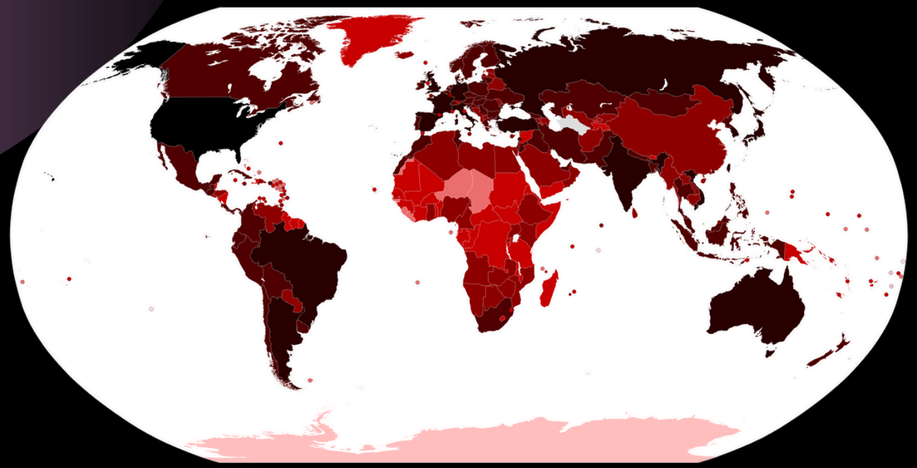
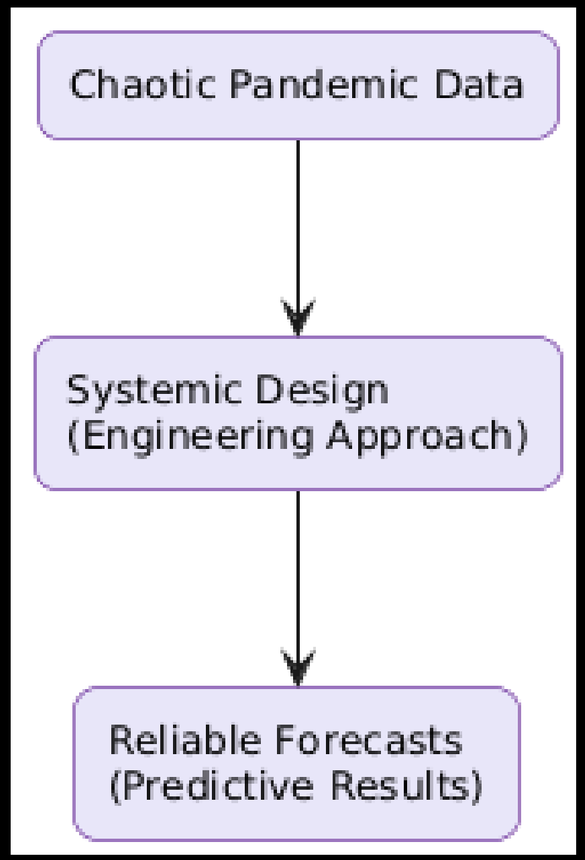
Covid 19 Forecasting Model design



Introduction

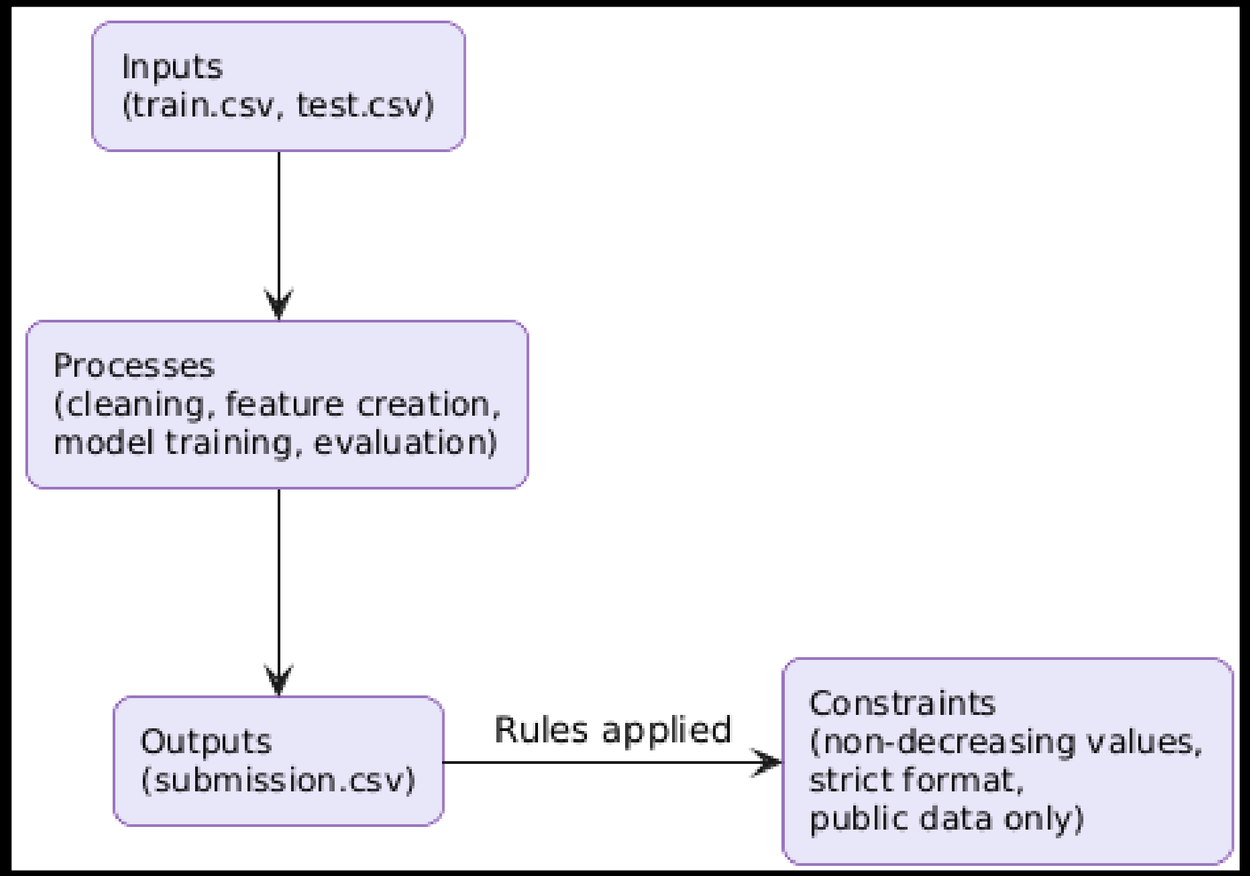
- The COVID-19 pandemic created a chaotic and unpredictable environment, where daily data changed rapidly and inconsistently.
- To address this, the Kaggle “COVID-19 Global Forecasting” competition challenged participants to design reliable forecasting systems.
- The main goal: predict confirmed cases and fatalities using global data under uncertainty.
- The system we developed applies systems engineering principles to handle chaos, randomness, and data instability.

PURPOSE AND MOTIVATION



- Many COVID-19 models focused only on accuracy but ignored reproducibility and adaptability.
- Our project aimed to build a modular and adaptive pipeline that could:
  - Process inconsistent data,
  - Adjust to sudden changes, and
  - Maintain stable predictions despite chaos.
- The system was designed not to avoid uncertainty, but to operate effectively within it.

