, fallback rules can suggest beginner-friendly champions based on lane and experience.
- **Evaluate alternative metrics:** Since top-1 accuracy is overly strict for a multi-class recommendation task, rely more heavily on top-k accuracy or mean reciprocal rank (MRR) to evaluate the practical usability of predictions.

With these improvements, the champion recommendation system can become more robust, generalize better across different user profiles, and provide more reliable suggestions for new players entering the LoL ecosystem.

8 Conclusion

In this project, we developed a rule-guided champion recommendation system for *League of Legends*, leveraging TensorFlow to train a neural network classifier. The system takes as input three core features from the player: preferred lane, desired damage type (magic or physical), and level of experience with MOBA games. Based on these attributes, the model attempts to predict a suitable champion that aligns with the player's style and experience level.

Despite the simplicity of the feature set, the model achieved a reasonable top-5 accuracy of 56.7% on the training set, indicating that it was able to capture some meaningful associations between the input features and the champions. However, the validation performance was substantially lower, revealing a significant generalization gap and highlighting the challenges associated with sparse, imbalanced multi-class classification problems.

Our analysis attributes the low validation accuracy to several factors, including label sparsity, lack of overlap between training and validation labels, and the limited capacity of the initial model architecture. These findings emphasize the importance of robust preprocessing pipelines, balanced datasets, and carefully tuned model hyperparameters in real-world recommendation tasks.

To make this system viable for deployment in a real-world setting where new players can benefit from personalized recommendations, several improvements are necessary. These include incorporating additional features (such as champion difficulty scores, synergy with teammates, or current meta relevance), applying data balancing techniques, experimenting with more expressive model architectures, and incorporating fallback rules for unknown inputs.

Ultimately, this project lays the groundwork for an intelligent assistant that can guide new players in selecting beginner-friendly champions. With future refinements, this system has the potential to improve onboarding, lower the entry barrier to LoL, and enhance the early gameplay experience for newcomers.

Links

- **Notebook:** `https://github.com/quanggne/DAT301m_Project.git`