AI-GCode-Generation

MII - Project 2

Student: Vatsal Mahajan
Matriculation No.: 7025694

Course of Studies: Master Industrial Informatics

Supervisors: M.Eng. Jeffrey Wermann

Submission date: September 11, 2025

University of Applied Sciences Emden/Leer · Faculty of Technology · Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · `http://www.hs-emden-leer.de`

ASSERTATION OF RIGHTS

The work presented in this thesis contains confidential / commercially applicable information, the rights of which lie outside the Hochschule Emden/Leer, University of Applied Sciences. It is only to be made available to those persons involved in the examination procedure, who are hereby made aware of their confidentiality obligation.

As far as my rights are affected, I agree that the work presented in this project can be made openly available to any and all members of the Hochschule Emden/Leer, University of Applied Sciences for further studies / teaching / or for research.

DECLARATION OF AUTHORSHIP

I hereby declare that I, the undersigned, am the sole author of this document. All sources consulted for this document have been listed; all quotations from and references to these sources have been properly cited and included in chapter notes and in the list of references. No version of this document, either whole or in part, has been used to achieve an academic degree or any other examination.

I understand that any false statements made in this declaration may be punishable by law.

[illegible]

Vatsal Mahajan
28.09.2025, Emden

Contents

1 Introduction

1.1 Motivation

The Digital Factory at the University of Applied Sciences Emden/Leer demonstrates Industry 4.0 technologies [**PlattformI40:WhatIsIndustrie40**] through modular setups that integrate various production units. Among these is the Laser Engraver Module, which can work in combination with other modules such as robotic arms and conveyor systems to form flexible manufacturing processes.

While the laser engraver is capable of producing high-quality results, preparing engraving jobs remains a manual and time-consuming task. In particular, creating G-code from business card designs requires the operator to manually extract text, logos, or QR codes, recreate layouts, and then generate the engraving file. This lack of automation limits the module's efficiency and repeatability within a smart factory setting.

To overcome these limitations, this project introduces an AI-based pipeline that can automatically generate laser-ready G-code directly from a photographed business card. By combining computer vision, OCR, layout analysis, and vector graphics generation, the system reduces manual effort, ensures consistency, and enables the Laser Engraver Module to operate more autonomously within the Digital Factory.

1.2 Goals and Requirements

The primary goal of this project is to develop an automated workflow that converts a photographed business card into laser-ready G-code for the Laser Engraver Module in the Digital Factory [**PlattformI40:WhatIsIndustrie40**]. This supports faster, more reliable, and repeatable engraving tasks without extensive manual preparation.

To achieve this, the system must meet the following requirements:

- **Information extraction:** Automatically detect and read text, logos, QR codes, and other visual elements from the card using computer vision and OCR.
- **Layout preservation:** Convert detected elements into millimetre-accurate positions on a standard card size (85 $\times$ 54 mm).
- **SVG generation and editing:** Assemble an SVG design that represents the card and allow optional adjustments through command-based or natural language instructions.

- **Rasterization and binarization:** Produce clean black-and-white images from the SVG to ensure high-contrast engraving quality.
- **G-code generation and preview:** Translate the processed image into optimized G-code and provide a preview of the laser paths for verification.
- **Workflow integration:** Orchestrate all components within a modular LangGraph pipeline, ensuring scalability and smooth interaction between agents.

By fulfilling these requirements, the project establishes an end-to-end system that improves usability, reduces errors, and enhances the integration of the Laser Engraver Module in the Digital Factory.

1.3 Documentation Structure

This report is divided into six main chapters:

- **Chapter 1 – Introduction:** Outlines the motivation, goals, requirements, and structure of the project documentation.
- **Chapter 2 – Basic Concepts:** Explains the theoretical background and technologies applied, such as OCR, computer vision, SVG graphics, G-code generation, and LangGraph orchestration.
- **Chapter 3 – Implementation:** Describes the step-by-step development of the system, based on the implemented modules and agents.
- **Chapter 4 – Validation of Requirements:** Evaluates how effectively the developed system meets the defined goals and requirements.
- **Chapter 5 – Conclusion:** Summarises the key results and contributions of the project.
- **Chapter 6 – Future Scope:** Highlights potential improvements and directions for further research and development.

2 Basic Concept

2.1 Laser Engraving

Laser engraving is a subtractive process that uses a focused laser beam to remove material from a surface, creating permanent marks without physical contact or tool wear. There are two principal techniques to generate these engravings: vector engraving and raster engraving. [**HaiTechLasers:MasterGuideLaserEngraving**]

- **Vector Engraving:** It relies on drawing continuous paths along predefined shapes, such as lines, curves, or outlines as shown in the figure 2.1. The laser follows these vector paths directly, similar to how a pen moves when drawing. This method is efficient for simple geometries such as borders, logos, or line art, but becomes less suitable for dense areas of filled text or images. [**CNCSourced:RasterVsVectorEngravingGuide**] [**xTool:RasterVsVectorEngraving**]

- **Raster Engraving:** by contrast, treats the design as a pixel-based image. The laser scans the surface line by line (similar to the operation of an inkjet printer), switching between ON and OFF states according to the black-and-white pattern of the image as shown in the figure 3.6. This method is more appropriate for detailed text, photographs, or shaded regions, where filled areas need to be represented precisely. [**CNCSourced:RasterVsVectorEngravingGuide**] [**xTool:RasterVsVectorEngraving**]

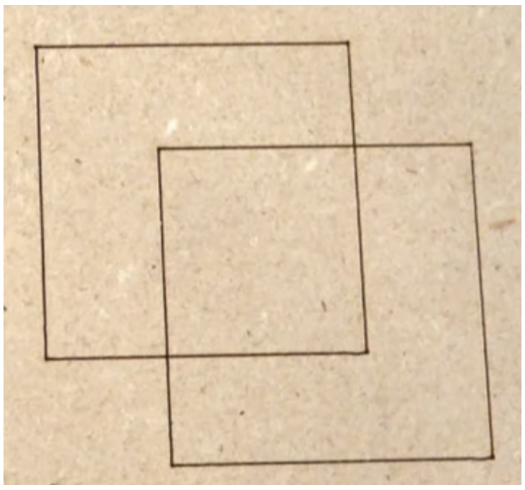


Figure 2.1: Vector Engraving

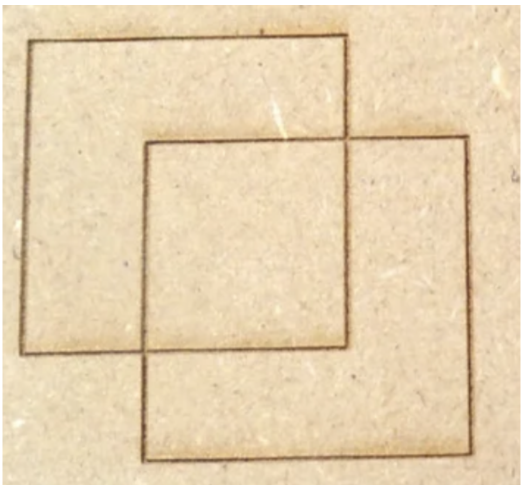


Figure 2.2: Raster Engraving

A widely used method in raster engraving is the scanline principle, whereby the laser head moves consistently across each row (scanline) and shifts down incrementally after each pass. It delivers high fidelity engraving of detailed images or filled patterns, though it is typically slower compared to vector engraving. [**OneLaser:RasterVsVectorEngraving**]

Raster engraving is especially apt for business card reproduction, capturing intricate details like logos, text, and QR codes with high accuracy, whereas vector engraving may only be practical for outlining or borders. By combining both methods, the system achieves both precision and flexibility, accommodating complex visual content and simple line art alike.

2.2 Core Concepts

2.2.1 Optical Character Recognition (OCR)

OCR converts images of printed or handwritten text into machine-readable text, enabling digital editing, indexing, or downstream processing.

In this project, OCR extracts name, contact details, and address from the scanned card. Preprocessing ensures clean input, while post-processing (e.g., pattern-based classification) groups text into semantic categories like "email" or "phone", which is a method commonly used in business card recognition tools. [**MadanKumar:BusinessCardOCR2019**]

2.2.2 Computer Vision & Layout Detection

Beyond raw text extraction, correctly identifying regions such as logos or QR codes relies on layout detection and classification. Vision-language models like Qwen2.5-VL can detect and localise multiple heterogeneous elements in a single pass. This object-level detection ensures that visual components are correctly distinguished before vectorisation. [**Wikipedia:DocumentLayoutAnalysis**]

Our system, layout analysis ensures that each detected element is recognized and mapped to accurate positions in the SVG design, which is essential for precise engraving control.