SYSTEM COMPONENTS AND CONSTRAINTS



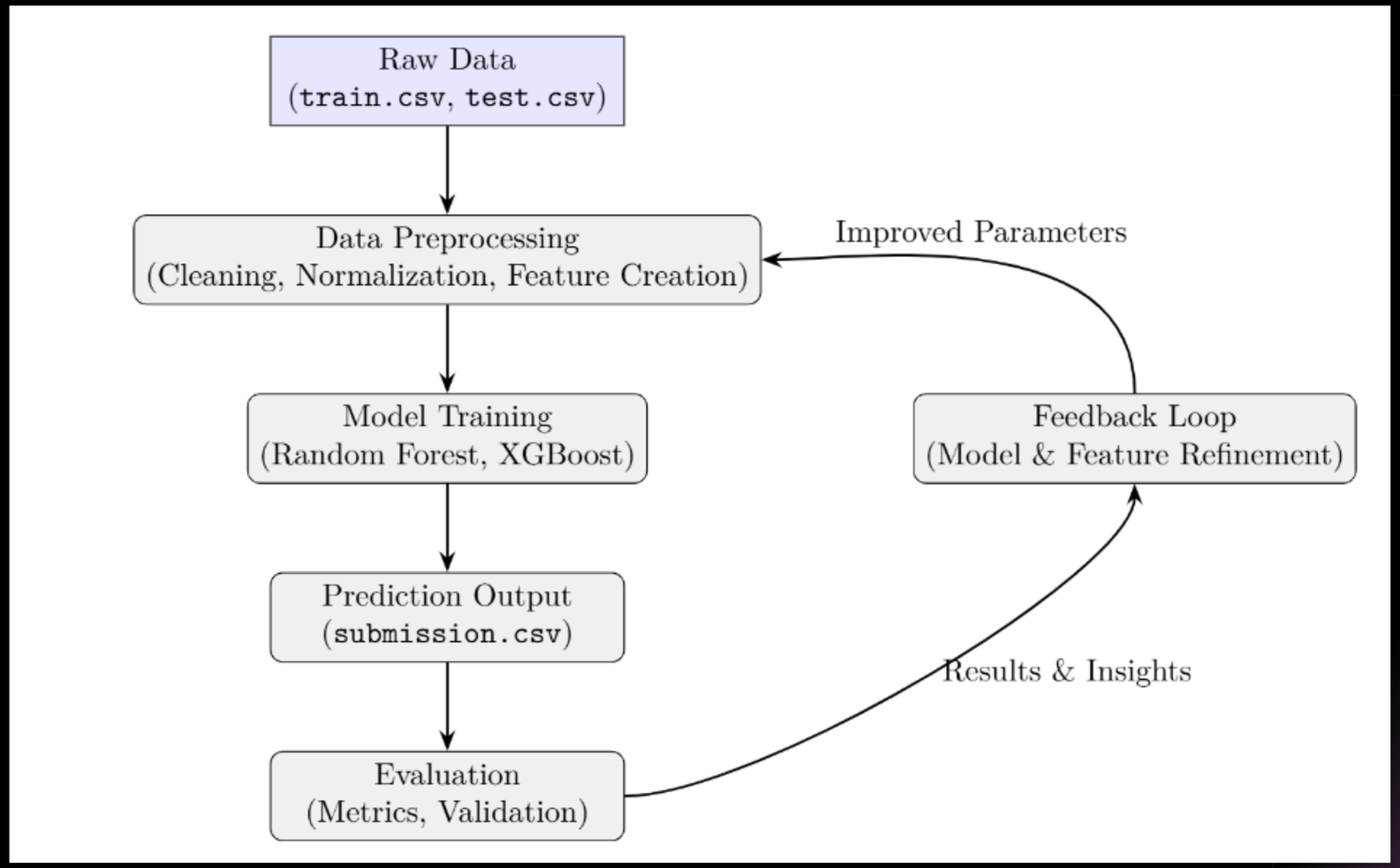
Main Components:

- Inputs: train.csv, test.csv → historical and future data.
- Processes: preprocessing, feature creation, model training, evaluation.
- Outputs: submission.csv → cumulative confirmed cases and deaths.

Main Constraints:

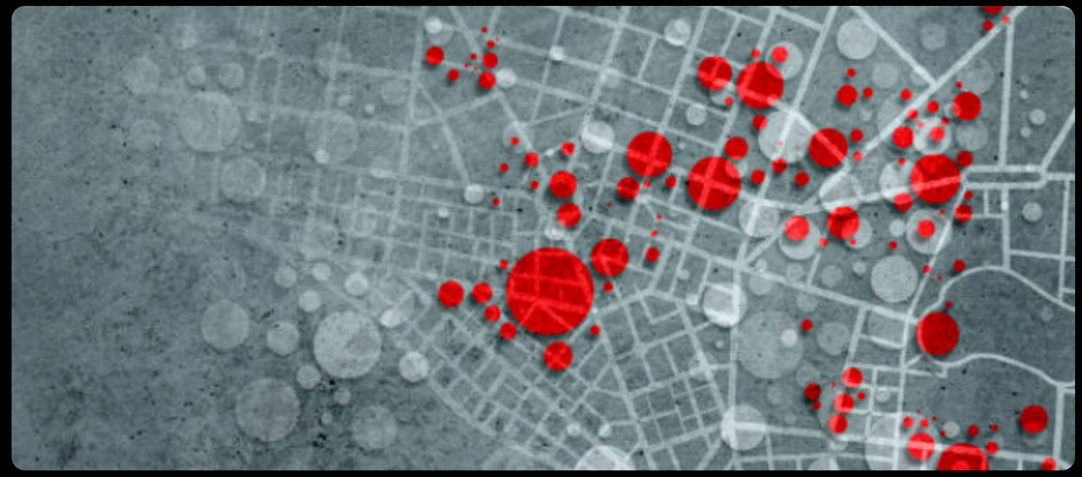
- Predictions must be non-decreasing and cumulative.
- Strict submission format required by Kaggle.
- Only publicly available data allowed.
- These constraints ensured fairness but increased system sensitivity to data noise.

INTERACTION BETWEEN MODULES



The interaction between modules defines the logical data flow from input to prediction output.

SYSTEM CONTEXT: CHAOS AND UNCERTAINTY

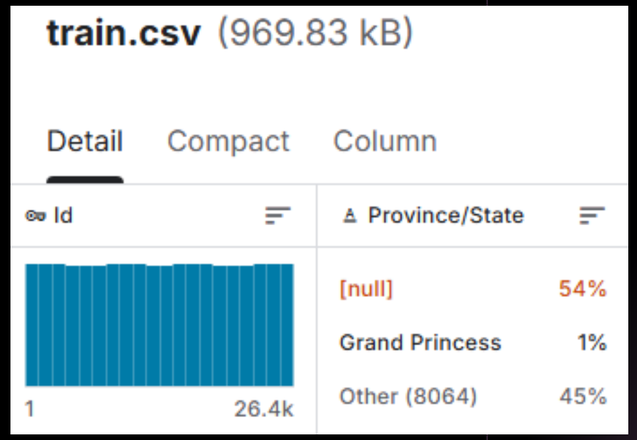


- Real-world data was irregular, delayed, or incomplete.
- Government actions (lockdowns, reopenings) and social behavior constantly changed the trend.
- Random reporting errors produced large shifts in the curves.
- The system had to stay stable and reproducible under these unpredictable dynamics.

