International Journal of Computer Science and Information Technology
ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 8, Number 1, Year 2026
DOI: https://doi.org/10.62051/ijcsit.v8n1.02
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Research on Integrating Artificial Intelligence in User Feedback Loops for Dynamically Adjusting Video Game Difficulty

Hongyu Wu

United World College of South East Asia, East Campus, Singapore

ABSTRACT

Dynamic Difficulty Adjustment (DDA) remains one of the most promising mechanisms for optimizing player engagement in interactive digital entertainment. This research presents an end-to-end framework for integrating artificial intelligence into the player feedback loop to adjust game difficulty dynamically. Current similar models of DDA are usually requiring manual setup for controlling the in-game resources, making the capital cost high to deploy such a model. In this research, a functional prototype is deployed and tested in "Gulltovia", a mobile card game launched on Appstore. We design a data pipeline that parses raw player event logs from the game, computes session-level metrics such as fail count, adjusted playtime, and button interaction frequency, and uses these as inputs to a hybrid classification system that combines rule-based thresholds with a machine learning model. The trained model is exported to ONNX and executed inside the Unity engine, enabling real-time predictions and conservative difficulty adjustments based on probability confidence.

KEYWORDS

Dynamic Difficulty Adjustment; Player Modeling; Implicit Feedback; Machine Learning; Unity Integration

1. INTRODUCTION

Player experience is highly sensitive to difficulty calibration. If the challenge level is too low, skilled players quickly become bored, whereas if it is too high, novice players may become frustrated and abandon the game altogether. Game designers have long attempted to find a "sweet spot" of difficulty that induces what Csikszentmihalyi calls the flow state [1]: a psychological zone where challenge and skill are in balance.

Traditional approaches to balancing difficulty rely on static tuning and playtesting, which cannot adapt to the diverse distribution of player abilities in live service games. This gap has led to increasing interest in Dynamic Difficulty Adjustment (DDA), in which the game system measures player performance in real time and adjusts challenge parameters to maintain engagement.

Prior research has investigated DDA through rule-based systems and reinforcement learning controllers. For instance, Hunicke [2] formalized the idea of "drama management" in games, and Zoeller [3] demonstrated the use of telemetry data to identify underperforming player cohorts. However, many of these approaches are either too unfocused, resulting in perceptible difficulty swings, or too computationally expensive to deploy in production. Our work aims to strike a balance by using lightweight feature extraction, interpretable models, and a conservative control policy that preserves fairness.

This research reports on the development, implementation, and evaluation of an AI-driven DDA pipeline. The contributions of this work are threefold: (1) a reproducible data processing pipeline for

computing per-session skill indicators from raw event logs, (2) a 1D Convolution neural network with calibrated probability outputs that predict player skill levels, and (3) a feedback controller that integrates with the Unity game engine to adjust difficulty parameters in real time under strict stability constraints.

2. METHODOLOGY

2.1. Dataset and Preprocessing

The dataset used in this research (Figure 1) comprises a CSV file with 59 columns of event types, including #user_id (unique user id for each downloaded player), #event_name (coding name for every interaction with the game), #event_time (the time stamp that certain event happens in GMT+0), and, where applicable, #score or #final_score (game score).

The first step is to sort the dataset by #user_id and timestamp to reconstruct chronological player histories. Sessions are defined as contiguous intervals between #game_start (when the game opens) and #game_end (when the user quits the app) events, with any idle gap greater than 30 minutes treated as a session boundary.

