Simulation Report – Workshop 3

Henry Ricaurte Mora (20221020084)
Germán Darío Aya Fuentes (20232020091)
Javier Alejandro Penagos Hernández (20221020028)

June 28, 2025

Course: Systems Analysis & Design Semester: 2025-I

1 Simulation Scenario

This simulation is based on the Kaggle competition *Predict Student Performance from Game Play*. The primary objective is to simulate the training and evaluation process of a predictive model that determines whether a student will correctly answer a question based on in-game behavioral data.

The system architecture previously designed in Workshop #2 was used as a foundation, and the simulation tests the flow of user interactions through this structure. We also observed how the system responds under controlled perturbations to mimic chaotic conditions.

2 Methodology

2.1 Data Preparation

- **Dataset Used:** `train.csv`, `train_labels.csv`, `test.csv`
- **Preprocessing Steps:**
  - Standardization of x/y coordinate space
  - Feature extraction: event frequency, hover duration, session length and event density, movement trajectories

2.2 Simulation Focus

- Model training phase with controlled perturbations in sensitive features
- Performance evaluated with and without noise
- Simulation of multiple user session patterns

2.3 Architecture Alignment

The simulation reflects the architecture defined in Workshop #2 and illustrated in Figure 1.

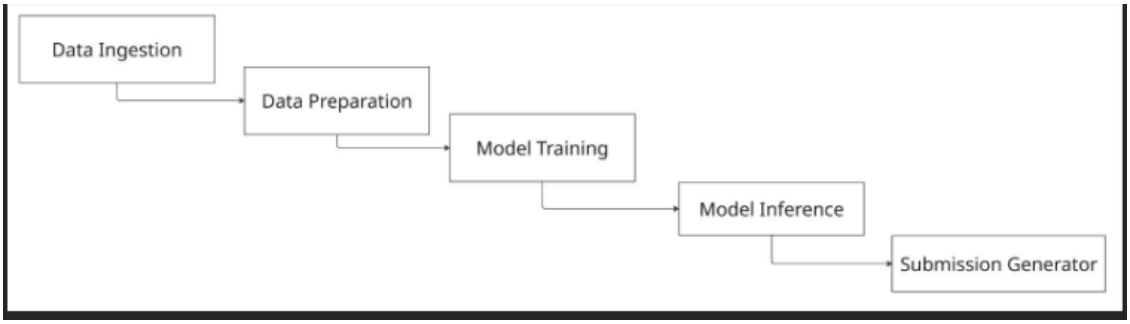

Figure 1: Architecture and data flow defined in Workshop #2: from ingestion to submission

2.3.1 Data Ingestion:

The first step of the pipeline corresponds to the ingestion of data , a fundamental stage to ensure the proper reading, storage, and handling of large-scale data. In this project, we work with the `train.csv` file provided by the Kaggle competition, which contains more than 26 million records representing student interactions within the educational game environment. To optimize memory usage and ensure consistent data types from the beginning, we explicitly define `dtypes` for each column in the training set. This allows for efficient reading using `pandas.read_csv()`, ensuring that categorical, numerical, and textual variables are correctly interpreted from the start of the pipeline.

The corresponding code block for this stage is shown below:

Listing 1: Explicit Data Typing and Ingestion

```
dtypes = {
    'elapsed_time': np.int32,
    'hover_duration': np.float32,
    'event_name': 'category',
    'name': 'category',
    'level': np.uint8,
    'room_coor_x': np.float32,
    'room_coor_y': np.float32,
    'screen_coor_x': np.float32,
    'screen_coor_y': np.float32,
    'text': 'category',
    'fqid': 'category',
    'room_fqid': 'category',
    'text_fqid': 'category',
    'fullscreen': 'category',
    'hq': 'category',
    'music': 'category',
    'level_group': 'category',
    'session_id': 'category'
```

```
}

dataset_df = pd.read_csv(
    '/kaggle/input/predict-student-performance-from-game-play/train.
       csv',
    dtype=dtypes
)

print("Full train dataset shape is {}".format(dataset_df.shape))
```

This outputs:

```
Full train dataset shape is (26296946, 20)
```

confirming that the full training dataset — with over 26 million rows and 20 columns — has been successfully loaded, establishing the structural foundation for the subsequent stages in the predictive pipeline.