COMPLEXITY, SENSITIVITY AND CONTROL

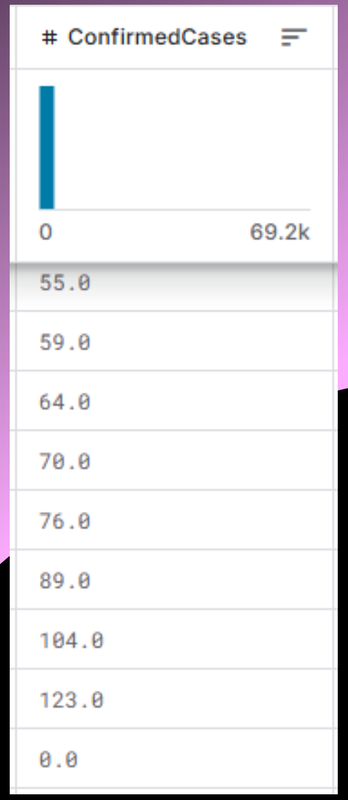
Complexity

- Incomplete or noisy data
- Overfitting risk
- Hyperparameter sensitivity



Sensitivity

Few inconsistent data points or outliers can lead to large differences in predicted confirmed cases or fatalities.



Expected Mechanism

Control

Algorithm 1 COVID-19 Forecasting and Feedback Control Loop

Require: train.csv, test.csv, retraining threshold δ
Ensure: submission.csv with predicted ConfirmedCases and Fatalities

1: Load train and test datasets
2: Preprocess data: clean, normalize, and engineer features
3: Train model (Random Forest or XGBoost)
4: Generate predictions for test data
5: Evaluate results using RMSLE metric
6: **if** error ¿ δ **then**
7: Retrain model with updated data
8: **end if**
9: Output submission.csv and update via Kaggle API

INGESTION MODEL OVERVIEW

- Normalizes all text fields; fills missing provinces with empty strings.
- Generates a unified region key (Country/Region + Province).
- Converts dates to consistent datetime objects; checks temporal integrity.
- Produces a complete metadata JSON for traceability.

- Validates file existence and computes SHA-256 hashes.
- Loads CSV files safely (robust parsing, null-handling, date parsing).
- Validates all required schema variants (train/test format).

EXAMPLE:
Code Implementantion for Unified Region Key

```python
def create_region_key(df: pd.DataFrame) -> pd.DataFrame:
    """Add/normalize a 'region' column combining Country/Region and Province/State."""
    df = df.copy()
    # Ensure columns exist
    if "Province/State" not in df.columns:
        df["Province/State"] = ""
    if "Country/Region" not in df.columns:
        raise ValueError("Missing required column 'Country/Region'")
    df["Province/State"] = df["Province/State"].fillna("").astype(str).str.strip()
    df["Country/Region"] = df["Country/Region"].fillna("").astype(str).str.strip()
    # create region key
    df["region"] = df["Country/Region"]
    # add province only if non-empty
    mask = df["Province/State"] != ""
    df.loc[mask, "region"] = df.loc[mask, "Country/Region"] + "__" + df.loc[mask, "Province/State"]
    return df
```

EVIDENCE OF INGESTION IMPROVEMENTS

- Cleaned distributions demonstrate reduced noise.
- Unified region identifiers produce coherent aggregations.
- Time series are consistent and free of reporting artifacts.
- Outliers and extreme values are now manageable and traceable.

Testing of validations with time data without coherence, and missing columns through load_and_validate function.

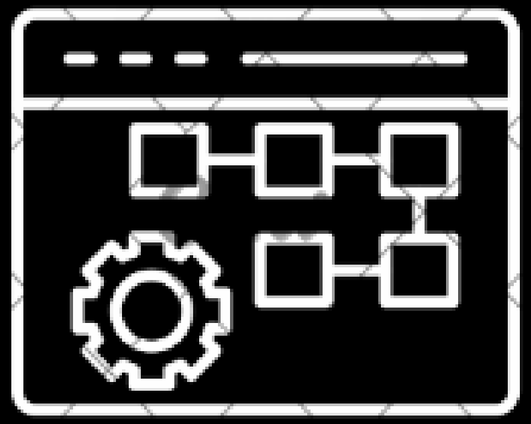
```
[TEST] Verifying Schema Validation (Missing Columns)...
PASSED: Caught missing columns -> Train columns validation failed: Missing columns: {'ConfirmedCases'}. Present columns: {'Country/Region', 'Id', 'Provi
nce/State', 'Date', 'Fatalities'}

[TEST] Verifying Type Safety (Invalid Date)...
PASSED: Caught invalid date -> Found 1 unparsable dates in test_ingestion_demo\train_bad_date.csv

[TEST] Verifying Logical Integrity (Negative Growth)...
PASSED: Detected negative growth anomalies in regions: ['B__A']

[TEST] Verifying Traceability (Manifest Generation)...
PASSED: Data manifest created successfully.
   -> Hash recorded: 352ca3aeff...
```

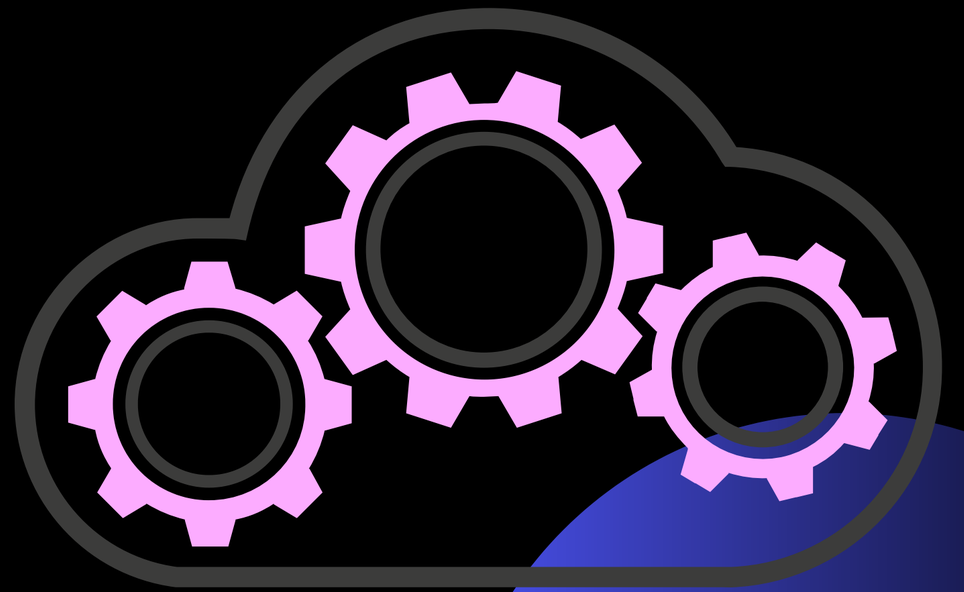
IMPACT ON SYSTEM ARCHITECTURE

- Ensures ML training is reproducible and free from structural noise.
- Enables consistent cross-region comparisons.
- Supports deterministic behavior in the event-based simulation.
- Reduces risk of cascading failure in the architecture.