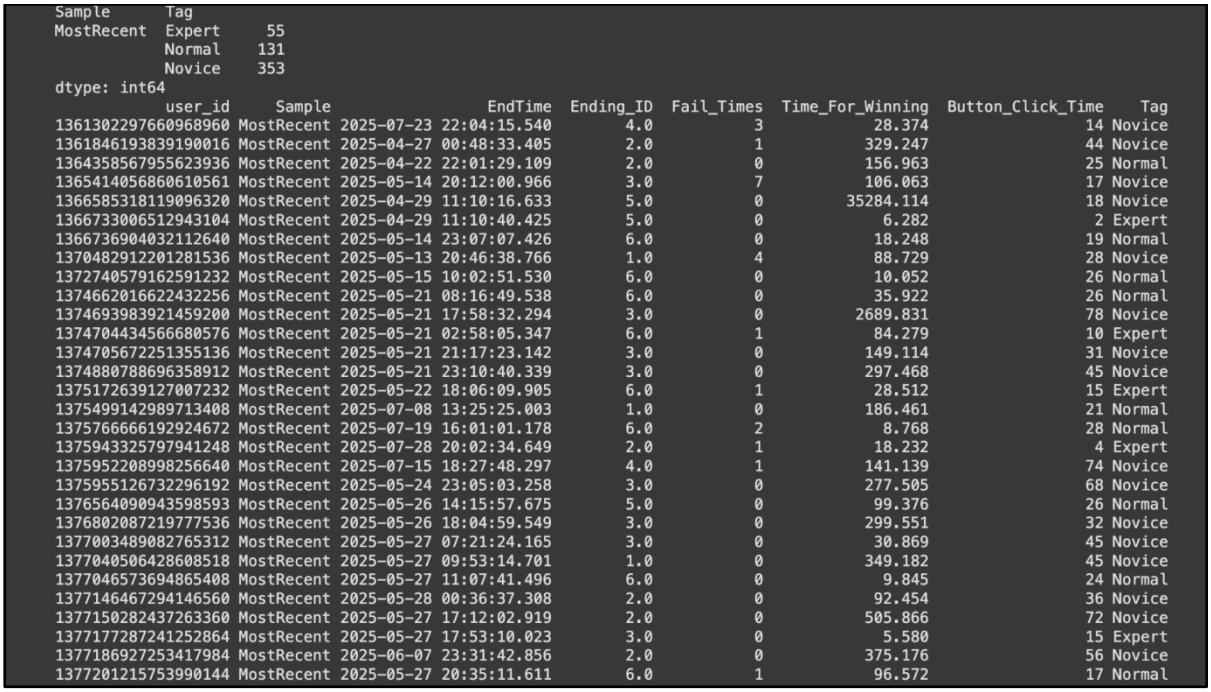
	#user_id	#event_name	#event_time	#account_id	#distinct_id	#server_time	#kafka_offset	#uuid	#dw_create_time	#dw_update_time
0	1395325891369799680	ta_app_install	2025-07-16 20:46:54.388	NaN	00089CB7-8597-44C8-AA48-F2E8BB9A737B	2025-07-17 08:46:56.269	51118016318	C1275F9A-515C-4D5C-BA8E-F727539078E2	2025-07-17 08:46:58.925	NaN
1	1395325891369799680	first_active	2025-07-16 20:46:55.415	NaN	00089CB7-8597-44C8-AA48-F2E8BB9A737B	2025-07-17 08:46:56.272	51118016307	8C59CC35-659D-4554-B368-CBC333414384	2025-07-17 08:46:57.431	NaN
2	1395325891369799680	app_start	2025-07-16 20:46:55.418	NaN	00089CB7-8597-44C8-AA48-F2E8BB9A737B	2025-07-17 08:46:56.272	51118016307	76846DE9-0582-4CD5-8679-02F2ED250F87	2025-07-17 08:46:57.216	NaN
3	1395325891369799680	app_launch	2025-07-16 20:46:55.418	NaN	00089CB7-8597-44C8-AA48-F2E8BB9A737B	2025-07-17 08:46:56.272	51118016307	1620211D-7660-4F81-9814-DFA77393EE62	2025-07-17 08:46:57.514	NaN
4	1395325891369799680	ta_app_start	2025-07-16 20:46:55.611	NaN	00089CB7-8597-44C8-AA48-F2E8BB9A737B	2025-07-17 08:46:56.272	51118016307	79DE5A45-F23A-4CF2-8AD6-0AE6227C48FB	2025-07-17 08:46:57.196	NaN