2.2.3 Scalable Vector Graphics (SVG)

SVG is an XML-based vector image format representing graphics as scalable paths and shapes, ideal for precise positioning and editing.

SVGs are pivotal in our project because they preserve resolution independence and allow command-based or natural-language editing before engraving. Recent research demonstrates how modern models can generate high-quality and semantically-rich SVGs directly from images or text. [**StarVector:Arxiv2312.11556v4**]

2.2.4 Rasterisation and Binarisation

Rasterisation converts vector graphics into pixel-based images, while binarisation reduces them to black-and-white bitmaps suitable for laser engraving. These steps ensure high-contrast and noise-free imagery optimising it for scanline-based laser execution.

2.2.5 G-code Generation

G-code is a numerical control language for directing machine tool movement (e.g., lasers or 3D printers). In laser engravers it instruct how to move and when to toggle the laser. It includes commands like G0 (rapid move) and G1 (controlled engraving move) define the path and engraving speed, while laser activation (e.g., M3, S) controls intensity. [**Yang:SVGtoGcode2021**]

In our implementation, a scanline algorithm parses raster bitmaps into efficient G-code sequences. The generated G-code enables control over movement and laser intensity, followed by a preview for validation before actual engraving.

2.3 System Data Flow

The system's data flow outlines the sequential transformation from a photographed business card to laser-ready G-code. Structuring the workflow in clearly defined stages ensures both modularity and interpretability—principles central to robust system design in automation contexts.

2.3.1 Overview of the Pipeline

The system follows a structured, multi-stage process:

- **Image Acquisition:** Capture or obtain a digital image of the business card.
- **OCR & Layout Detection:** Visual content—including text, logos, QR codes, and embedded elements, is detected using OCR and vision-language models, which provide semantic and positional information. These models increasingly replace multi-step pipelines by combining detection and recognition in one pass.
- **SVG Synthesis:** Detected elements are rendered into an SVG format, respecting physical dimensions (e.g., 85 $\times$ 54 mm) and spatial relationships. SVG's vector nature ensures scale independence and editing flexibility.
- **Optional Editing:** Users may adjust the design via structured commands or natural language through an LLM, ensuring human oversight while preserving automation.

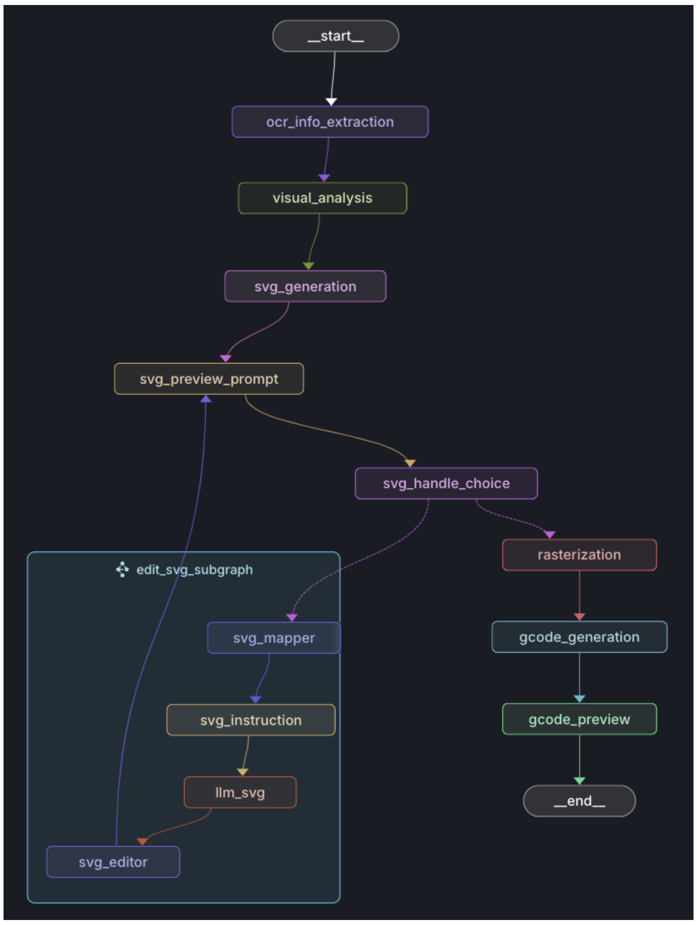


Figure 2.3: Langgraph Workflow

- **Rasterization and Binarization:** The SVG is converted into a high-contrast black--and-white bitmap, suitable for precise laser engraving.
- **G-code Generation:** Using a scanline strategy, the raster image is translated into G-code (e.g., G0/G1 commands with laser ON/OFF), encoding laser movement and power control.
- **Preview & Execution:** A visual preview of the G-code toolpath is rendered for validation before being sent to the Laser Engraver Module in the Digital Factory.

2.3.2 Diagrammatic Representation

Above, you'll see an illustrative pipeline diagram 2.3 that represents this workflow:

- **Start:** Business card image input
- **Next:** OCR & layout detection

- **Then:** SVG generation → optional editing
- **Following:** Rasterization → binarization
- **Then:** G-code generation
- **End:** Preview → Laser engraver

This diagram helps readers visualize how data transitions across stages and underscores the modular orchestration via agent-based design.

2.4 Agents & LangGraph Orchestration

In modern AI systems, particularly those involving sequential or multi-step tasks, it's advantageous to structure the workflow as a collection of specialized agents, each responsible for a well-defined task, rather than a monolithic process. This modular approach enhances clarity, maintainability, and scalability, while enabling dynamic control flows and human oversight. [**Chase:ThinkAboutAgentFrameworks2025**]

2.4.1 Agents and Workflow Decomposition

An agent in this context is an autonomous component—often powered by a large language model, that processes input, performs a task, and passes output forward. Multi-agent architectures allow [**Chase:ThinkAboutAgentFrameworks2025**]:

- **Specialization:** Each agent focuses on a single responsibility, improving clarity and testability.
- **Modularity:** Agents can be developed, tested, and iteratively improved in isolation.
- **Controlled orchestration:** The flow between agents can be governed dynamically, based on state or external feedback, facilitating flexibility in the face of uncertain inputs

2.4.2 Introduction to LangGraph

LangGraph is a graph-based orchestration framework built to manage agentic systems. It combines declarative graph definitions with imperative node logic, enabling both predictability and flexibility. Key features include:

- **Nodes** represent discrete agent tasks or decision points
- **Edges** define transitions, which can be condition-driven.

- **State persistence** allows workflows to pause, resume, or even support **human-in-the-loop** interventions, making it applicable to interactive systems where approval or corrections may be necessary.
- LangGraph supports **streaming updates**, memory handling, and integrates with tools like LangSmith for debugging and visualization.

2.4.3 Application in our Project

This project leverages the agentic paradigm and LangGraph orchestration, fig. 2.3 to construct a clean, modular pipeline:

- **OCR Agent:** Extracts textual information from the input image.
- **Visual Analysis Agent:** Detects layout elements like logos, QR codes, and text blocks.
- **SVG Generation Agent:** Assembles detected components into an editable vector design.
- **Editing Agent (LLM-SVG):** Receives human or language-based inputs to adjust layout or content.
- **Rasterisation Agent:** Converts SVG to a clean, black-and-white bitmap.
- **G-code Agent:** Translates scanlines into laser movement instructions.
- **Preview Agent:** Generates a visual tool-path preview for validation.

Each agent processes only its designated output and hands off structured state to the next, following the pipeline-of-agents pattern [**Honchar:PipelineOfAgents2025**]. This design allows:

- **Isolated development and debugging** of agents.
- **Human oversight** at the SVG editing or preview stages.
- **Stateful execution** the workflow can pause for edits or resume after failures.
- **Traceability** the graph structure enables clear tracking of data and decisions across stages.

2.4.4 Benefits

Using LangGraph orchestration in this project provides specific advantages for automating the laser engraving pipeline:

- **Predictability and Agency:**

  The engraving pipeline requires deterministic steps (e.g., rasterisation must always follow SVG generation) but also benefits from flexible decision points (e.g., when a user chooses to edit the SVG or proceed directly to G-code). LangGraph ensures the overall flow is predictable, while still allowing human-in-the-loop interventions where agency is needed.

- **Fault Tolerance and Robustness:**

  In practice, OCR errors or misaligned layout detection can occur when processing diverse business cards. LangGraph's persistent state allows the workflow to pause, request corrections (e.g., manual confirmation of text or layout), and then resume without restarting the entire process. This robustness is essential for maintaining accuracy across different card designs.

- **Alignment with Agentic AI Principles:**

  The project demonstrates how modular agents, OCR, visual analysis, SVG generation, editing, rasterisation, and G-code preview, can be coordinated into a coherent workflow. This mirrors current research trends in agentic AI, where specialized modules are orchestrated to solve complex, multi-step tasks reliably in industrial contexts.

