Intelligent User Interfaces

Ziad Morsy & Kholoud Jalilati

Magic Wand

Project 1

Project Summary

- 🎯 **Task**: Recognize different pre-defined gestures using trained model on recorded sensor data.
- 🧪 **Data**: Collected with Magic Wand (Arduino Bluno, accelerometer + gyroscope).
- 🧹 **Preprocessing**: Used **pandas** to smooth + interpolate time series sensor data.
- 🧠 **Classifier**: Trained a **RandomForestClassifier** from **scikit-learn** using statistical features.
- 📊 **Evaluation**: Confusion matrix + accuracy from **sklearn.metrics**, visualized with **seaborn**.

Approach

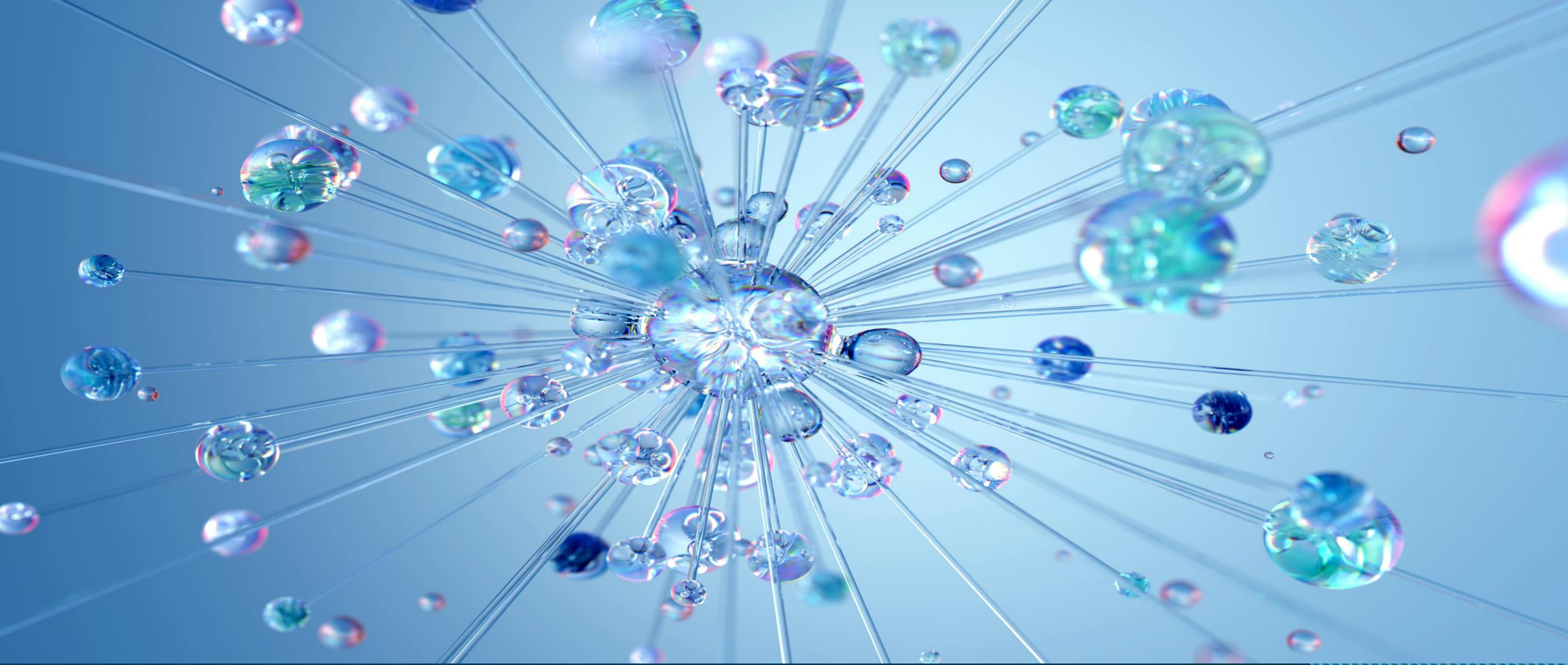
- ✊ Defined gesture vocabulary: Rock, Paper, Scissors.
- 💾 Recorded 100+ CSV files per gesture using the Python recorder.
- 📊 Plotted sensor data to identify outliers and remove faulty recordings.
- 🔍 Cleaned the dataset down to ~100 reliable samples per class.
- ⚠ **Challenge**: Variability in gesture performance led to some inconsistent recordings that needed to be filtered out.

Solution

- 🧹 **Data Cleaning**: Applied smoothing + interpolation to sensor data using custom Python script.
- 📃 Saved cleaned CSVs per gesture and person for reliable model training.
- 🧠 **Feature Extraction**: Calculated mean + standard deviation for each accelerometer and gyroscope axis.
- 🌲 **Classifier**: Trained RandomForestClassifier using scikit-learn.
- 📈 **Evaluation**: Assessed performance using accuracy and confusion matrix.

Wizard Arena

- 🧪 Integrated trained model into the Wizard Arena for real-time gesture recognition.
- 🧝 Users could cast spells with gestures like Rock, Paper, Scissors.
- ⚠️ **Challenge**: Some gestures were harder to detect reliably due to user variation and subtle differences.
- ✅ Final system was responsive and fun to interact with.