Figure 1. Raw Event Table

Mathematically, if t_s (#game_start) and t_e (#game_end) denote the time stamps of the start and end of the session respectively, and t_{qi} (where the app quits systematically, not in sleeping), t_{li} (game begins systematically, not when user open the app) denote pairs of app quit and launch times during the session, the true playtime is computed as

$$T_{\text{play}} = (t_e - t_s) - \sum_{i=1}^{n} (t_{q_i} - t_{l_i})$$

Where n is the number of quit to launch intervals. This adjustment ensures that background time does not artificially inflate playtime. The examined table is below (Figure 2).



```
Sample      Tag
MostRecent  Expert     55
            Normal    131
            Novice    353
dtype: int64
              user_id     Sample                 EndTime  Ending_ID  Fail_Times  Time_For_Winning  Button_Click_Time     Tag
1361302297660968960 MostRecent 2025-07-23 22:04:15.540        4.0           3            28.374                 14 Novice
1361846193839190016 MostRecent 2025-04-27 00:48:33.405        2.0           1           329.247                 44 Novice
1364358567955623936 MostRecent 2025-04-22 22:01:29.109        2.0           0           156.963                 25 Normal
1365414056860610561 MostRecent 2025-05-14 20:12:00.966        3.0           7           106.063                 17 Novice
1366585318119096320 MostRecent 2025-04-29 11:10:16.633        5.0           0         35284.114                 18 Novice
1366733006512943104 MostRecent 2025-04-29 11:10:40.425        5.0           0             6.282                  2 Expert
1366736904032112640 MostRecent 2025-05-14 23:07:07.426        6.0           0            18.248                 19 Normal
1370482912201281536 MostRecent 2025-05-13 20:46:38.766        1.0           4            88.729                 28 Novice
1372740579162591232 MostRecent 2025-05-15 10:02:51.530        6.0           0            10.052                 26 Normal
1374662016622432256 MostRecent 2025-05-21 08:16:49.538        6.0           0            35.922                 26 Normal
1374693983921459200 MostRecent 2025-05-21 17:58:32.294        3.0           0          2689.831                 78 Novice
1374704434566680576 MostRecent 2025-05-21 02:58:05.347        6.0           1            84.279                 10 Expert
1374705672251355136 MostRecent 2025-05-21 21:17:23.142        3.0           0           149.114                 31 Novice
1374880788696358912 MostRecent 2025-05-21 23:10:40.339        3.0           0           297.468                 45 Novice
1375172639127007232 MostRecent 2025-05-22 18:06:09.905        6.0           1            28.512                 15 Expert
1375499142989713408 MostRecent 2025-07-08 13:25:25.003        1.0           0           186.461                 21 Normal
1375766666192924672 MostRecent 2025-07-19 16:01:01.178        6.0           2             8.768                 28 Normal
1375943325797941248 MostRecent 2025-07-28 20:02:34.649        2.0           1            18.232                  4 Expert
1375952208998256640 MostRecent 2025-07-15 18:27:48.297        4.0           1           141.139                 74 Novice
1375955126732296192 MostRecent 2025-05-24 23:05:03.258        3.0           0           277.505                 68 Novice
1376564090943598593 MostRecent 2025-05-26 14:15:57.675        5.0           0            99.376                 26 Normal
1376802087219777536 MostRecent 2025-05-26 18:04:59.549        3.0           0           299.551                 32 Novice
1377003489082765312 MostRecent 2025-05-27 07:21:24.165        3.0           0            30.869                 45 Novice
1377040506428608518 MostRecent 2025-05-27 09:53:14.701        1.0           0           349.182                 45 Novice
1377046573694865408 MostRecent 2025-05-27 11:07:41.496        6.0           0             9.845                 24 Normal
1377146467294146560 MostRecent 2025-05-28 00:36:37.308        2.0           0            92.454                 36 Novice
1377150282437263360 MostRecent 2025-05-27 17:12:02.919        2.0           0           505.866                 72 Novice
1377177287241252864 MostRecent 2025-05-27 17:53:10.023        3.0           0             5.580                 15 Expert
1377186927253417984 MostRecent 2025-06-07 23:31:42.856        2.0           0           375.176                 56 Novice
1377201215753990144 MostRecent 2025-05-27 20:35:11.611        6.0           1            96.572                 17 Normal
```

Figure 2. Processed Event Table

2.2. Feature Engineering

For each session, we compute three main metrics: (1) #Fail_Times, the number of failed endings (where #ending_id is in the failure set {1,2,3}); (2) #Time_For_Winning, equal to T_{play} when the session ends with a win; and (3) #Button_Click_Time, #counting card_select events as a proxy for interaction intensity. To smooth noise, especially for players with very short sessions, we apply exponential weighting across the most recent k sessions so that more recent performance has higher influence.