Allows the system to meet design criteria:

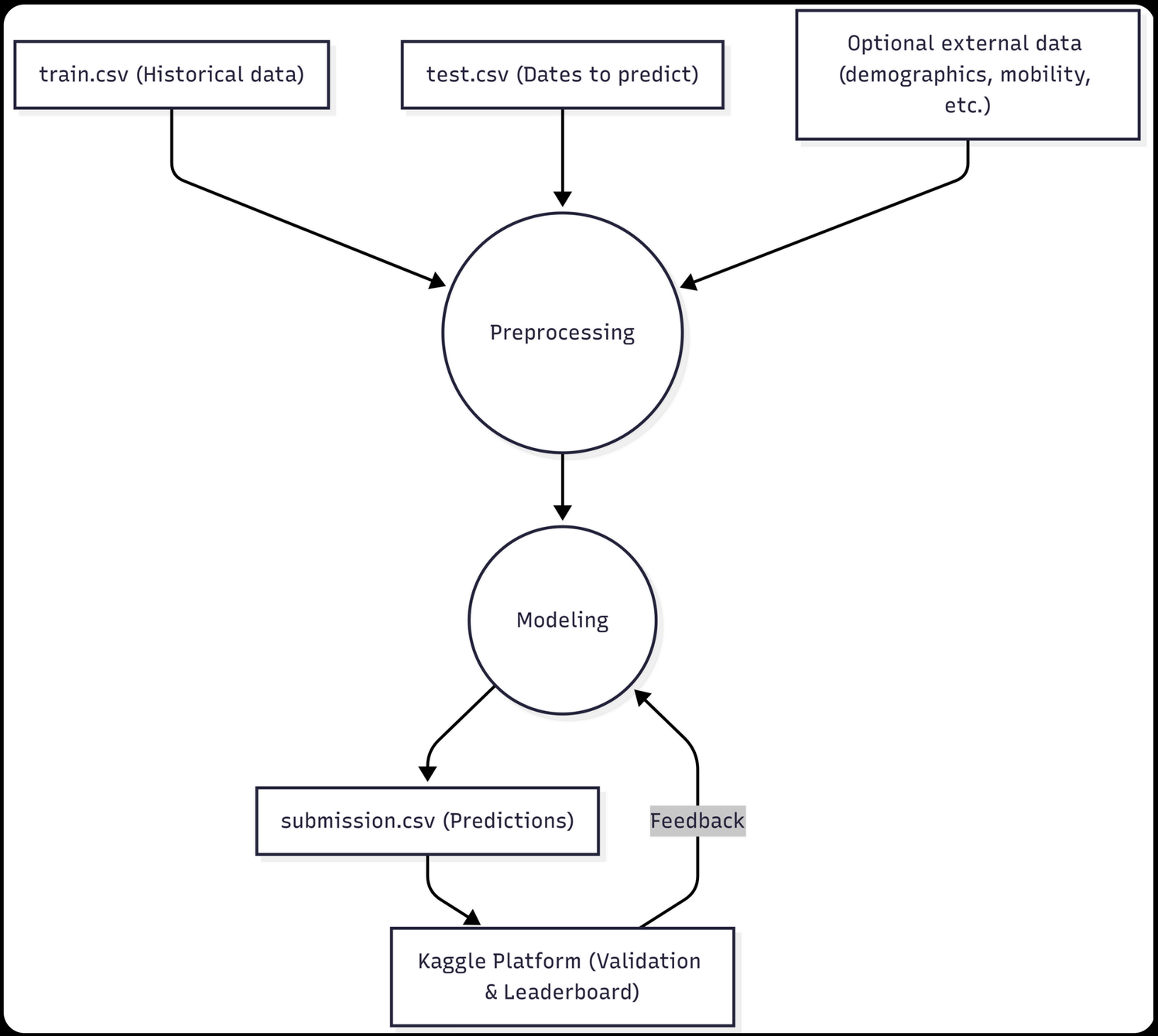
predictability
consistency
control
reliability
traceability





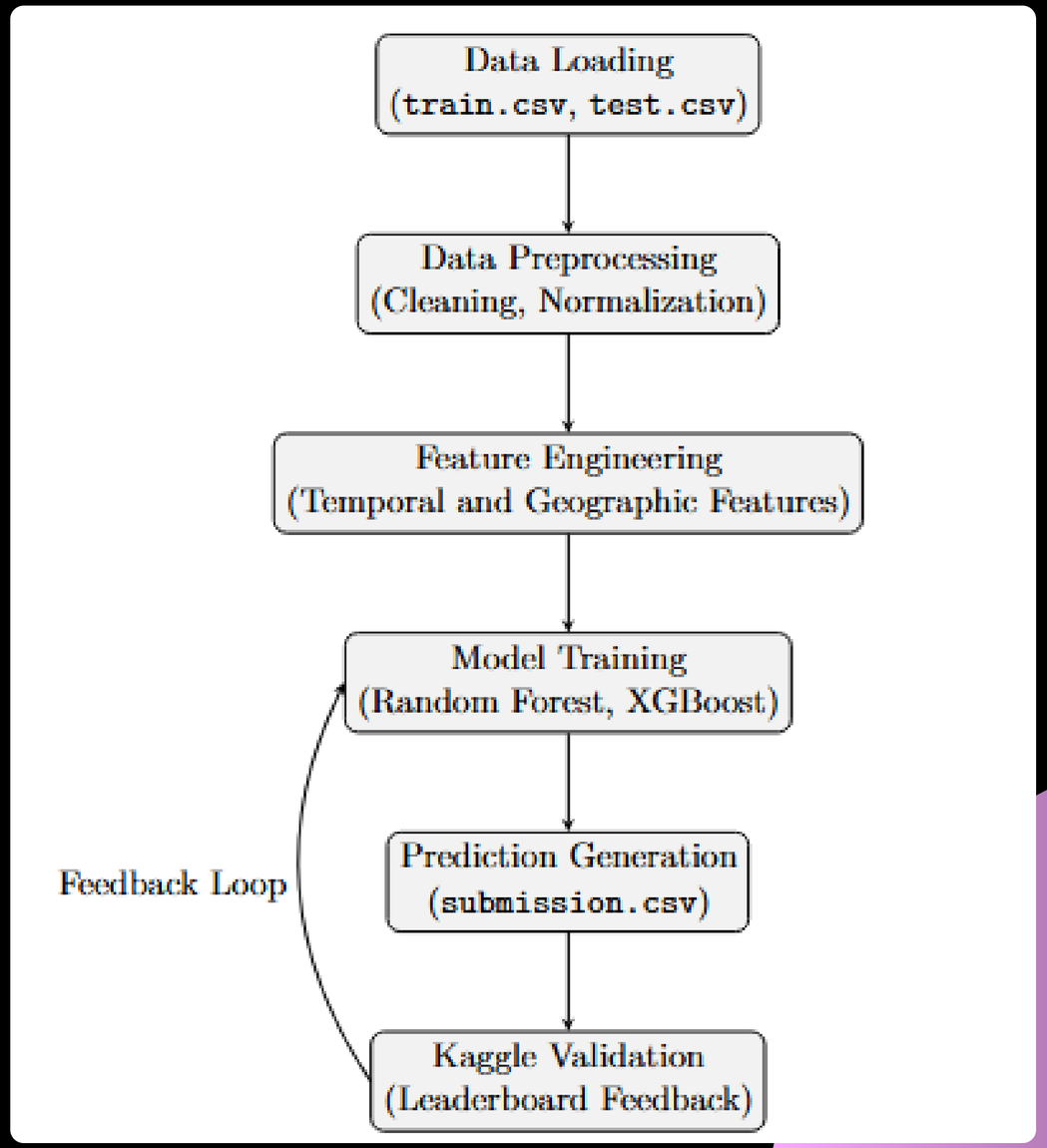
SYSTEM ANALYSIS

- Inputs
- Processes
- Outputs
- Actors



TECHNICAL STACK AND IMPLEMENTATION SKETCH

Python is used to develop the system due to its adaptability and the strength of its data science and machine learning ecosystem. This language enables effective data management, rapid prototyping, and seamless integration with analytical tools.



