AutoMockAI – Schema-Aware AI-Powered Mock Data Generator

Akhundzada Nijat

Organizational part

Personal goals

My personal goal for this thesis is to develop a practical and innovative tool that directly addresses a common pain point in software development - generating realistic and relationally consistent mock data for testing and development environments. Through this project, I aim to deepen my skills in database schema analysis, AI/ML model training, and CLI tool development.

Time

I plan to dedicate 4-5 hours per week to project development, with more hours during implementation and testing phases.

Compute

I have access to:

- Hp Pavilion (Intel i7, 16GB RAM, 512GB SSD)
- MacBook Pro M1 (8GB RAM, 512GB SSD)

Advisor

Vladislav Goncharenko – Experienced researcher and educator specializing in artificial intelligence, machine learning, and applied computer science. He has extensive academic and industry experience in developing AI-driven solutions and mentoring student projects.

Profiles:

- University Page: https://en.kb-12.com/kb12faces/vlad-goncharenko
- LinkedIn: https://www.linkedin.com/in/vladislav-goncharenko/

Problem statement

In modern software development, generating realistic test data is essential for proper application testing, performance evaluation, and user experience validation. Existing mock data generators

such as Faker produce random values but lack context-awareness, domain specificity, and understanding of database relationships. Developers often spend significant time manually writing seed scripts to ensure data realism, especially in applications with complex schemas and foreign key dependencies. This manual process is error-prone, repetitive, and slows down development. There is a clear need for a tool that can automatically read a project's database schema, understand its structure and relationships, and populate it with realistic, contextually accurate, and domain-specific mock data without manual configuration.

Relevance

This project is highly relevant to the needs of modern software teams, especially in agile and continuous integration/continuous delivery (CI/CD) environments where rapid iteration is critical. By automating schema analysis and realistic data generation, AutoMockAI significantly reduces the time and effort required for test data preparation, enabling developers and QA teams to focus on core functionality instead of repetitive setup tasks. The integration of AI/ML ensures that the generated data is not only syntactically correct but also semantically meaningful and context-aware, making it valuable for realistic testing scenarios, user acceptance testing, and demo environments.

Input and output

Input:

- Database credentials provided via CLI arguments (DB type, host, port, username, password, database name).
- Database schema automatically extracted using ORM reflection.
- AI/ML model files stored locally in the `model/trained_model` directory.

Output:

- Populated database with realistic, relationally consistent mock data.
- AI-generated values for known fields, Faker-generated values for unknown fields.

Metrics

To evaluate the quality and effectiveness of AutoMockAI, the following metrics will be used:

1. **Field Accuracy** – Percentage of generated values matching the expected format/type.
2. **Referential Integrity** – Percentage of valid foreign key references generated.
3. **Realism Score** – Measured through **expert review panels**, where software engineers and domain specialists rate samples on a 1–5 scale based on plausibility, coherence, and domain accuracy (e.g., whether names match cultural norms, whether salaries match realistic ranges). Multiple reviewers' scores are averaged to form the final Realism Score.

4. **Uniqueness** – Percentage of non-duplicate values in unique fields.

Data

The model will be trained using multiple open datasets, merged into a unified training set of `(field_name, type, value)` triples.

Datasets include:

- **Northwind Database** – contains customer, order, and product information with realistic business relationships.
- **HR Employee Dataset** – contains employee names, departments, job titles, and salary ranges.
- **Financial Transactions Dataset** – contains realistic transaction records including amounts, merchants, and timestamps.

These datasets provide a wide variety of field types (text, numeric, categorical, date/time) and real-world distributions, enabling the AI model to learn domain-specific value patterns.

AI/ML Model:

> AutoMockAI uses a **hybrid approach**: a lightweight transformer-based text generation model combined with statistical pattern learners trained on real datasets. The model learns mappings between field names, types, and value distributions (e.g., recognizing that “email” fields map to realistic addresses, “salary” to numeric ranges). For unseen fields or domains, the model generalizes by leveraging semantic similarity between the new field name and previously learned ones, ensuring plausible outputs. When confidence is low, AutoMockAI falls back to Faker to guarantee type correctness.

Links to datasets

- **Northwind Database** – https://github.com/pthom/northwind_psql
- **HR Employee Dataset** – https://www.kaggle.com/datasets/rhuebner/human-resources-data-set
- **Financial Transactions Dataset** – https://www.kaggle.com/datasets/garfield18/credit-card-transactions

Validation

Validation will be performed by:

- Splitting the training data into training and validation sets using an 80/20 split.
- Comparing generated data against the validation set to measure accuracy and realism.

- Manual inspection by domain experts to assess semantic correctness and contextual relevance.
- Verifying referential integrity by checking foreign key relationships after data generation.
- **Generalization Test** – Generating values for unseen fields (fields excluded from training data) to measure how well the model generalizes.

Edge Cases

AutoMockAI incorporates fallbacks and heuristics for rare fields, custom data types, and unusual business rules:

- **Custom fields** (e.g., `loyalty_tier`) → mapped using semantic similarity to known categories.
- **Rare types** (e.g., JSON blobs, geospatial coordinates) → generated using type-specific rules or Faker extensions.
- **Business rules** (e.g., price > 0, end_date > start_date) → enforced via schema-level constraints and post-generation validation checks.

Solutions overview

Faker Library – https://faker.readthedocs.io/
Faker is a Python library for generating fake data such as names, addresses, and emails. It is widely used in development and testing but lacks schema awareness, domain-specific knowledge, and the ability to preserve database relationships.

Mockaroo – https://mockaroo.com/
Mockaroo is a web-based mock data generator that allows custom schema definitions and exports in various formats. However, it requires manual schema setup for each project and does not automatically infer relationships from an existing database.

Synthetic Data Vault (SDV) – https://sdv.dev/
SDV is a framework for generating synthetic data using machine learning models trained on real datasets. While it supports relational data generation, it requires complex configuration and training, making it less convenient for rapid developer workflows.

Experiments

Baseline

Faker-generated data will be used as the baseline. This method produces syntactically correct but context-agnostic values without preserving foreign key relationships or domain-specific constraints.

Baseline metrics and results

Field Accuracy: High for format/type correctness.
Referential Integrity: Low, as foreign keys are not consistently maintained.
Realism Score: Low, since data lacks semantic context and domain-specific patterns.
Uniqueness: Moderate, depending on field type and generator configuration.

Main solution

AutoMockAI – a CLI tool that:

1. Connects to the database and automatically extracts the schema.
2. Uses an AI/ML model trained on real-world datasets to generate context-aware values for recognized fields.
3. Falls back to Faker for unknown fields.
4. Preserves foreign key relationships and enum constraints.

Main model metrics and results

Field Accuracy: High, matching expected formats and types.
Referential Integrity: Very high, due to schema-driven relationship handling.
Realism Score: High, based on domain-expert evaluation.
Uniqueness: High for unique fields.

Code repository

The code will be hosted on GitHub under an MIT license:
https://github.com/nijat-akhundzada/automockai

Usage

Usage
Developers can install AutoMockAI via:

bash
```
pip install automockai
```

Then run:

bash
```
automockai --db-type postgresql --host localhost --port 5432 --user
admin --password pass --database mydb --count 50
```

The tool will:

1. Automatically read the database schema, including tables, columns, data types, and relationships.
2. Use the AI/ML model to generate context-aware, domain-specific values for known fields.
3. Fall back to Faker for unknown or unrecognized fields.
4. Insert the generated, relationally consistent mock data directly into the target database.