2.5 Key Tools and Libraries Used

The implementation of the AI-GCode-Generation project relies on a combination of software frameworks, libraries, and AI models, each serving a specific role in the pipeline. By integrating these tools, the system achieves modularity, robustness, and adaptability within the Digital Factory's Laser Engraver Module.

2.5.1 Programming Environment

- `Python 3.12`

  The core programming language and environment for implementing the system. Python's extensive ecosystem of libraries for computer vision, vector graphics, and machine learning makes it highly suitable for developing AI-assisted engraving workflows.

2.5.2 Workflow Orchestration

- `LangGraph (langgraph-cli, langgraph-api)`

  Provides the orchestration layer for structuring the pipeline into modular agents. It defines the graph-based workflow, connects nodes representing tasks (OCR, SVG generation, editing, rasterisation, etc.), and ensures predictable state transitions with support for human-in-the-loop editing. This makes the system both scalable and extensible in the context of the Digital Factory

2.5.3 Computer Vision and Image Processing

- `OpenCV (opencv-python)`

  Used in modules such as `visual_analysis_agent.py` and `rasterization.py` for image preprocessing, layout analysis, and bitmap conversion. OpenCV enables operations like contour detection, grayscale transformations, and QR code recognition.

- `NumPy`

  Provides efficient array and matrix operations required in rasterisation and scanline generation. It is also used in processing pixel data before translation into G-code.

- `Matplotlib`

  Applied for debugging and visual verification, e.g., plotting bounding boxes during layout analysis or displaying rasterised images before engraving.

2.5.4 Vector Graphics Handling

- `svgwrite`

  Used in `svg_agent.py` to generate SVG files programmatically. It enables precise positioning of text, logos, and QR/NFC elements in millimetre coordinates, ensuring layout fidelity.

- `svgpathtools`

  Provides geometric path manipulation for SVG elements, particularly useful when refining coordinates and handling transformations within the generated designs.

- `CairoSVG`

  Converts generated SVG files into PNG images, which serve as the input for binarisation and engraving preparation.

2.5.5 G-code Generation

- `svg2gcode`

  A dedicated library that translates vector paths into G-code. In this project, it is integrated with a custom rasterisation module to produce scanline engraving paths, where the laser toggles ON for black pixels and OFF for white pixels

- `Custom G-code Generator`

  Used in `gcode_agent.py`, extends beyond svg2gcode by implementing raster scanline algorithms, zig-zag motion control, and feedrate tuning for the laser engraver.

2.5.6 AI Models

- `Qwen2.5-VL`

  A vision-language model used in `ocr_agent.py` and `visual_analysis_agent.py` to detect layout elements (logos, QR codes, NFC chips, text regions) and extract structured data such as names, phone numbers, and email addresses. This enables semantic interpretation of card images

- `Mistral 7B (via OpenRouter)`

  Employed in `llm_svg_agent.py` to interpret `natural language editing commands` (e.g., "move logo to top-right" or "replace title with 'Senior Engineer'") and convert them into structured SVG edit instructions. This allows intuitive, user-friendly interactions

2.5.7 User Interaction and Preview

- `Tkinter`

  A lightweight GUI toolkit used in `svg_preview_agent.py` and `gcodePreview_agent.py`. It provides zoomable preview windows for both SVG and G-code, enabling the user to validate layouts and engraving toolpaths before final execution.

2.5.8 Supporting Libraries

- `pydantic`

  Used for data validation and structured JSON handling, ensuring that extracted OCR and layout data conforms to expected formats.

- `python-dotenv`

  Handles environment variables, storing API keys and configuration securely (e.g., Fireworks API for OCR, OpenRouter API for LLM).

- `python-dotenv`

  Handles environment variables, storing API keys and configuration securely (e.g., Fireworks API for OCR, OpenRouter API for LLM).

- `qrcode, pyzbar` Support detection and embedding of QR codes in the pipeline, ensuring scannable codes are correctly reproduced on engraved cards.

3 Implementation

This chapter presents the technical realisation of the AI-GCode-Generation project, describing how the pipeline was implemented in practice. The section proceeds from environment setup and installation of libraries, through orchestration of agents, to the execution of the complete workflow. Selected code examples are included where they are essential for understanding.

3.1 Project Setup and Installation

The project was implemented in Python 3.12, making use of specialized libraries for computer vision, SVG handling, and G-code generation. To ensure reproducibility, a `requirements.txt` file was prepared containing all necessary dependencies.

3.1.1 Clone Repository

The codebase is organized as a modular agent-based system, accessible through the project repository. To begin, the repository is cloned locally 3.1:

Listing 3.1: Clone Repository

```bash
#!/bin/bash
# Clone the Repository
git clone https://github.com/mahajan-vatsal/AI-GCode-
    Generation.git
cd AI-GCode-Generation
```

3.1.2 Python Environment

A dedicated virtual environment ensures isolation and compatibility 3.2 :

Listing 3.2: Create Virtual Environment

```bash
#!/bin/bash
# Create the Virtual environment
python3.12 -m venv venv
source venv/bin/activate   # macOS/Linux
venv\Scripts\activate      # Windows
```

3.1.3 Install Dependencies

The required libraries are installed from the provided `requirements.txt` file 3.12 :

Listing 3.3: Install Dependencies

```
#!/bin/bash
# Install required dependencies
pip install -r requirements.txt
```

This installs core packages including LangGraph, OpenCV, svgwrite, cairosvg, svg2gcode, and AI model connectors such as OpenRouter and Fireworks APIs.

3.1.4 Listing APIs

API keys are managed via a .env file to ensure secure access to external models 3.4 :

Listing 3.4: APIs Listing

```
#!/bin/bash
# Listing all the necessaries APIs
# Fireworks (OpenAI) for OCR and layout detection
FIREWORKS_API_KEY=your_fireworks_key_here
# OpenRouter (Mistral) for generating SVG edit commands
OPENROUTER_API_KEY=your_openrouter_key_here
#Langgraph for defining the Workflow
LANGCHAIN_PROJECT=AI-GCode-Generator
export LANGCHAIN_API_KEY=your_langchain_key_here
export LANGCHAIN_ENDPOINT=https://api.smith.langchain.com
LANGSMITH_TRACING=true
LANGSMITH_PROJECT=AI-GCode-Generator
```

3.2 Running the LangGraph Workflow

The orchestration of the pipeline is managed by `LangGraph`, which interprets the defined workflow in `main_graph.py` and `subgraph.py`

3.2.1 Start LangGraph Server

LangGraph provides a local development server for executing workflows 3.5 :

Listing 3.5: Langgraph Server

```
#!/bin/bash
# Start the Langraph Server
langgraph dev
```

This command launches the workflow as defined in `langgraph.json`, where the entry point `main_graph.py:graph_app` specifies the top-level pipeline.

3.2.2 Workflow Execution

Once the server is active, the workflow can be initiated with an input business card image (e.g., `.jpg` or `.png`) 3.6 as show in fig. 3.1. . The pipeline then proceeds through the following stages.

1. OCR and Layout Detection
2. SVG Generation
3. SVG Editing and Preview
4. Rasterisation and Binarisation
5. G-code Generation and Preview

Listing 3.6: Sample Input

```
        {"image_path": "samples/business_card3.png"}
```

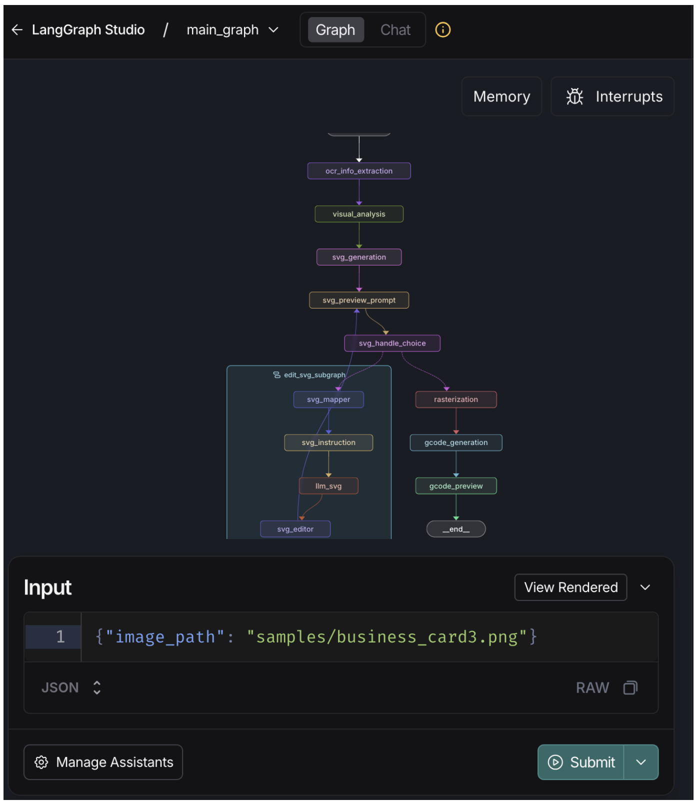
3.3 Agent-Level Implementation

3.3.1 OCR Agent

- **Purpose**: Extracts structured textual data (names, phone numbers, emails, addresses) from the card image using Fireworks API (Qwen2.5-VL).
- **Implementation:** Defined in `ocr_agent.py`, it preprocesses the image via OpenCV and invokes the vision-language model for recognition.
- **Output:** JSON structure with extracted text fields as shown in fig. 3.2 .