ARCHITECTURE DIAGRAM

illustrates the modular flow of the forecasting system. Data ingestion and preprocessing form the foundation of the pipeline, ensuring data integrity and consistency.

FUTURE ENGINEERING

- New features created
- Day index
- Week of year
- Growth rate
- Moving averages (7 days)
- Rate of change
- Country + region encoded
- Outlier flags

```python
# 2. FEATURE ENGINEERING
def feature_engineering(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()

    df["dayofweek"] = df["Date"].dt.dayofweek
    df["weekofyear"] = df["Date"].dt.isocalendar().week.astype(int)

    df["case_growth_rate"] = (
        df.groupby("region")["ConfirmedCases"]
        .pct_change()
        .replace([np.inf, -np.inf], np.nan)
        .fillna(0)
    )

    df["fatal_growth_rate"] = (
        df.groupby("region")["Fatalities"]
        .pct_change()
        .replace([np.inf, -np.inf], np.nan)
        .fillna(0)
    )

    df["cases_ma7"] = (
        df.groupby("region")["ConfirmedCases"]
        .rolling(7)
        .mean()
        .reset_index(level=0, drop=True)
        .fillna(0)
    )

    df["fatal_ma7"] = (
        df.groupby("region")["Fatalities"]
        .rolling(7)
        .mean()
        .reset_index(level=0, drop=True)
        .fillna(0)
    )

    return df
```

SIMULATION PLANNING

Scenario 1 — Data-Driven Simulation

- ML-based simulation
- System learning evaluation
- Disturbances and sensitivity analysis
- Feedback loop

Scenario 2 — Event-Based Simulation

- Cellular automata
- Spatial diffusion
- Local rules → emergent behaviors
- Disturbances and noise

Simulation Implementation

Main components implemented

- model_simulation.py
- ML training
- Linear regression
- Random forest
- MLPRegressor
- Learning curves
- Sensitivity analysis
- Feedback loop simulation
- cellular_automata.py
- Grid
- Neighbors
- Transition rules
- Noise
- Temporal evolution
- Heat map result

ML SIMULATION

What does the system do?

- Trains classic ML models
- Evaluates MAE and RMSE
- Generates learning curves
- Calculates sensitivity to 5% changes
- Generates a feedback loop:
  - The prediction is used as new input → simulates how the system “learns”

```
=== TRAINING BASE MODELS  ===

Cross validation for xgboost...
MAE CV: 50.9762 ± 46.0207
Training final model for xgboost...
Model saved on: Project/models/xgboost.pkl
Updated report: Project/models/model_report.csv

Cross validation for random_forest...
MAE CV: 25.5948 ± 32.1650
Training final model for random_forest...
Model saved on: Project/models/random_forest.pkl
Updated report: Project/models/model_report.csv

Cross validation for gradient_boosting...
MAE CV: 16.4361 ± 17.8946
Training final model for gradient_boosting...
Model saved on: Project/models/gradient_boosting.pkl
Updated report: Project/models/model_report.csv
```

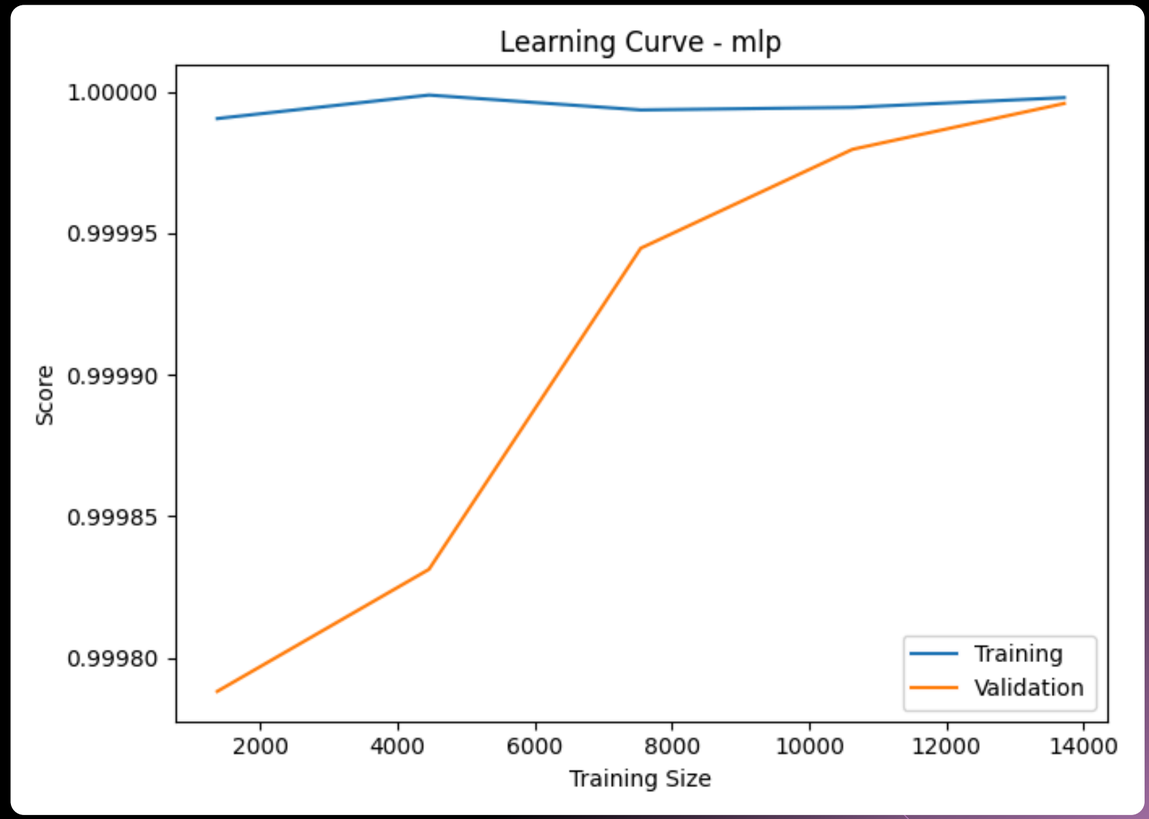
LEARNING CURVES

What do they show?