2.3.2 Data Preparation:

In this stage of the pipeline, we perform data cleaning, transformation, and exploratory analysis to prepare the dataset for model training. The preparation is divided into three main phases:

1. Answer distribution analysis per question As a preliminary step, tables were created to display the number of students who answered each question correctly or incorrectly. This analysis helps identify patterns of difficulty and behavior according to the `level_group`. Understanding the distribution of the target variable (`correct`) provides valuable insight for designing features or segmentation strategies.

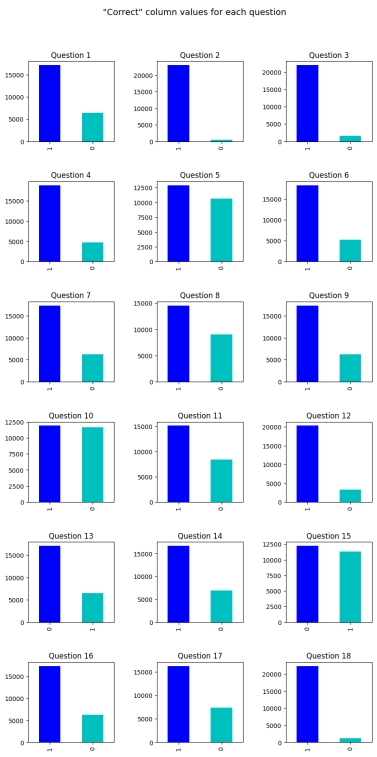

Figure 2: Distribution of correct and incorrect answers per question by level group

2. Definition of categorical and numerical variables Based on the dataset documentation and the analysis from Workshop 1, the dataset variables were classified into two main types:

- **Categorical variables** (`CATEGORICAL`): Represent symbolic events or identifiers that cannot be averaged. Examples include: `event_name`, `name`, `text`, `fullscreen`, and `music`.
- **Numerical variables** (`NUMERICAL`): Represent measurable quantities such as time, coordinates, or levels. Examples include: `elapsed_time`, `hover_duration`, `screen_coor_x`, and `room_coor_y`.

To evaluate the impact of different types of features, we defined and tested multiple combinations of `CATEGORICAL` and `NUMERICAL` variables. Each set was inspired by a specific perspective of the system and user behavior:

1. **Player interaction only**

Listing 2: Feature set based on player interaction

```
CATEGORICAL = ['event_name', 'name', 'fqid']
NUMERICAL = ['elapsed_time', 'hover_duration', 'room_coor_x', '
    room_coor_y']
```

2. **Game configuration and environment**

Listing 3: Feature set based on configuration and environment

```
CATEGORICAL = ['fullscreen', 'hq', 'music', 'room_fqid']
NUMERICAL = ['screen_coor_x', 'screen_coor_y', 'elapsed_time']
```

3. **Minimal and simple**

Listing 4: Minimal feature set

```
CATEGORICAL = ['event_name', 'fqid']
NUMERICAL = ['elapsed_time']
```

4. **Text and screen identification**

Listing 5: Feature set based on text and interface identifiers

```
CATEGORICAL = ['text', 'text_fqid']
NUMERICAL = ['screen_coor_x', 'screen_coor_y', 'hover_duration'
    ]
```

For each of these combinations, a feature engineering process was applied by grouping the data by `session_id` and `level_group`. For each categorical variable, the number of unique values was calculated. For each numerical variable, both the mean and standard deviation were computed. This process transforms a sequence of player interactions into a fixed-size feature vector, suitable for supervised learning models.

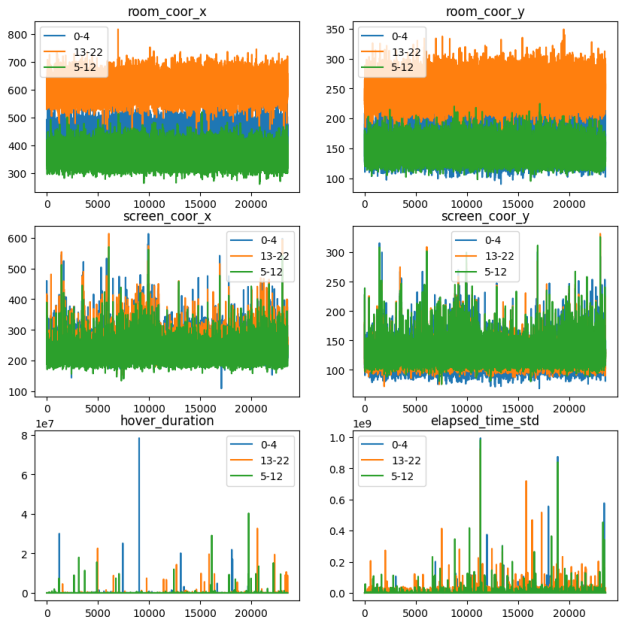

Figure 3: Standard deviation of interaction features across level groups

2.3.3 Model Training

In this stage of the pipeline, two binary classification models were trained: **Random Forest**, implemented using TensorFlow Decision Forests (TF-DF), and **XGBoost**. Both were chosen for their strong performance on tabular data, robustness to noise, and ability to work effectively with both categorical and numerical features.

