Team Name: BURAQ
Project Name: QASIM (Quality Assessment of Simulation)

GitHub Repository: A new link has been created for this particular submission, as per the requirements.
Link: https://github.com/Safius-Sifat/AIHackathon-Team-Buraq
Mobile App: The link from which you can download and try out our Solution,
Link: QASIM Link

📄 Technical Documentation

1. Problem Statement

Many students and electronics enthusiasts struggle to understand and solve electrical circuits due to the inability to visualize complex problems and the lack of accessible, user-friendly tools. Traditional circuit simulators often require installation on desktops, rely on expensive software or hardware, and demand significant technical knowledge or large training datasets for accuracy. These limitations create a gap between learning and practical application, especially in resource-constrained environments. To address this, there is a need for a mobile application that can take images of handwritten circuit diagrams, automatically interpret them, simulate the circuit, and display key results such as node voltages and current flow—making circuit analysis more intuitive, portable, and accessible to all learners.

2. Solution Approach

- Solves handwritten electrical circuits using AI and simulation tools. Users upload a photo of a circuit diagram through the app.
- The image is preprocessed (denoised, skeletonized),
- YOLO_element_best.pt: YOLO model for component detection.
- YOLO_text_best.pt: YOLO model for text region detection.
- crnn_inference_model.h5: CRNN model for OCR.
- Postprocessing logic maps connections and generates a SPICE netlist, which is simulated using NGSPICE.
- A FastAPI backend handles the processing, and results like node voltages are visualized in a Flutter-based frontend.

3. AI Model Details

Field	Description
Architecture	Custom-trained YOLOv8 for object detection
Libraries	`ultralytics`, `OpenCV`, `TensorFlow`, `scikit-learn`
Dataset	Our partial data set link - click
Preprocessing	Skeletonization, denoising, and component separation
Post Processing	Clustering, coordinate-based connection logic

4. Deployment Process

- **Server:** FastAPI app hosted at `http://172.105.41.70:8000`
- **Model Execution:** Loaded and run locally during API call
- **Circuit Simulation:** Netlist parsed with `SpiceParser`, simulated with `NgSpiceShared`

Working Prototype of Assigned Challenge

Project Deployment

- **Live API URL:** `http://172.105.41.70:8000`
- **Live API Docs URL:** `http://172.105.41.70:8000/docs`

API Endpoints & Testing Instructions

1. `POST /image-to-netlist`

- **Description:** Accepts a handwritten circuit image and returns the SPICE netlist and voltage results.
- **Request:**

- Method: `POST`
- URL: `http://172.105.41.70:8000/image-to-netlist`
- Body: `multipart/form-data`
  - `file`: image file (e.g., PNG, JPG)

- **Response:**


```
{
   “imageId”:  “b439939f-de68-4a26-a021-0c5a9422e32e”,
  "netlist": "* Circuit Description\nV1 1 0 DC 10V\nR1 1 2 2\nR2 2 0 3\n.END",
  "voltages": {
    "1": 10.0,
    "2": 6.0,
    "0": 0.0
  }
}
```


- **Curl Example:**

```
curl -X POST "http://172.105.41.70:8000/image-to-netlist" \
  -H "accept: application/json" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@circuit.jpg"
```

---

2. `POST /chat-about-circuit`

- **Description:** Asks a question about a previously processed circuit image.
- **Request:**
  - Method: `POST`
  - URL: `http://172.105.41.70:8000/chat-about-circuit`
  - Body: `application/json`
    - `question`: your question

        - `image_id`: UUID of the circuit image

- **Response:**

```
{
  "response": "markdown body"
}
```

- **Curl Example:**

```
curl -X POST "http://172.105.41.70:8000/chat-about-circuit" \
  -H "accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{"question": "What does this circuit do?", "image_id": "abc123-uuid"}'
```

⚙️ Local Setup Guide & Deployment Process

Prerequisites

- Python 3.12
- pip
- Git

This guide walks you through setting up both the backend (FastAPI) and frontend (Flutter app) on your local machine.

📁 Folder Structure

```
project-root/
│
├── backend/        # Contains FastAPI backend code
│   ├── main.py
│   ├── solve.py
│   └── ...
│
├── images/         # Folder to store input/output images (must be created manually)
```

```
│
├── models/          # Pre-trained models for inference
│   ├── YOLO_element_best.pt
│   ├── YOLO_text_best.pt
│   └── crnn_inference_model.h5
│
├── frontend/        # Contains Flutter mobile app code
│   └── ...
```

⚙️ Backend Setup (FastAPI)

1. **Navigate to the backend folder:**

```
cd backend
```

2. **Install required system packages:**

```
sudo apt update
sudo apt install libngspice0-dev
```

3. **Create a virtual environment and activate it:**

```
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

4. **Install Python dependencies:**

```
pip install fastapi uvicorn opencv-python-headless google-generativeai numpy tensorflow ultralytics
scikit-learn scikit-image pillow pyspice
```

5. **Create an `images/` folder (if not already created):**

```
mkdir images
```

6. **Ensure the `models/` folder contains the following files:**

- `YOLO_element_best.pt`
- `YOLO_text_best.pt`
- `crnn_inference_model.h5`

7. **Run the FastAPI server:**

```
uvicorn main:app --reload
```

Your backend will now be available at: http://127.0.0.1:8000

Frontend Setup (Flutter)

1. **Navigate to the frontend folder:**

```
cd frontend
```

2. **Ensure Flutter is installed.**
   If not, follow the instructions here: Flutter Installation Guide

3. **Install project dependencies:**

```
flutter pub get
```

4. Change api base url in api.dart to your backend url.

5. **Run the app on a connected device or emulator:**

```
flutter run
```

The app will launch and communicate with the backend you just hosted.

🚀 Future Improvements

- Improve accuracy by expanding training dataset with more circuit types
- Add support for capacitors, inductors, and other elements.
- Enable circuit editing and simulation inside the web UI
- Add error handling for malformed diagrams
- Introduce OCR for text components (e.g., voltage values)