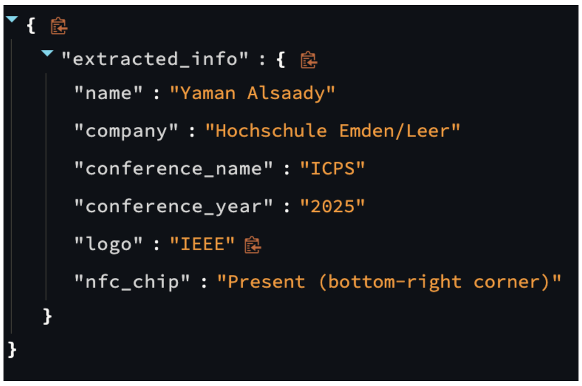
3.3.2 Visual Analysis Agent

- **Purpose:** Detects logos, QR codes, NFC chips, and text block bounding boxes.
- **Implementation:** Uses OpenCV contour analysis and Qwen2.5-VL for semantic enrichment in `visual_analysis_agent.py`.
- **Output:** Bounding box coordinates converted to millimetre space (85 $\times$ 54 mm) 3.3.



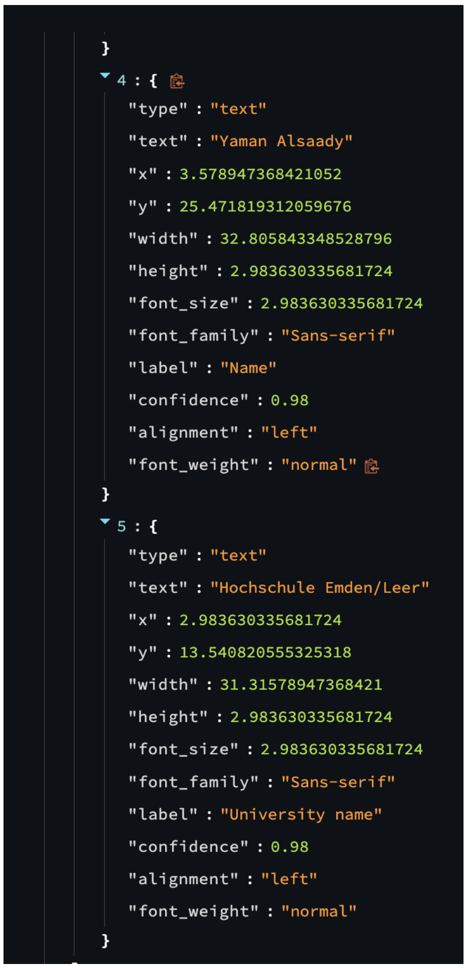
```
{"image_path": "samples/business_card3.png"}
```

Figure 3.1: Sample Input



```
{
  "extracted_info" : {
    "name" : "Yaman Alsaady"
    "company" : "Hochschule Emden/Leer"
    "conference_name" : "ICPS"
    "conference_year" : "2025"
    "logo" : "IEEE"
    "nfc_chip" : "Present (bottom-right corner)"
  }
}
```

Figure 3.2: Node OCR JSON Output



```
}
4 : {
   "type" : "text"
   "text" : "Yaman Alsaady"
   "x" : 3.578947368421052
   "y" : 25.471819312059676
   "width" : 32.805843348528796
   "height" : 2.983630335681724
   "font_size" : 2.983630335681724
   "font_family" : "Sans-serif"
   "label" : "Name"
   "confidence" : 0.98
   "alignment" : "left"
   "font_weight" : "normal"
}
5 : {
   "type" : "text"
   "text" : "Hochschule Emden/Leer"
   "x" : 2.983630335681724
   "y" : 13.540820555325318
   "width" : 31.31578947368421
   "height" : 2.983630335681724
   "font_size" : 2.983630335681724
   "font_family" : "Sans-serif"
   "label" : "University name"
   "confidence" : 0.98
   "alignment" : "left"
   "font_weight" : "normal"
}
```

Figure 3.3: Node Visual JSON Output

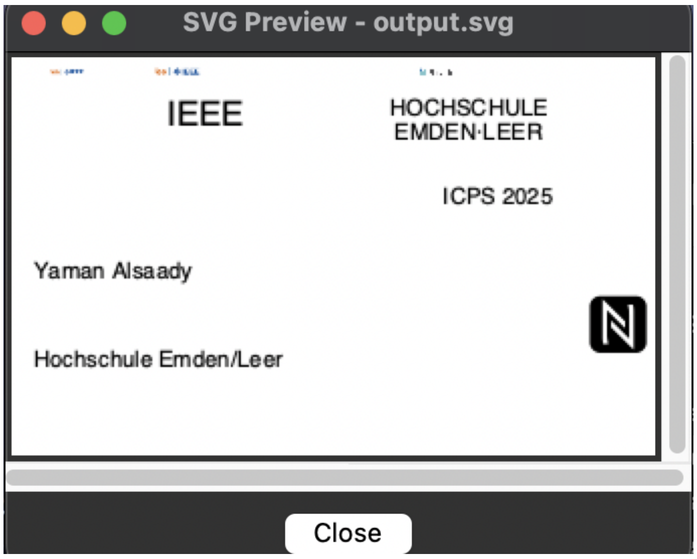


Figure 3.4: Node SVG Output

3.3.3 SVG Agent

- **Purpose:** Generates a Scalable Vector Graphics (SVG) file representing the card layout.
- **Implementation:**
  - Text placed using `<text>` elements.
  - Logos or QR codes embedded with `<image>` or `<path>`.
  - Coordinates flipped to match SVG's top-left coordinate system.
- **Library Used:** `svgwrite` for programmatic construction.
- **Output:** `output.svg` 3.4.

3.3.4 Editing & Preview Agents

- **LLM-SVG Agent (`llm_svg_agent.py`):**
  - Converts natural language edit commands (e.g., "Replace 'Yaman' with 'Vatsal'. or add_text tagline at x=10 y=6 text='Created by Vatsal' size=3.2") into structured SVG modifications.
  - Uses Mistral 7B via OpenRouter.
- **SVG Editor Agent (`svg_editor_agent.py`):**

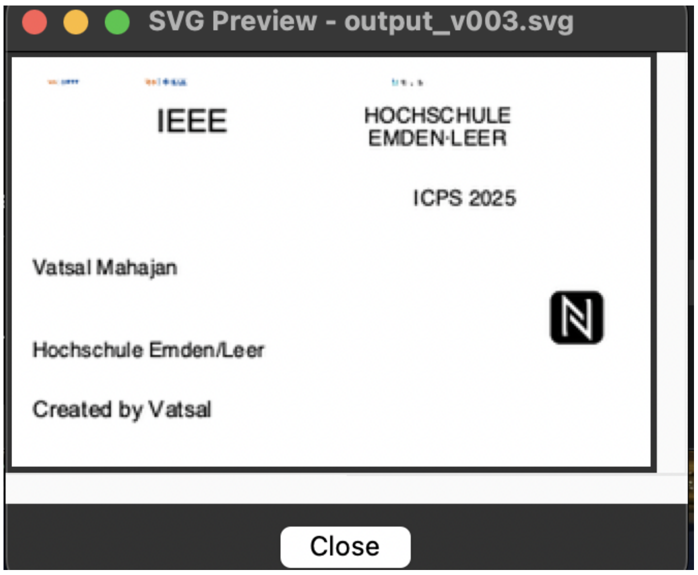


Figure 3.5: Node SVG-Edited Output

   - Applies parsed commands (move, replace, delete, add_text).

- **SVG Preview Agent** (`svg_preview_agent.py`):
   - Provides GUI preview of the card in Tkinter, enabling human-in-the-loop validation.

- **Output**
   - After applying the editing instructions the SVG file looks like 3.5.