- How the error evolves with more data

Detects:

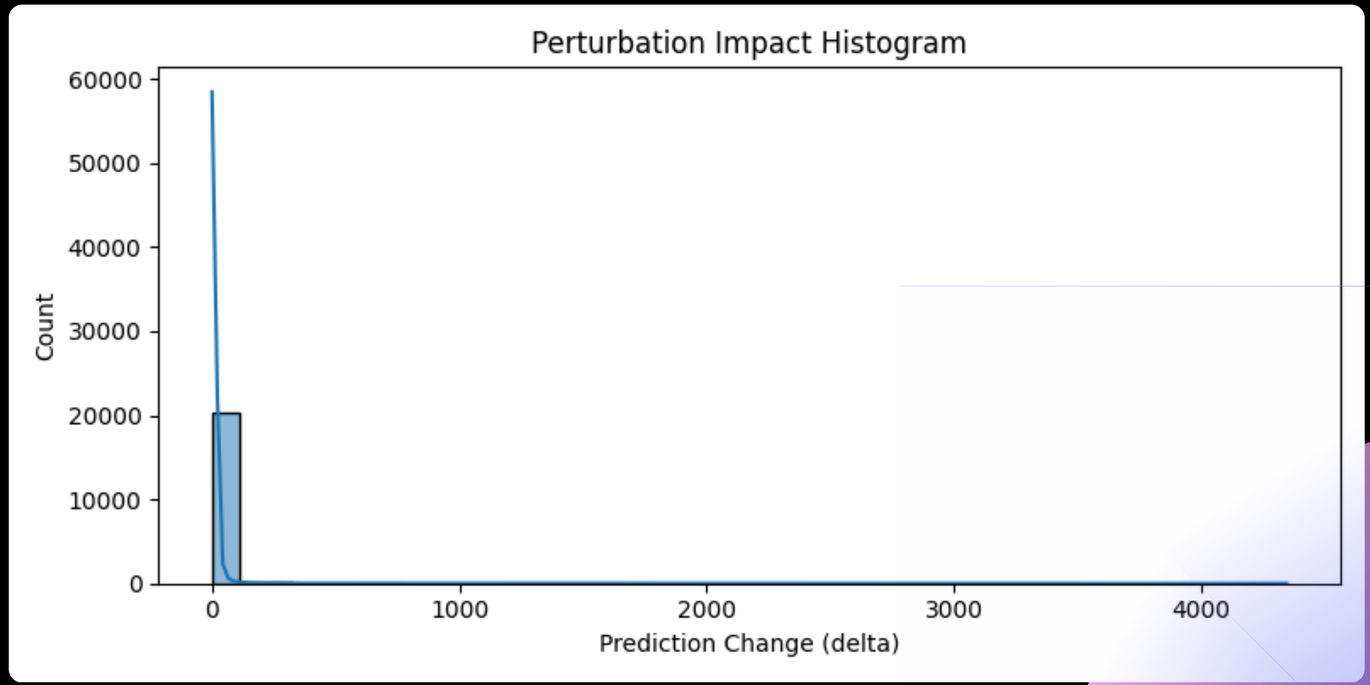
- Overfitting
- Underfitting
- Model stability



SENSITIVY ANALYSIS

How does it work?

- We alter the features by ±5%.
- We recalculate the predictions.
- We measure the absolute difference.
- We construct a sensitivity histogram.



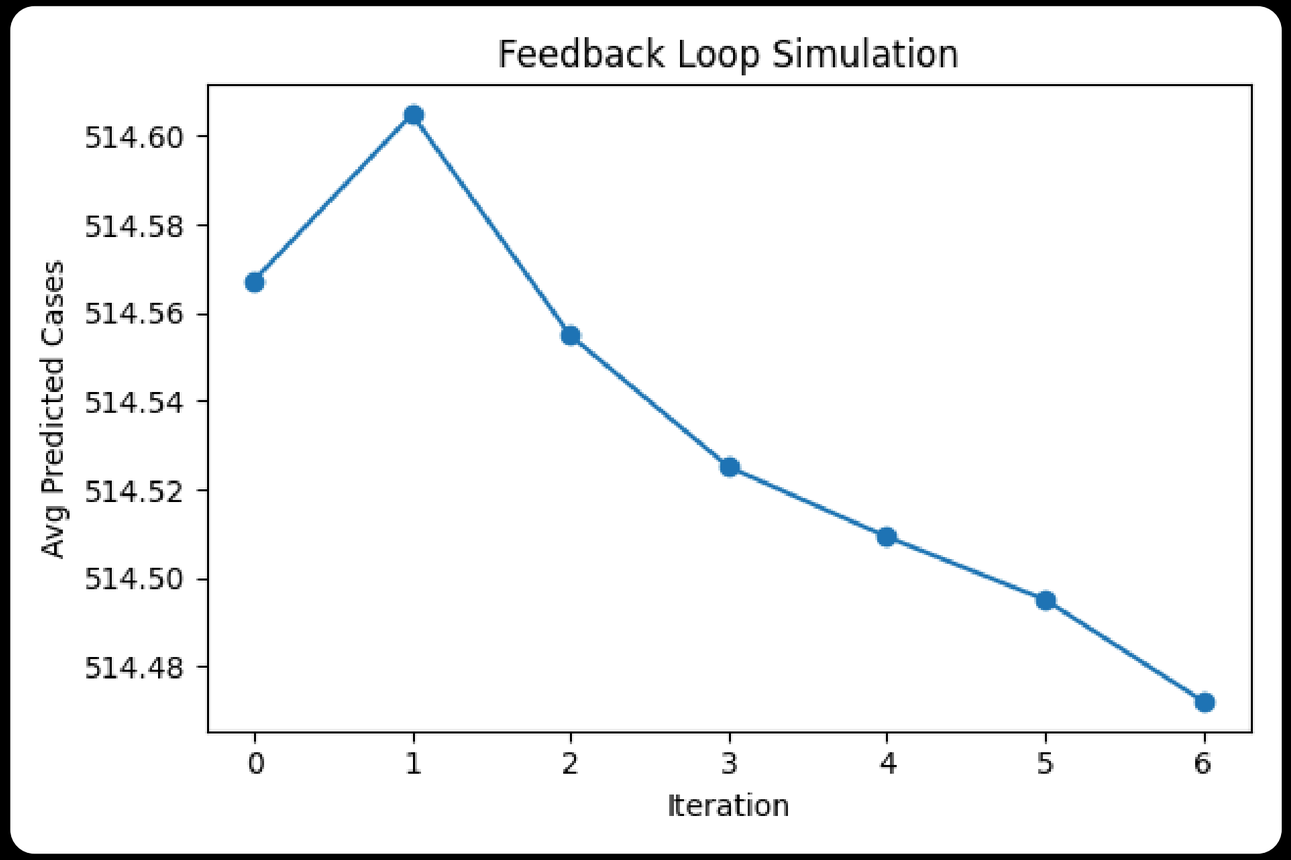
Feedback Loop Simulation

What does it measure?

- How stable is the system when its own predictions feed into it?