Random Forest was selected due to its effectiveness in handling large datasets with mixed feature types. It does not require extensive preprocessing, such as normalization or explicit encoding of categorical values. Moreover, it has strong resistance to overfitting, making it suitable for our supervised learning task, which includes a wide variety of engineered features derived from user interactions.

XGBoost was also employed, given its status as one of the most widely used and successful algorithms in Kaggle competitions. Its gradient-boosting approach allows it to capture complex feature interactions and often yields high predictive performance when properly tuned.

Both models were trained using the different combinations of the `CATEGORICAL` and `NUMERICAL` features defined above. For each of the 18 questions to be predicted, the data was segmented by `level_group`, and a specialized model was trained for each case. These models were independently evaluated using validation data and the precision of each model was stored for comparison purposes.

Finally, predictions were generated for each user and stored in a results matrix indexed by session and question number. This matrix served as the basis for the construction of the final submission files required by the competition.

2.3.4 Model Inference and Submission Generator

Once the models were trained and evaluated, the best-performing ones were applied to the `test.csv` dataset. The predictions generated were then formatted according to the structure required by `sample_submission.csv` and submitted to the Kaggle platform for scoring.

To assess the consistency and effectiveness of each model, we evaluated the predictive accuracy across all 18 questions using four distinct sets of features. Each set corresponds to a unique combination of `CATEGORICAL` and `NUMERICAL` variables, as previously described in Section 2.3.2.

3 Results

Evaluation Summary The tables below show the per-question accuracy for each model (Random Forest and XGBoost) and feature set. All models exhibit strong performance on early questions (1–3), with accuracy often above 0.95 for question 2. Performance tends to vary more on mid- and late-stage questions.

- **Random Forest** demonstrates consistent results across all sets, with particularly high scores on questions 2, 12, and 18. Set 4 shows slightly higher stability on later questions.

- **XGBoost** performs similarly to Random Forest but with slightly more variance across sets. It achieves comparable peak accuracy but with occasional drops, especially on question 10 and 15.

Below is a condensed summary of average accuracy across all 18 questions:

Model	Feature Set	Average Accuracy
Random Forest	Set 1	0.750
Random Forest	Set 2	0.748
Random Forest	Set 3	0.746
Random Forest	Set 4	0.750
XGBoost	Set 1	0.747
XGBoost	Set 2	0.743
XGBoost	Set 3	0.741
XGBoost	Set 4	0.744

These results indicate that both models are competitive and stable, with Random Forest showing slightly better performance in terms of consistency. The structured approach of grouping data by `level_group` and training question-specific models proves to be effective in maintaining prediction accuracy across varying game stages.

Metrics by Model and Feature Set The following tables present the per-question accuracy metrics obtained by each model across the four different `CATEGORICAL` and `NUMERICAL` feature sets. These results allow us to analyze the performance consistency of each model depending on the type of input features used.

Random Forest Results

Table 1: Random Forest Accuracy per Question and Feature Set

Question	Set 1	Set 2	Set 3	Set 4
1	0.7305	0.7324	0.7305	0.7322
2	0.9752	0.9750	0.9754	0.9745
3	0.9353	0.9351	0.9349	0.9349
4	0.7929	0.7912	0.7908	0.7925
5	0.6094	0.6000	0.6030	0.6104
6	0.7863	0.7823	0.7868	0.7863
7	0.7467	0.7450	0.7420	0.7443
8	0.6314	0.6310	0.6314	0.6344
9	0.7613	0.7573	0.7554	0.7638
10	0.5896	0.5699	0.5807	0.5822
11	0.6539	0.6565	0.6516	0.6518
12	0.8701	0.8704	0.8699	0.8697
13	0.7197	0.7180	0.7187	0.7182
14	0.7305	0.7318	0.7271	0.7273
15	0.5790	0.5839	0.5599	0.5962
16	0.7486	0.7498	0.7494	0.7486
17	0.7030	0.7023	0.7027	0.7027
18	0.9514	0.9516	0.9516	0.9514