3.3.5 Rasterisation Module

- **Purpose:** Converts SVG into a high-resolution PNG, then applies binarisation to produce a black-and-white engraving-ready bitmap.
- **Implementation:** `rasterization.py` uses CairoSVG + OpenCV for conversion.
- **Output:** `output_bw.png` 3.6.

3.3.6 G-code Agent & Preview

- **Purpose:** Translates the binarised bitmap into scanline G-code.
- **Implementation:**
   - `gcode_agent.py` generates line-by-line motion instructions (G0, G1, laser ON/OFF).

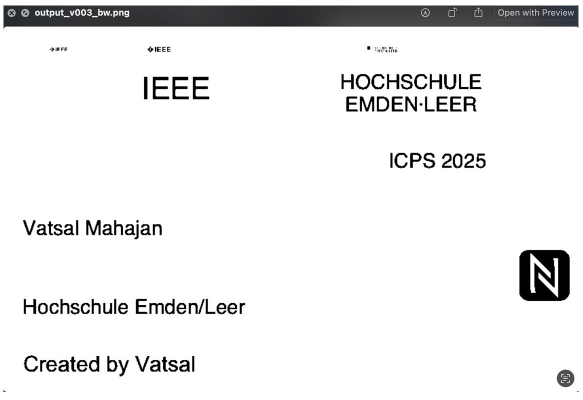


Figure 3.6: Node Rasterisation Output

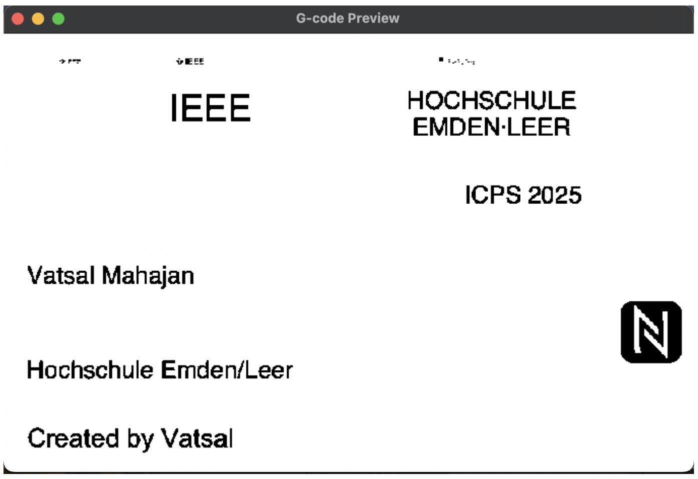


Figure 3.7: Node Gcode Preview Output

  - Raster scanning implemented as zig-zag passes for efficiency.

- **Preview:** `gcodePreview_agent.py` displays toolpaths with Tkinter as shown in fig 3.7.

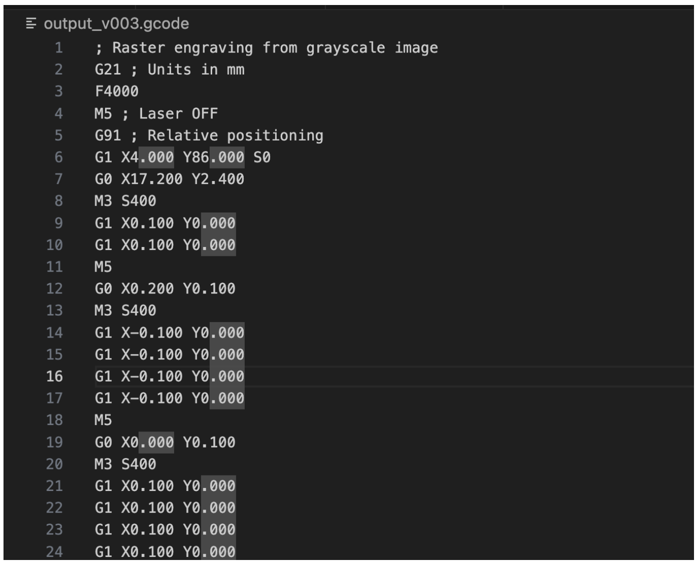
- **Output:** `output.gcode` 3.8.

3.3.7 Human-in-the-Loop Integration

- At two stages, the system pauses for optional human input:

  1. **User Input** – Asking from Users whether they wanted to edit the svg or proceed to generate GCode 3.9.

  2. **SVG Editing** – User may issue natural language commands to reposition or



```
output_v003.gcode
; Raster engraving from grayscale image
G21 ; Units in mm
F4000
M5 ; Laser OFF
G91 ; Relative positioning
G1 X4.000 Y86.000 S0
G0 X17.200 Y2.400
M3 S400
G1 X0.100 Y0.000
G1 X0.100 Y0.000
M5
G0 X0.200 Y0.100
M3 S400
G1 X-0.100 Y0.000
G1 X-0.100 Y0.000
G1 X-0.100 Y0.000
G1 X-0.100 Y0.000
M5
G0 X0.000 Y0.100
M3 S400
G1 X0.100 Y0.000
G1 X0.100 Y0.000
G1 X0.100 Y0.000
G1 X0.100 Y0.000
```

Figure 3.8: Node Gcode Generate Output

replace elements 3.10.

- This hybrid approach combines automation for efficiency with manual oversight for reliability, aligning with best practices in AI-assisted manufacturing.

3.4 Web User Interface Implementation

3.4.1 Purpose of the Web UI

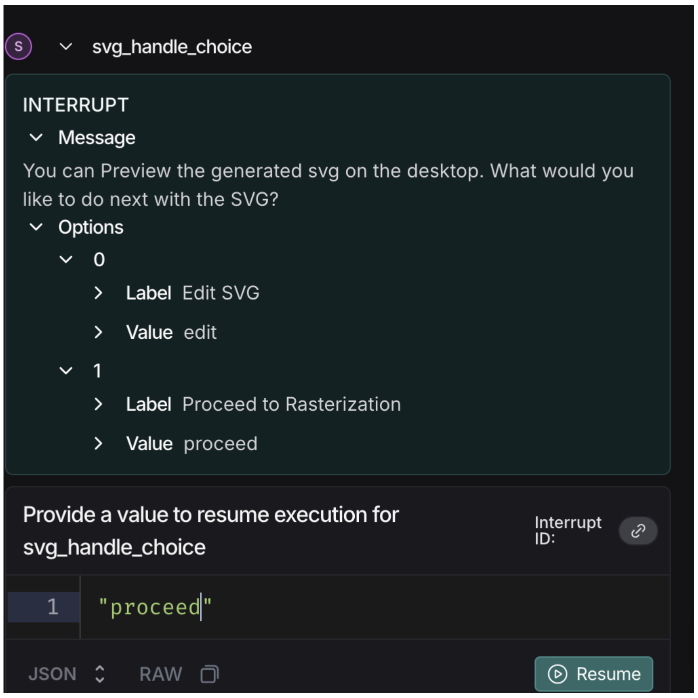
While the backend pipeline orchestrates OCR, layout analysis, SVG generation, rasterisation, and G-code creation, it is equally important to provide users with an accessible interface. To address this, a lightweight web-based user interface (UI) was developed, enabling interaction with the system without requiring direct command-line operations. This UI lowers the technical barrier for end-users and aligns with human-machine interaction principles commonly adopted in digital manufacturing environments.

3.4.2 Tools and Frameworks

The web UI was built using:

- `streamlit.py`: Provides a rapid, Python-based framework for building interactive web applications. It enables file upload, dynamic visualisation, and step-by-step user workflows .

- `server_api.py`: Implements REST-style endpoints to connect the front end with the



svg_handle_choice

INTERRUPT

Message

You can Preview the generated svg on the desktop. What would you like to do next with the SVG?

Options

0

Label Edit SVG

Value edit

1

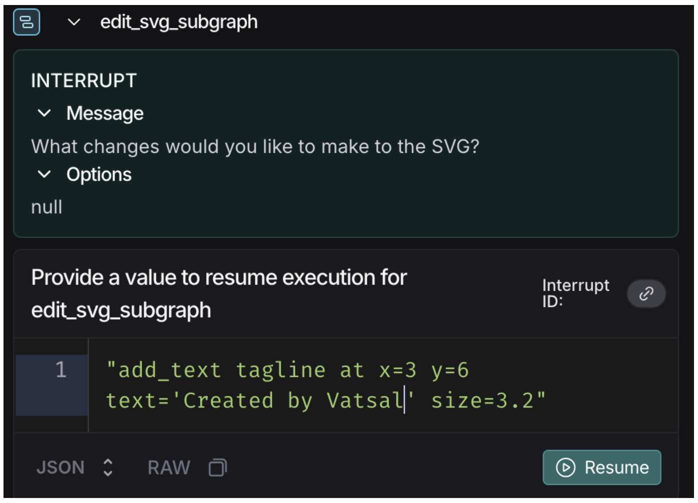
Label Proceed to Rasterization

Value proceed

Provide a value to resume execution for svg_handle_choice

```
"proceed"
```

Figure 3.9: Asking User choice



edit_svg_subgraph

INTERRUPT

Message

What changes would you like to make to the SVG?