LLM Writing Assistant

Project 2

Project Summary

- 🧠 **Goal**: Develop a writing assistant that helps users improve their text.
- ✍️ **Functionality**: Supports grammar correction, rewriting, and summarization.
- 🖥️ Runs locally with no external API calls.

Approach

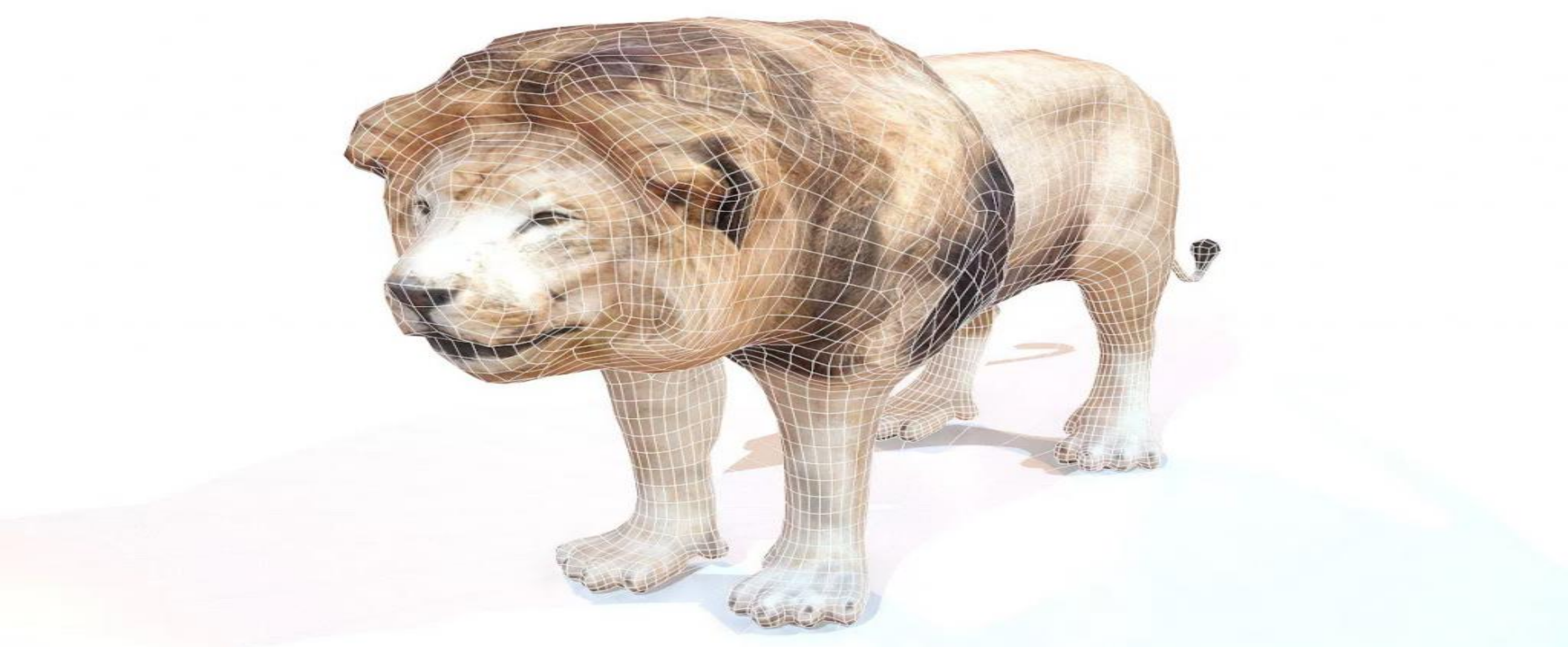
- 🧩 Used Streamlit to build a lightweight, interactive frontend.
- 🔁 Backend handled by FastAPI, connecting user input to the language model.
- 🧠 Chose to integrate llama3:8b locally using Ollama, to avoid external dependencies.
- ⚙️ Implemented distinct prompt templates for each mode: revise, rewrite, summarize.

Challenges

- ⚠ Model sometimes broke on malformed or weird input (e.g., gibberish).
- ❓ If the user input was phrased as a question, the model tried to answer it instead of rewriting.

Solution

- ✍️ Adjusted the prompts to make the model stick to rewriting instead of answering.
- 🔁 Tested different wording to reduce issues with weird input.
- ✅ After some trial and error, the output became more stable.



3D Model Generation

Project 3

Project Summary

- 🎯 Goal: Enable users to generate 3D models via multiple input types.
- ✨ Modes: Prompt-to-model, Sketch-to-model, Image-to-model.
- 🧰 Tech Stack: Streamlit frontend + FastAPI backend.
- 🔌 External APIs: Used GenerIO’s test API for model generation.

Approach

- 🔌 Explored the available GenerIO API routes to understand what each one does.
- 🖼️ Implemented a sketch canvas using Three.js for in-browser drawing.
- 💬 Added an optional prompt input to help improve model quality for sketches.
- 🖼️ For image-to-model, handled image uploads and passed them to the API.

Challenges

- 🔌 API server was sometimes down, which made testing difficult.
- 🧩 Some responses were confusing, either missing data or unclear error messages.
- 🐛 Slower endpoints made it hard to tell if something failed or was just delayed.

Solution

- 🔁 Handled all modes through one shared API route to simplify processing.
- ⏳ Added polling to wait for model generation and avoid broken results.
- 👀 Embedded a viewer to display the generated .glb model directly in the UI.