Observed behaviors:

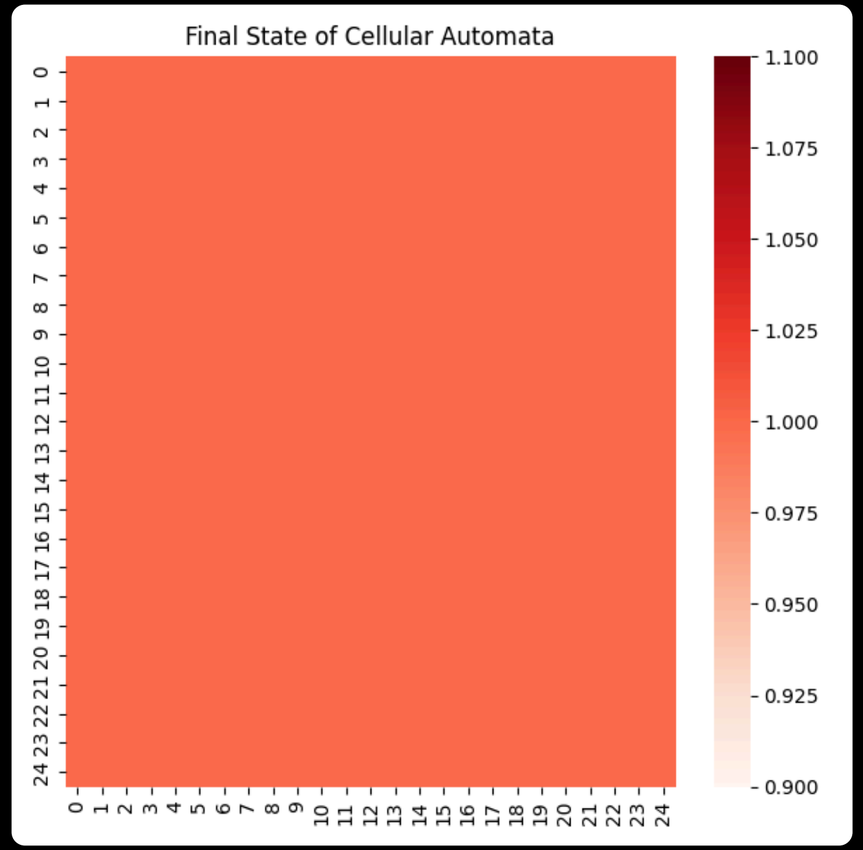
- Convergence
- Divergence
- Oscillations
- Exponential explosion (MLP in some cases)



EVENT-BASED SIMULATION

What does the cellular automata do?

- It creates a 25×25 matrix.
- States: infected (1) or healthy (0)
- Moore neighborhood (8 neighbors)
- Probability of infection:
- Evolves through 30 iterations
- Produces a final heatmap



$$P = \text{infection\_rate} \times N_{\text{infected neighbors}} + \text{noise}$$

INTEGRATION WITH THE PIPELINE

How do simulations connect to the system?

Simulations help to:

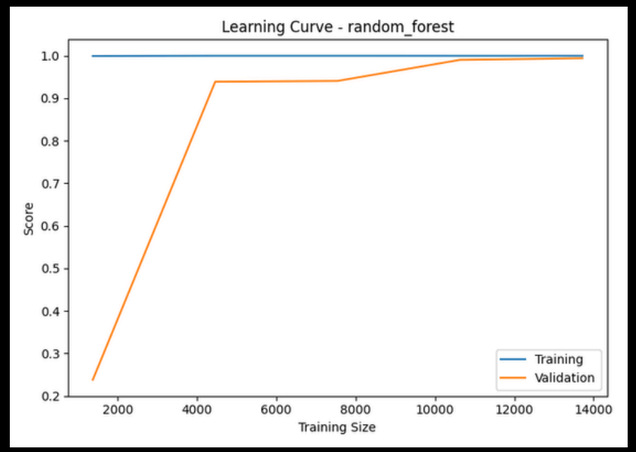
- evaluate model stability
- detect sensitivity
- understand spatial behavior
- complement ML modeling
- anticipate structural failures in real data

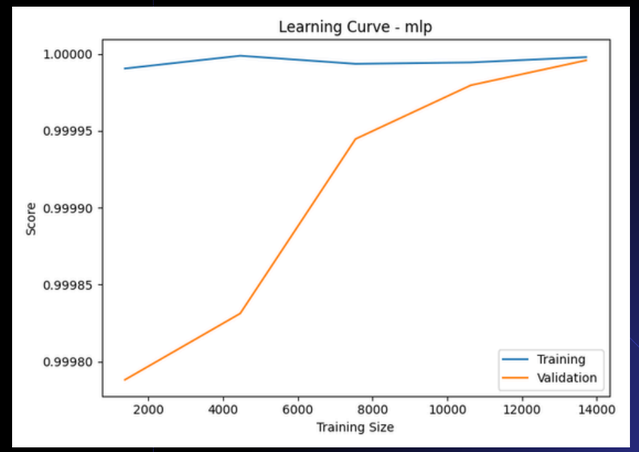
The system becomes more robust and explainable.

RESULTS

I. Model Comparison and Precision

- ***Maximum Precision (Scenario 1):*** Multi-Layer Perceptron was the most stable model, achieving high precision (MAE: 1.16).
- ***Spatial Insight (CA):*** Cellular Automata provided non-linear, dynamic vision, modeling emergent spatial patterns to complement ML.
- ***Model Comparison:*** MLP outperformed Random Forest (MAE: 3.04), confirming its superior stability.

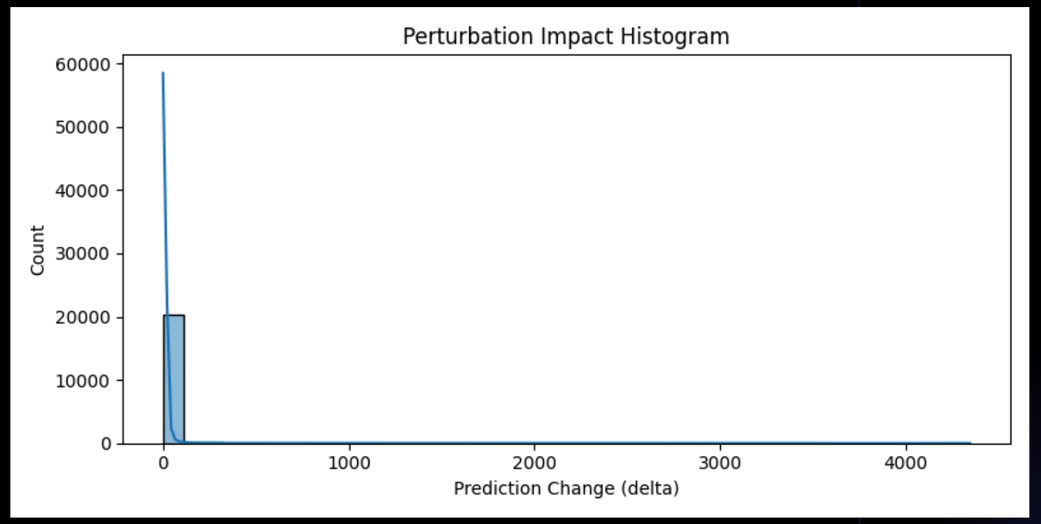




RESULTS

II. Stability, Robustness, and Continuous Learning

- ***High Robustness:*** Sensitivity Analysis (+- 5% noise) confirmed high resilience to data irregularities and chaos.
- ***Model Stability:*** Learning Curves showed low error divergence, validating no overfitting and strong generalization ability.
- ***Adaptive Design:*** The Feedback Loop proved the system's capacity for stable parameter convergence and continuous refinement.