Options

null

Provide a value to resume execution for edit_svg_subgraph

```
"add_text tagline at x=3 y=6
text='Created by Vatsal' size=3.2"
```

Figure 3.10: Instruction for editing

backend agents. User actions (e.g., upload or edit requests) are translated into API calls.

- `server_hmi.py`: Functions as the Human-Machine Interface (HMI) layer, bridging user interactions with backend orchestration.
- LangGraph Integration: The UI triggers LangGraph nodes to perform OCR, SVG editing, and G-code generation, ensuring a consistent workflow across both backend automation and frontend interactivity

3.4.3 Advantages of the Web UI

- **Accessibility:** Users without programming expertise can run the full engraving pipeline.
- **Transparency:** Intermediate previews (SVG, raster image, G-code) are exposed, enhancing trust and reducing risk of errors.
- **Integration with Digital Factory:** The UI demonstrates how an AI-driven engraving module can be operated as part of a larger smart factory setup, with clear potential for extension to mobile or remote control interfaces

3.4.4 Triggering the Web UI

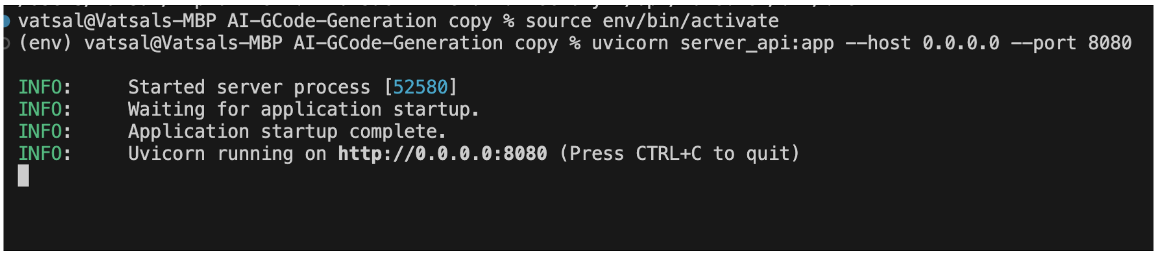
To enable accessibility, the Web UI can be triggered directly from the project environment without requiring complex setup. After installing dependencies and configuring API keys in the `.env` file (see Section 3.1.3), the following commands launch the user interface:

Start the Backend API Server

Listing 3.7: Start backend Server

```
uvicorn server.server_api:app --host 0.0.0.0 --port 8080
```

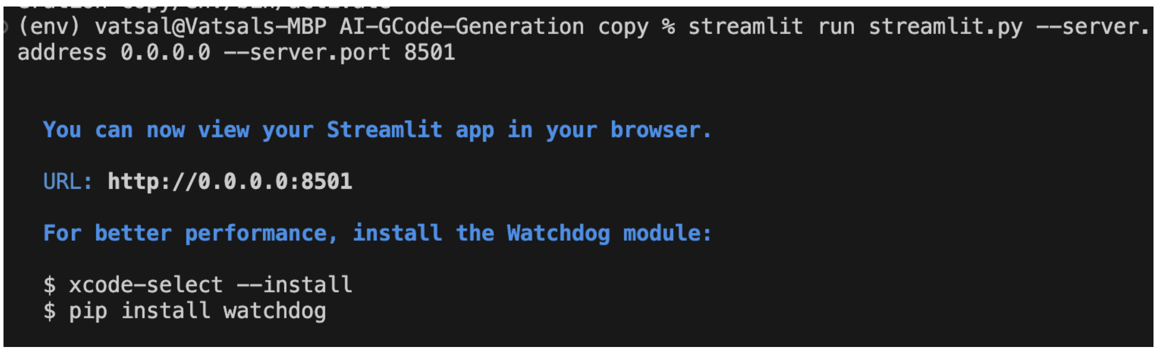
- **unicorn:** A lightweight ASGI server for running FastAPI or similar Python apps.
- **server_api:app:** Load the file `server_api.py` and Inside that file, find the object named app (usually a FastAPI instance).
- **–host 0.0.0.0:** Binds the server to all available network interfaces (not just localhost). This allows access from other machines on the same network (e.g., from another PC or Raspberry Pi).
- **–port 8080:** The backend server listens for HTTP requests on port 8080.



```
vatsal@Vatsals-MBP AI-GCode-Generation copy % source env/bin/activate
(env) vatsal@Vatsals-MBP AI-GCode-Generation copy % uvicorn server_api:app --host 0.0.0.0 --port 8080

INFO:     Started server process [52580]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Figure 3.11: Backend Output



```
(env) vatsal@Vatsals-MBP AI-GCode-Generation copy % streamlit run streamlit.py --server.
address 0.0.0.0 --server.port 8501


  You can now view your Streamlit app in your browser.

  URL: http://0.0.0.0:8501

  For better performance, install the Watchdog module:

  $ xcode-select --install
  $ pip install watchdog
```

Figure 3.12: Frontend Output

This starts your backend API layer 3.11, exposing endpoints like /upload, /process_svg, /generate_gcode, etc.

Frontend Web UI

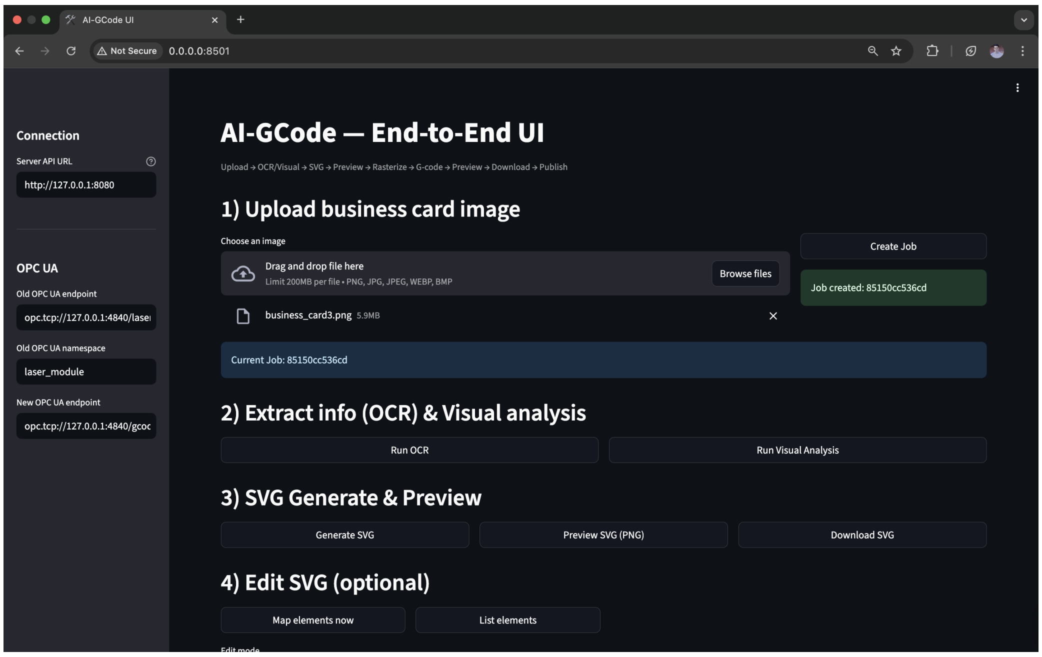
In another terminal write the following command and in the terminal it looks like 3.12:

Listing 3.8: Start Frontend Web UI