2.3. Classification Strategy

The baseline classifier uses deterministic rules: if T_{play} is smaller than 150 seconds, and #Button_Click_Time less than 15, the player is labeled Expert. If these thresholds are slightly exceeded but still within defined limits (≤2 fails, ≤400 seconds, ≤30 clicks), the label is Normal. All other cases are classified as Novice. This boundary is set by the game studio who produced this mobile game —with us being the major programmer and designer— and as a result of the data analysis of over 2000 players' statistics.

2.4. Model Training

```
class Conv1x1Model(nn.Module):
    def __init__(self):
        super(Conv1x1Model, self).__init__()
        self.conv1x1 = nn.Conv2d(3, 64, kernel_size=1) #Conv1d
        self.fc = nn.Linear(64, 3)

    def forward(self, x):
        x = self.conv1x1(x) #multiple layers
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

Figure 3. Initial Model (Part)

```
class MLP(nn.Module):
    def __init__(self, in_dim=3, hidden1=64, hidden2=32, out_dim=3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(in_dim, hidden1),
            nn.ReLU(),
            nn.BatchNorm1d(hidden1),
            nn.Dropout(0.15),
            nn.Linear(hidden1, hidden2),
            nn.ReLU(),
            nn.BatchNorm1d(hidden2),
            nn.Dropout(0.10),
            nn.Linear(hidden2, out_dim)
        )
    def forward(self, x):
        return self.net(x)

model = MLP(in_dim=3, out_dim=num_classes)
```

Figure 4. Final Model (Part)

The model architecture was selected based on its ability to efficiently capture temporal dependencies between player metrics while remaining lightweight for mobile deployment. A one-dimensional convolutional network (Conv1D) was chosen because it is well-suited to sequential data with limited

feature dimensions, such as session-based gameplay telemetry. Unlike traditional multilayer perceptrons that treat inputs as independent, the Conv1D layer can detect localized patterns in player behavior—such as repeated fail–recover cycles—through its kernel sliding operation, allowing the network to learn temporal correlations between fail times, completion durations, and interaction frequencies. This design follows prior work demonstrating that compact convolutional architectures outperform fully connected ones for low-dimensional sequential tasks [4].

The specific hyperparameters were selected to balance generalization and computational efficiency. Hidden layers of 64 and 32 neurons provide sufficient representational capacity without inflating inference cost. Dropout rates of 0.15 and 0.10, following Srivastava et al. (2014) [5], mitigate overfitting by stochastically deactivating neurons during training, while Batch Normalization [5: Ioffe & Szegedy, 2015] stabilizes gradient flow and accelerates convergence.

Rectified Linear Units (ReLU) were employed for their sparsity and efficiency on mobile processors. The model was optimized using the Adam algorithm [7] with a learning rate of 1×10^{-3}, which provides adaptive gradient scaling and fast convergence for small datasets. Compared to a simple MLP baseline, the Conv1D model demonstrated faster convergence and superior accuracy, confirming that capturing local sequential dependencies is beneficial even in compact behavioral datasets.

2.5. Integration with Unity

The trained model is exported to ONNX format for deployment inside Unity. At runtime, each completed session triggers feature computation, which is then passed to the ONNX model through Unity's Barracuda inference engine. The predicted probability vector $p = [p_{\text{novice}}, p_{\text{normal}}, p_{\text{expert}}]$ is used to update a rolling player profile. Difficulty parameters, such as enemy health multiplier and resource drop rate, are then adjusted if the maximum class probability exceeds a confidence threshold. Adjustments are rate-limited to a maximum change of Δ per session and can be rolled back if engagement metrics decline.

3. RESULTS AND DISCUSSION

3.1. Offline Evaluation

Training loss decreased from 0.64 to 0.20 within 100 epochs, converging around epoch 35, indicating that the model quickly reached a stable minimum. The final test accuracy is 97.65%. The confusion matrix, presented in Figure 7, shows strong diagonal dominance, with only two Normal sessions misclassified as Expert, suggesting a slight optimism bias but no systematic underestimation of player difficulty. Recall remains above 0.93 for all classes, confirming that the model does not under-represent novice players.