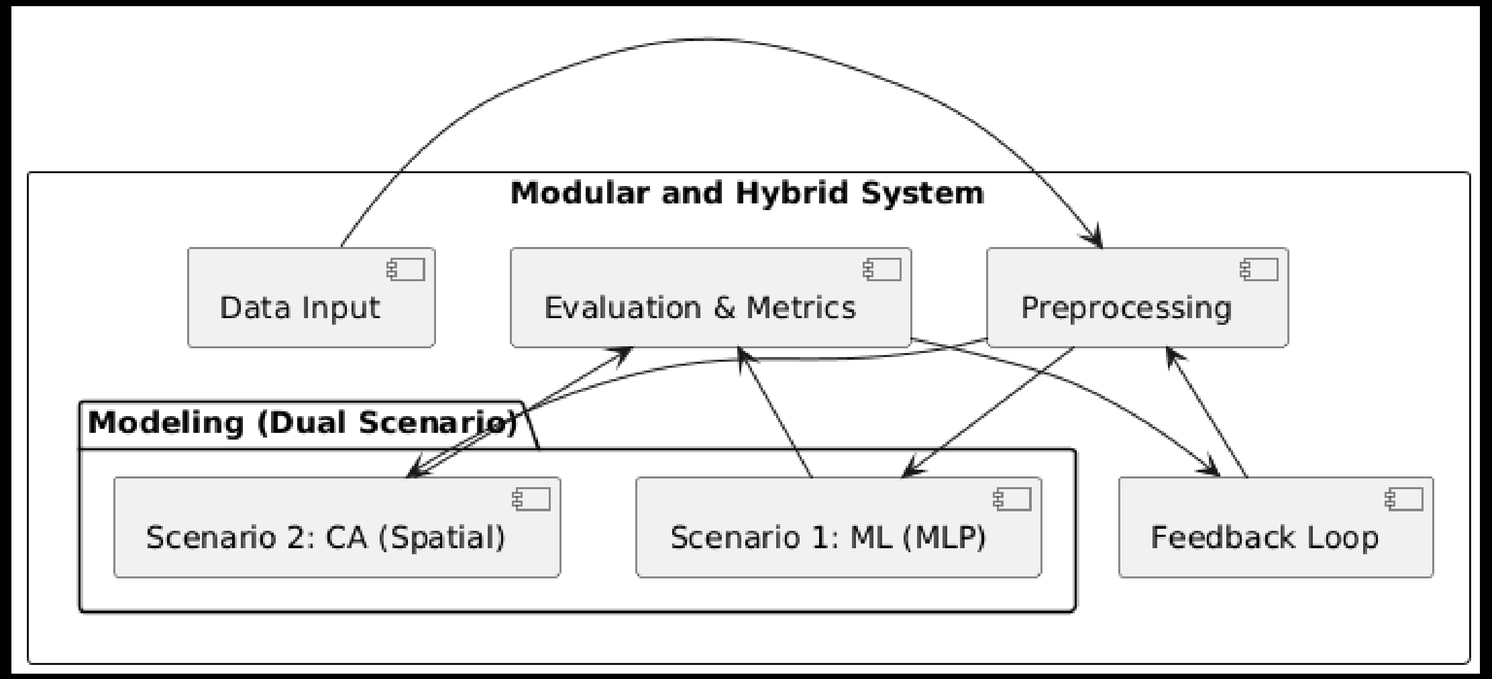
CONCLUSIONS

Hybrid Architecture and Design Validation

- ***Hybrid Design Success:*** The system successfully validated a dual-scenario architecture, integrating MLP Regressor (for numerical accuracy) and Cellular Automata (CA) (for spatial dynamics).

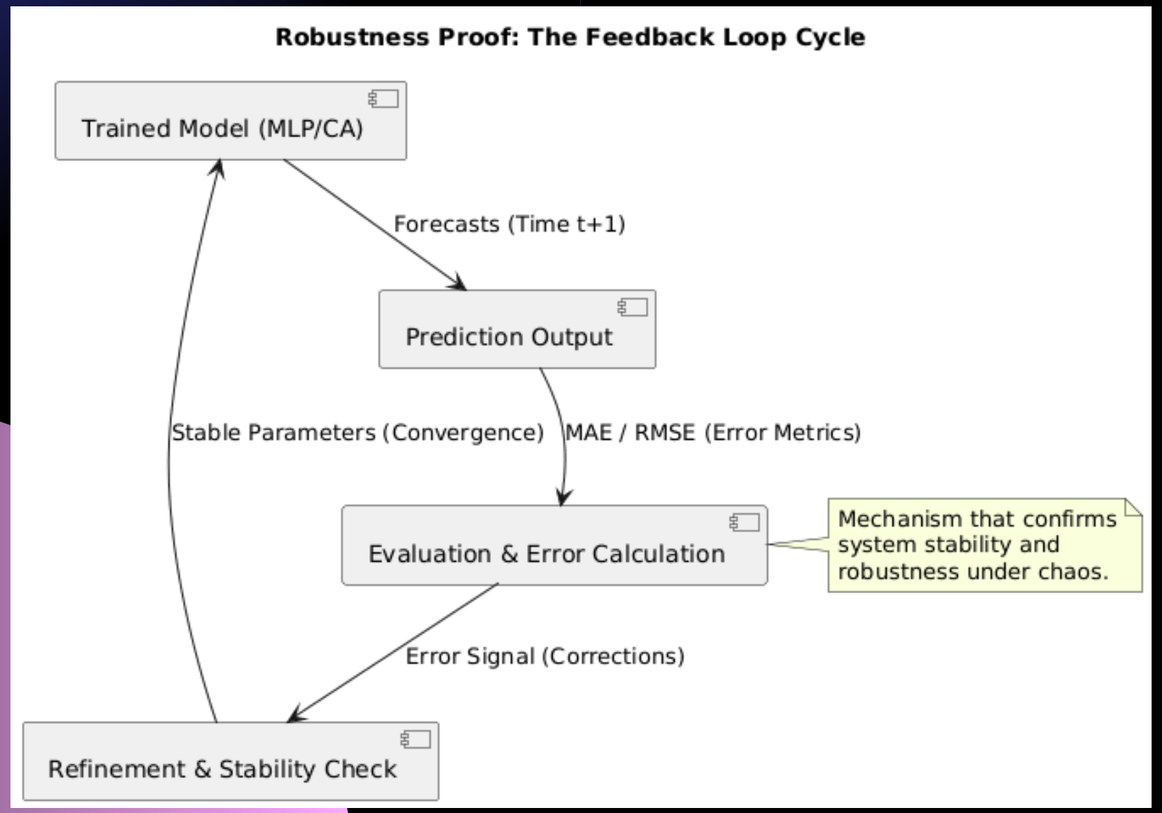
- ***Engineering Goal Met:*** Application of Systems Engineering principles resulted in a modular, scalable, and reproducible data pipeline, ensuring data traceability and coherence.

- ***Quantitative Accuracy:*** ML models achieved high trend-prediction accuracy, with the stable MLP model registering a MAE of 1.16.



CONCLUSIONS

Robustness and Stability Demonstrated



- ***High Robustness:*** Sensitivity Analysis (+-5% noise) confirmed the system's high resilience to input data instability and real-world perturbations.
- ***Validated Stability:*** Learning Curves showed a low error divergence, confirming the absence of overfitting and high generalization capacity.
- ***Continuous Refinement:*** The Feedback Loop proved the system's ability to simulate continuous learning, achieving parameter convergence essential for deployment in dynamic environments.

Thank **You**

-