```
streamlit run server/streamlit.py --server.address 0.0.0.0
    --server.port 8501
```

- **streamlit run streamlit.py:** Launches the file `streamlit.py` as a Streamlit web app.
- **–server.address 0.0.0.0:** Makes the app accessible on all network interfaces (so not just localhost:8501, but also `http://<your_machine_ip>:8501)`, So that other can also access.
- **–server.port 8501:** Runs the app on port `8501` (default Streamlit port).

This starts your user-facing UI as shown in fig. 3.13, where users can upload cards, preview results, and trigger backend calls.



0.0.0.0:8501

Connection

Server API URL: http://127.0.0.1:8080

OPC UA

Old OPC UA endpoint: opc.tcp://127.0.0.1:4840/laser

Old OPC UA namespace: laser_module

New OPC UA endpoint: opc.tcp://127.0.0.1:4840/gcoc

business_card3.png 5.9MB

Job created: 85150cc536cd

Current Job: 85150cc536cd

Figure 3.13: Web UI

3.4.5 API Documentation with FastAPI

A key feature of the backend `server_api.py` is the automatic generation of interactive API documentation as shown in fig. 3.14 , provided by the FastAPI framework. Once the server is started with 3.7. The documentation can be accessed at:

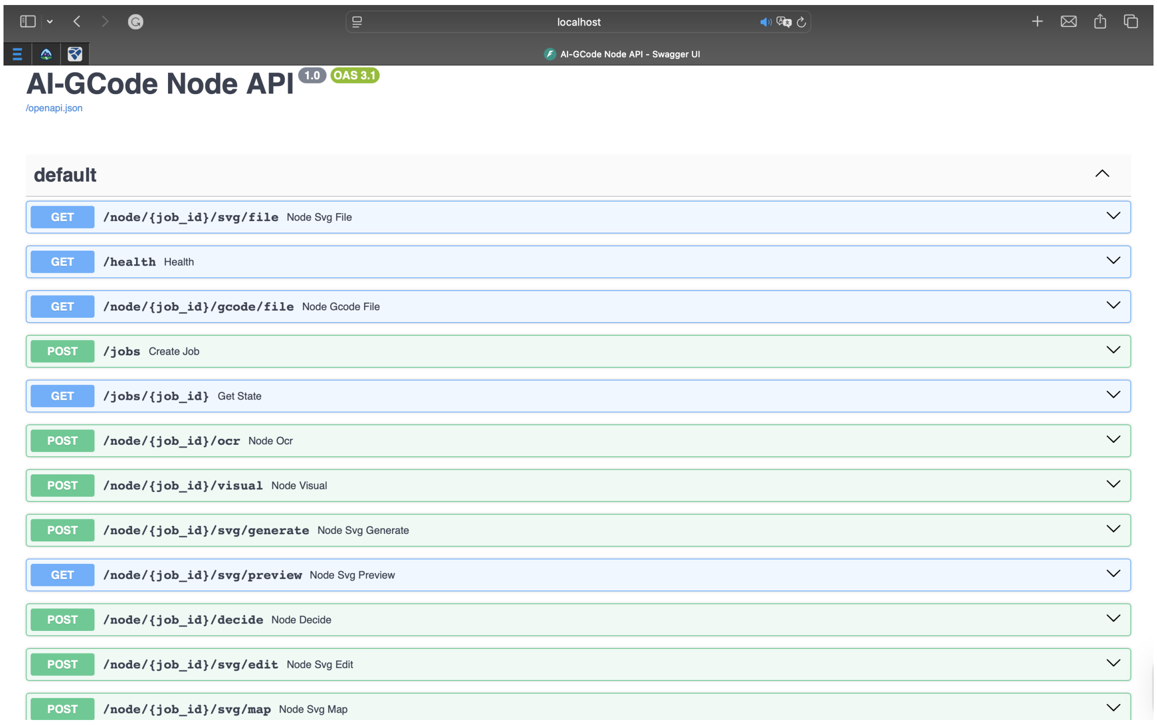
Listing 3.9: API Documentation

```
http://localhost:8080/docs
```

This launches the Swagger UI, an OpenAPI-based interactive interface that lists all available endpoints. Users and developers can:

- **Inspect endpoints:** View the methods implemented in the backend, such as uploading images, generating SVG files, or producing G-code.
- **Test interactively:** Submit requests directly in the browser (e.g., upload a card image and check the JSON output).
- **Validate responses:** Confirm that each endpoint returns the correct output before triggering the full workflow via the Web UI.

This documentation serves both as a developer tool for debugging and as a transparency layer, showing exactly how the Web UI communicates with the backend. Such self-documenting APIs are considered best practice in modern Industry 4.0 software development, as they enhance reproducibility, usability, and integration potential.



localhost

AI-GCode Node API - Swagger UI

AI-GCode Node API 1.0 OAS 3.1

/openapi.json

default

GET /node/{job_id}/svg/file Node Svg File

GET /health Health

GET /node/{job_id}/gcode/file Node Gcode File

POST /jobs Create Job

GET /jobs/{job_id} Get State

POST /node/{job_id}/ocr Node Ocr

POST /node/{job_id}/visual Node Visual

POST /node/{job_id}/svg/generate Node Svg Generate

GET /node/{job_id}/svg/preview Node Svg Preview

POST /node/{job_id}/decide Node Decide

POST /node/{job_id}/svg/edit Node Svg Edit

POST /node/{job_id}/svg/map Node Svg Map

Figure 3.14: API Documentation

3.5 OPC UA

The OPC UA implementation is embedded in the backend files `server_api.py, server_hmi.py, and client_hmi.py`. The structure follows a typical server–client architecture:

- `server_api.py`: Hosts variables and methods representing engraving jobs (e.g., job ID, file path, status).
- `server_hmi.py`: Provides a structured way to interact with OPC UA nodes, enabling monitoring and updating of job status.
- `client_hmi.py`: Connects to the OPC UA server to retrieve or update information, such as reading the generated G-code file path or confirming job completion.

3.5.1 Workflow with OPC UA

- After generating the final G-code file, the user specifies the target OPC UA endpoint directly in the Web UI sidebar under the OPC UA Section as shown in the fig 3.13.
- In the Digital Factory context, this endpoint corresponds to the HMI server connected to the Laser Engraver Module. The user provides the server's IP address and the correct namespace URI.

- Once configured, the backend publishes the file path and relevant job metadata (e.g., job ID, timestamp) to the designated OPC UA server.
- The OPC UA client then retrieves this information from the server.
- Finally, the client forwards the engraving job to the Laser Engraver Module in the Digital Factory, where it can be executed as part of the modular production workflow.

3.6 Docker Integration

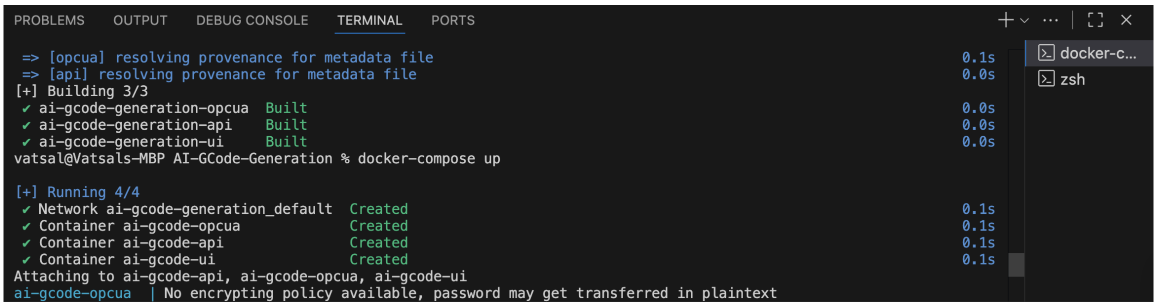
3.6.1 Purpose of Dockerisation

Docker was used to containerise the system in order to simplify deployment, ensure reproducibility, and allow others to run the full AI-GCode-Generation pipeline locally on their own machines without manual setup. By encapsulating all dependencies—such as Python libraries, LangGraph, AI models, and OPC UA modules—within Docker images, the system achieves portability across different operating systems and environments.

3.6.2 Implementation

The project includes a `Dockerfile` and `docker-compose.yml` configuration (see repository root). These files define how the environment is built and how different services (backend API, Streamlit UI, and OPC UA integration) are orchestrated together.

- `Dockerfile:` Specifies the base image (Python 3.12), installs required packages from requirements.txt, copies project files, and sets environment variables.
- `docker-compose.yml:` Configures multi-service deployment, including
  - **Backend API container (FastAPI with Uvicorn)**
  - **Streamlit UI container**
  - **OPC UA service container** (server or client, depending on configuration)
- `.env file:` stores API keys (Fireworks, OpenRouter, LangGraph) and is mounted inside the container for secure configuration. For ensure proper execution:
  - The **.env** file must be in the same directory as **docker-compose.yml**.
  - The **.env** file should contain all required API keys and must not have any file extensions (e.g., no .txt).