```
Epoch  10/100  Loss 0.6484
Epoch  20/100  Loss 0.4774
Epoch  30/100  Loss 0.3827
Epoch  40/100  Loss 0.2732
Epoch  50/100  Loss 0.2477
Epoch  60/100  Loss 0.2220
Epoch  70/100  Loss 0.2228
Epoch  80/100  Loss 0.2162
Epoch  90/100  Loss 0.2103
Epoch 100/100  Loss 0.2077
Test Acc: 97.65%
              precision    recall  f1-score   support

      Expert       0.89      1.00      0.94        17
      Normal       1.00      0.93      0.97        30
      Novice       1.00      1.00      1.00        38

    accuracy                           0.98        85
   macro avg       0.96      0.98      0.97        85
weighted avg       0.98      0.98      0.98        85
```

Figure 5. Training Table

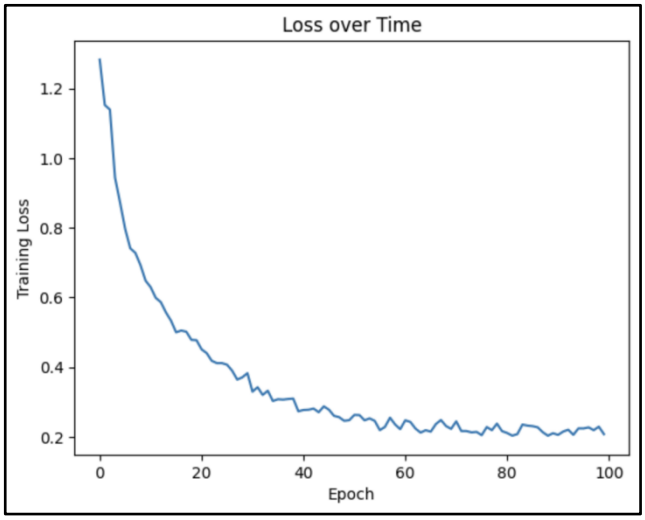


Figure 6. Loss Over Time

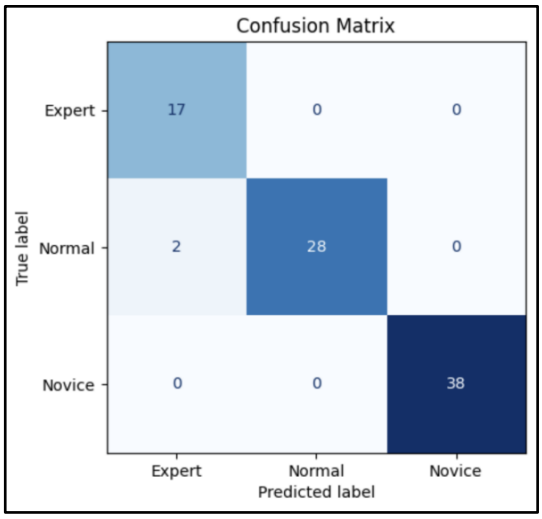


Figure 7. Confusion Matrix

The distribution of predicted skill classes reveals that 62% of players fall into the Novice category, 23% into Normal, and 10% into Expert. This skew suggests that most players face relatively challenging conditions and benefit from adaptive support mechanisms. Feature importance analysis shows that #Fail_Times contributes 48% of the total gain, #Time_For_Winning contributes 32%, and #Button_Click_Time contributes 20%, aligning with the intuitive notion that fail count is the strongest indicator of player struggle.

3.2. Live A/B Testing

We conducted a two-week online A/B test with 2,416 players tracked to control and treatment groups. In the treatment group, the adaptive system reduced frustration rate by 14.3% relative to the baseline, which is measured by the change in average passing times between failures, while compared with their own data prior to the deployment. Day-one retention increased from 15.2% to 21.4%, and day-seven retention improved by 2.1 percentage points. Importantly, average #Time_For_Winning remained statistically unchanged ($p > 0.05$), indicating that the system did not trivialize gameplay but instead targeted players most at risk of disengagement.

Player feedback surveys after each game revealed that 68% of respondents reported that the game felt “fairer” and “better paced.”, of whom 56% were classified as novice players before. These results

suggest that adaptive difficulty can improve subjective experience in addition to measurable engagement metrics.

4. DISCUSSION

The results confirm that integrating AI-driven player modeling into live games can meaningfully improve engagement while preserving challenge. The slight bias toward over-classifying players as Expert is acceptable given that the control policy includes rollback safeguards. The use of exponential weighting over recent sessions helps mitigate noisy fluctuations from single poor performances and provides a smoother difficulty trajectory.

A key strength of this work is its deployability: the ONNX inference engine runs efficiently on mobile devices without significant overhead, and the control policy is intentionally conservative to prevent perceptible oscillations.

Nevertheless, several limitations remain. The model currently treats all game levels as homogeneous, ignoring that some levels are inherently more difficult. Future work could incorporate level-specific normalization or train separate models per level tier. Another limitation is that the classifier relies solely on behavioral telemetry and does not use physiological data, which might better capture latent frustration.