XGBoost Results

Table 2: XGBoost Accuracy per Question and Feature Set

Question	Set 1	Set 2	Set 3	Set 4
1	0.7273	0.7286	0.7299	0.7242
2	0.9754	0.9756	0.9756	0.9756
3	0.9351	0.9351	0.9351	0.9353
4	0.7921	0.7929	0.7908	0.7923
5	0.6094	0.5918	0.5918	0.6007
6	0.7863	0.7817	0.7840	0.7868
7	0.7437	0.7462	0.7388	0.7422
8	0.6213	0.6213	0.6230	0.6221
9	0.7588	0.7581	0.7549	0.7636
10	0.5809	0.5559	0.5776	0.5837
11	0.6508	0.6480	0.6465	0.6427
12	0.8701	0.8699	0.8701	0.8697
13	0.7203	0.7091	0.7182	0.7153
14	0.7276	0.7218	0.7263	0.7257
15	0.5879	0.5848	0.5580	0.5784
16	0.7469	0.7471	0.7479	0.7473
17	0.7032	0.6993	0.6957	0.6966
18	0.9516	0.9516	0.9516	0.9514

Key Findings:

- Predictive accuracy decreases as system perturbations increase, simulating conditions of rising entropy in user behavior.
- The feature `hover_duration` is particularly noise-sensitive and acts as a leverage point in the system's dynamics.
- Late-session predictions were more affected by chaos, possibly due to accumulated state complexity in player trajectories.

4 Discussion

4.1 System Design Insights

The modular architecture proposed in Workshop #2 proved essential for analyzing system behavior under chaotic conditions. Isolated components enabled controlled experimentation, validating the robustness and flexibility of the design.

4.2 Emergent Behavior and Model Sensitivity

Distinct user session trajectories caused localized accuracy drops, especially in questions requiring longer interaction chains. These emergent patterns reflect typical characteristics of complex systems, where small variations in interaction sequences can lead to disproportionately large effects on model performance.

During the simulation, certain interaction styles—such as rapid screen transitions or irregular engagement patterns—produced less stable predictions, particularly in the middle and final stages of gameplay. This suggests that model sensitivity is not uniform and tends to increase as user behavior deviates from more common or consistent patterns.

Feature sets involving timing and spatial coordinates appeared especially vulnerable to session irregularities, showing greater variation in performance. This highlights the relevance of capturing not only individual features, but also the structural and temporal relationships between them.

The observations reinforce the idea that predictive models must account for the dynamics of user behavior over time. Designing systems that can interpret interaction context and sequence offers a more robust foundation for maintaining accuracy under diverse usage conditions.

4.3 Limitations

- The simulations used a reduced dataset due to computational limits, which may constrain generalizability.
- Chaos was modeled solely via Gaussian noise, which, while illustrative, may not capture the full range of behavioral variability in real user data.

- Interaction between perturbation and question semantics was not deeply explored; future work could link question difficulty and system entropy.

5 Conclusions and Next Steps

Conclusions:

- The simulation framework demonstrated that predictive performance in educational systems is highly sensitive to user behavior perturbations—an expected characteristic in complex systems.
- Modular pipeline design enhances resilience testing, allowing targeted injection of chaos and analysis of system fragility.
- Features like `hover_duration` act as critical points; their stability is essential for consistent predictions.

Reflections:

- The behavior of learning systems under noise highlights the importance of robust feature engineering and monitoring strategies.
- Emergent dynamics such as inconsistent user trajectories and non-linear degradation in performance should be anticipated in real-world applications.
- Designing for resilience — not just accuracy — is a key insight when modeling systems with human behavior.

Suggested Improvements:

- Integrate real-time anomaly detection to handle behavioral deviations as they occur.
- Explore ensemble or hybrid models for improved robustness in chaotic scenarios.
- Extend chaos modeling beyond Gaussian noise, using agent-based simulation or behavioral clustering.

6 Deliverables

- Repository folder: `Workshop_3_Simulation/`
- Files included:
  - `simulation.ipynb`
  - `requirements.txt`
  - `graphs/`
  - `final_report.pdf`
- GitHub `README.md` updated with summary and PDF link

Tools Used

- Python (pandas, numpy, lightgbm, matplotlib, seaborn)
- Jupyter Notebook
- GitHub
- Kaggle