```
 => [opcua] resolving provenance for metadata file                                                            0.1s
 => [api] resolving provenance for metadata file                                                              0.0s
[+] Building 3/3
 ✔ ai-gcode-generation-opcua  Built                                                                          0.0s
 ✔ ai-gcode-generation-api    Built                                                                          0.0s
 ✔ ai-gcode-generation-ui     Built                                                                          0.0s
vatsal@Vatsals-MBP AI-GCode-Generation % docker-compose up

[+] Running 4/4
 ✔ Network ai-gcode-generation_default  Created                                                              0.1s
 ✔ Container ai-gcode-opcua             Created                                                              0.1s
 ✔ Container ai-gcode-api               Created                                                              0.1s
 ✔ Container ai-gcode-ui                Created                                                              0.1s
Attaching to ai-gcode-api, ai-gcode-opcua, ai-gcode-ui
ai-gcode-opcua  | No encrypting policy available, password may get transferred in plaintext
```

Figure 3.15: Docker Run

3.6.3 Workflow with Docker

Clone the repository:

Listing 3.10: Clone Repository

```
git clone https://github.com/mahajan-vatsal/AI-GCode-
    Generation.git
cd AI-GCode-Generation
```

Build the Docker images:

Listing 3.11: Build Image

```
docker-compose build
```

Launch the containers:

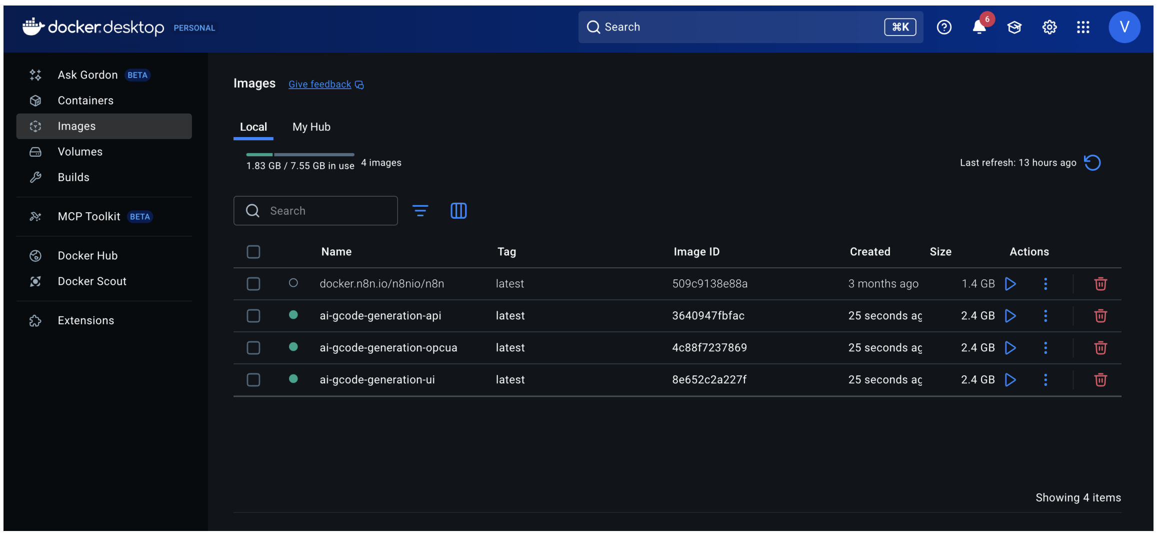
Listing 3.12: Launch Container

```
docker-compose up
```

Access the system locally:

- **Web UI**→ `http://localhost:8501`
- **Backend API** → `http://localhost:8080`

This approach enables collaborators to launch the system with a single command, without manually configuring Python environments or dependencies.



Name	Tag	Image ID	Created	Size
docker.n8n.io/n8nio/n8n	latest	509c9138e88a	3 months ago	1.4 GB
ai-gcode-generation-api	latest	3640947fbfac	25 seconds ag	2.4 GB
ai-gcode-generation-opcua	latest	4c88f7237869	25 seconds ag	2.4 GB
ai-gcode-generation-ui	latest	8e652c2a227f	25 seconds ag	2.4 GB

Figure 3.16: Docker Desktop

4 Validation of the Requirements and Limitations

4.1 Validation

This section validates how each project requirement was fulfilled through the implementation and testing processes.

4.1.1 Information Extraction

- The OCR Agent successfully extracted structured text (name, role, email, phone) using Qwen2.5-VL via Fireworks API.
- The Visual Analysis Agent identified layout elements such as logos and QR codes with bounding boxes mapped to millimetre coordinates (85×54 mm).

4.1.2 Layout Preservation

- The SVG Agent embedded extracted content into a vector file, preserving relative positions and scales. The Y-axis inversion (from bottom-left to SVG's top-left) was handled consistently, ensuring fidelity to physical card proportions.
- Validation involved overlaying bounding boxes on the generated SVG and comparing them with original card layouts, demonstrating positional accuracy consistent.

4.1.3 SVG Generation and Editing

- The system produced initial SVG files automatically, which could then be refined using either command-based editing or natural language editing (via the LLM-SVG Agent). Human-in-the-loop testing confirmed that edits such as "Replace 'Yaman Alsaady' with 'Vatsal Mahajan'." or "add_text tagline at x=3 y=6 text='Created by Vatsal' size=3.2" were correctly parsed and applied.

4.1.4 SVG Generation and Editing

- The system produced initial SVG files automatically, which could then be refined using either command-based editing or natural language editing (via the LLM-SVG Agent).
- Human-in-the-loop testing confirmed that edits such as "Replace 'Yaman Alsaady' with 'Vatsal Mahajan'." or "add_text tagline at x=3 y=6 text='Created by Vatsal' size=3.2" were correctly parsed and applied.

4.1.5 Rasterisation and Binarisation

- The Rasterisation Module (CairoSVG + OpenCV) converted SVGs into high-contrast bitmaps, ensuring readability for raster scanline engraving.
- Empirical tests showed that thresholding removed background noise and preserved fine details such as text edges and QR code readability.

4.1.6 G-code Generation and Preview

- The G-code Agent implemented a zig-zag scanline algorithm, minimizing non-productive travel while maintaining engraving precision.
- The G-code Preview Agent displayed the toolpaths, enabling error detection before physical execution.
- Comparative checks between preview and engraved outputs confirmed high fidelity, in line with G-code raster engraving principles.

4.2 Limitation

Although the system fulfils its primary objective of generating laser-ready G-code from business card images, several limitations remain:

- When logos and text are embedded within the same region, the system prioritises the text, often leaving the logo unprocessed. This limits the accuracy of reproducing complex card layouts.
- The vision model is not explicitly trained for arbitrary logo recognition. Logos are only reproduced if they already exist in the system's database, restricting generalisation to unseen designs.
- In cases where logos are retrieved from the database, the system writes them into the SVG with reduced size, which can distort the intended visual balance of the design.

- The model successfully detects the position and size of QR codes, it does not render them into the SVG due to security restrictions.
- Both the large language model (LLM) and vision-language model (VLM) currently run on free credits. Once these are exhausted, continued operation requires subscription costs, which may affect scalability.
- The system relies on openly available, free versions of LLMs and VLMs, which show limited accuracy compared to commercial or fine-tuned models, particularly for detailed layout extraction.
- The editing interface interprets only a narrow set of commands via regex matching (e.g., move, delete, replace, add_text). This restricts the naturalness of user interactions and may limit usability in broader contexts.

5 Conclusion and Future Scope

This project presented a complete pipeline to transform a photographed business card into laser-ready G-code within the context of the Digital Factory. By integrating computer vision, OCR, vector graphics, rasterisation, G-code generation, and agent-based orchestration with LangGraph, the system demonstrated how a traditionally manual process can be automated and streamlined. The implementation showed that:

- Information extraction and layout preservation were achieved using a vision-language model (Qwen2.5-VL) and OCR.
- SVG generation and LLM-assisted editing enabled both automation and human-in-the-loop refinement, balancing efficiency with control.
- Rasterisation and binarisation produced engraving-ready bitmaps, ensuring clarity and contrast.
- Scanline G-code generation with preview translated designs into precise toolpaths, with user validation reducing risk of errors.
- The LangGraph orchestration provided modularity, robustness, and scalability, allowing individual agents to work independently while contributing to a coherent workflow.

Overall, the project validated its requirements: the pipeline proved capable of automatically generating engraving instructions, with opportunities for user-driven refinements, thereby enhancing the functionality of the Laser Engraver Module in an Industry 4.0 setting.

6 Future Scope

While the project achieved a functional and well-integrated system, there are possibilities for further developments as mentioned below:

- Extend the workflow to include direct image capture from a Raspberry Pi camera, allowing business cards to be scanned and processed automatically. The captured image could be sent directly into the system's input folder, triggering the pipeline and automating G-code generation with minimal human intervention.
- Develop a dynamic GUI for displaying and editing the extracted data, allowing users to directly interact with text, logos, and layout elements. Such an interface would improve usability and provide greater flexibility compared to the current command-based approach.
- Integrate a drag-and-drop mechanism into the editing process, enabling users to reposition or resize elements intuitively. This would complement the existing natural-language editing feature and broaden accessibility for users without technical expertise.
- Develop a dedicated logo vectorisation agent using deep learning models (e.g., convolutional autoencoders or diffusion-based tracers) to accurately reproduce logos not present in the database.