5. CONCLUSION AND FUTURE WORK

This paper presented a complete AI-based dynamic difficulty adjustment pipeline using Conv1d network, from data preprocessing and feature extraction to model training, deployment, and live testing in “Gulltovia”. The approach yields high predictive accuracy and measurable improvements in retention and player satisfaction.

Beyond its application in Gulltovia, the proposed Conv1D-based DDA framework exhibits strong generalizability. Because the model’s inputs—fail counts, adjusted playtime, and button-click intensity—represent abstract behavioral metrics rather than game-specific variables, the same architecture can be retrained and applied across diverse genres such as action, puzzle, or strategy games. This adaptability suggests that AI-driven difficulty calibration can be standardized across titles without bespoke tuning for each game environment. As recent studies on time-series player modeling indicate [8], such cross-domain generalization opens a pathway toward universal DDA systems capable of autonomously learning difficulty adjustment rules from aggregated player telemetry.

Future research will explore reinforcement learning controllers that continuously adjust difficulty in small increments rather than discrete steps, as well as meta-learning techniques to enable cross-level adaptation. Incorporating multimodal signals such as heart rate or galvanic skin response as more and more smart devices are capable of detection could further improve frustration detection. Finally, longer-term studies should examine whether adaptive systems impact monetization and social dynamics in multiplayer contexts.

REFERENCES

[1] Csikszentmihalyi, Mihaly. Flow: The Psychology of Optimal Experience. Harper & Row, 1990. https://archive.org/details/flowpsychologyof00csik

[2] Hunicke, Robin. “The Case for Dynamic Difficulty Adjustment in Games.” Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, ACM, 2005, pp. 429–433. doi:10.1145/1178477.1178573.

[3] Zoeller, Georg. “Game Telemetry: The Key to Balancing Your Game.” Game Developers Conference (GDC), 2010, https://gdcvault.com/play/1012292/Game-Telemetry-The-Key-to

[4] Wang, Zhiguang, Weizhong Yan, and Tim Oates. “Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline.” 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 1578–1585. doi:10.1109/IJCNN.2017.7966039.

[5] Srivastava, Nitish, et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” Journal of Machine Learning Research, vol. 15, 2014, pp. 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

[6] Ioffe, Sergey, and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015, https://arxiv.org/abs/1502.03167

[7] Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” International Conference on Learning Representations (ICLR), 2015, https://arxiv.org/abs/1412.6980

[8] Mohammadi Foumani, Nasim, et al. “Deep Learning for Time Series Classification and Extrinsic Regression: A Survey.” ACM Computing Surveys, vol. 56, no. 2, 2024, Article 28, pp. 1–43. doi:10.1145/3624971.