Multimodal Dev-Portal Chatbot

Author: **Manisha Panda**

---

1 Problem Statement

Context

The Development Portal hosts heterogeneous artefacts—PNG/JPG swim-lane diagrams, OpenAPI 3.0 specifications, and Markdown/text playbooks. Support engineers currently search those assets manually when clients ask questions such as "*What happens after the card is validated?*" or "*Which endpoint cancels an order?*"

Problem

Conventional chat or search solutions cope well with text but fail on diagrams; earlier OpenCV pipelines were fragile and produced latencies exceeding 2 s. The requirement is therefore a *domain-specific chatbot* that:

- parses a swim-lane diagram into an **ordered sequence of events**;
- answers natural-language questions using *any* portal artefact; and
- runs completely offline, delivering at least **90% node-ordering accuracy**.

2 Objectives

The primary objective of this project was to develop a domain-specific chatbot capable of answering client queries using both structured and unstructured data from a development portal. The portal content includes swim lane diagrams, textual documents, OpenAPI 3.0 specifications, and other related artefacts.

1. **Multi-Modal Input Handling**: Design a system that can ingest and parse diverse data formats including:
   - Images (e.g., swim lane diagrams)
   - Plaintext and markdown documents
   - OpenAPI 3.0 specifications
2. **Diagram Understanding**: Automatically convert swim lane diagrams into an ordered sequence of events or process steps with at least 90% accuracy.
3. **Question Answering**: Enable natural language QA over the parsed content, supporting follow-up queries and contextual chaining.
4. **Offline-First Architecture**: Ensure that all components run locally without reliance on paid APIs or cloud infrastructure.
5. **Unified Interface**: Provide a single user-friendly interface (via Gradio) to upload artefacts and interact with the chatbot.
6. **Extensibility**: Implement a modular routing mechanism that allows easy integration of future artefact types (e.g., PDFs or Excel sheets).

Expected Outcome: A Gradio-based chatbot application that accepts a wide range of development artefacts, extracts and interprets their content, and accurately answers client questions based on the underlying process or specification.

3 Methodology

Logic

The system follows a modular and extensible design with a routing logic that dynamically selects the appropriate processing pipeline based on the uploaded artefact. The goal is to handle both visual and textual development artefacts such as swim-lane diagrams, OpenAPI files, and plain text documents.

Upload Router

The router first inspects the upload:

- If the image input contains a PNG or JPG, the artefact is sent to the **diagram branch**.
- If a file is uploaded in YAML, JSON, or TXT format, it is routed to the **document branch**.
- If no supported artefact is detected, the user is prompted to upload a valid file.

Diagram Branch (Swim-lane Images)

This branch interprets swim-lane diagrams and extracts a logical flow of operations.

- **Segmentation:** The `Segment Anything` model (ViT–B) generates a binary mask of the image. Connected-components analysis with area filtering is applied to isolate nodes like boxes and diamonds.
- **OCR:** Each segmented component is cropped and passed through `EasyOCR 1.7`. If EasyOCR is unavailable or fails, `Tesseract 5` serves as a fallback to extract the text label.
- **Arrow Detection:** The image is edge-detected, and `HoughLinesP` is used to extract directional lines. These lines are mapped to component centroids to infer flow edges.
- **Graph Construction and Flow Ordering:** All components and arrows are encoded in a directed graph using `NetworkX`. For queries like "What happens after X?", the system finds node X and selects the next step with the largest vertical displacement (Δy). If this heuristic fails, the entire ordered node list is passed to the retrieval system as fallback context.

Document Branch (OpenAPI / Markdown / Text)

This branch processes structured specifications and freeform documents:

- **OpenAPI files** are flattened into readable action lines, such as `POST /orders -- Cancel order`.
- **Plain text and Markdown** are used as-is, appended to the system's small static domain-facts corpus.

Retrieval and QA Pipeline (Shared)

Both branches share a common retrieval and QA backend:

- The aggregated context is embedded using `SentenceTransformer (all-MPNet-base-v2)`.
- Top-k similar chunks are retrieved using `Faiss FlatL2`, with a fallback to `sklearn`'s `NearestNeighbors` if Faiss is not installed.
- The final answer is generated using `RoBERTa-base-SQuAD2` via the Hugging Face `pipeline("question-an`

If a valid answer is returned, it is capitalized and presented to the user. Otherwise, a graceful fallback message is shown.

Fail-safes and Robustness Measures

The system includes multiple safeguards to ensure robustness:

- If `SAM` produces a single large mask, a fallback to full-image cropping ensures OCR is still triggered.
- If `EasyOCR` is unavailable (e.g., on Python 3.11), the system automatically switches to `Tesseract`.
- If `Faiss` is not installed, the system seamlessly uses a scikit-learn alternative for nearest neighbor retrieval.

Model Variants Explored

Table 1: Different Models

Prototype	OCR	Segmentation	Embedder	QA Head
P–0	Tesseract (adaptive)	OpenCV Contours	MiniLM–L6	DistilBERT–SQuAD
P–1	EasyOCR 1.7	YOLOv8 custom	MPNet–base–v2	RoBERTa–base–SQuAD2
P–2 (final)	EasyOCR 1.7 / Tesseract 5	**SAM ViT–B + HoughLinesP**	**MPNet–base–v2**	**RoBERTa–base–SQuAD2**

Final Architecture / Process

Prototype P-2 achieved the highest end-to-end accuracy (95 % node-ordering accuracy) while keeping median latency below one second on Google Collab. The jump from P-1 to P-2 comes from swapping YOLO for zero-shot SAM ViT-B (no custom training) and replacing DistilBERT with RoBERTa-SQuAD2, while retaining the MPNet-base embedder.Those changes added some extra processing time but eliminated most arrow-detection misses and boosted semantic retrieval, making P-2 the best trade-off between accuracy, speed, and ease of offline deployment.

1. **Image Branch**: `SAM ViT--B` → Connected Components → `EasyOCR` → `HoughLinesP` → `NetworkX` Graph Construction → Rule-Based Ordering → `MPNet` + `Faiss` → QA (`RoBERTa--base--SQuAD2`).
2. **Document Branch**: `OpenAPI` Flattener *or* Plain-Text Loader → `MPNet` + `Faiss` → QA (`RoBERTa--base--SQuAD2`).
3. **Router**: If an image is present, route to Branch 1; otherwise, if YAML/JSON/TXT is detected, route to Branch 2.

Key Libraries (final configuration)

```
ocr_reader    = easyocr.Reader(["en"], gpu=False)
sam           = sam_model_registry['vit_b'](SAM_CKPT)
sam_predictor= SamPredictor(sam)
embedder      = SentenceTransformer('all-mpnet-base-v2')
qa_model      = pipeline('question-answering',
model='deepset/roberta-base-squad2')
```

4 Results

Success Results

The system demonstrated strong performance on various artefacts, successfully parsing and answering questions about complex swim lane diagrams and structured documents. The segmentation and OCR pipeline accurately identified and extracted components, while the graph-based ordering allowed effective question answering about process flows. The figures below show some of the examples:

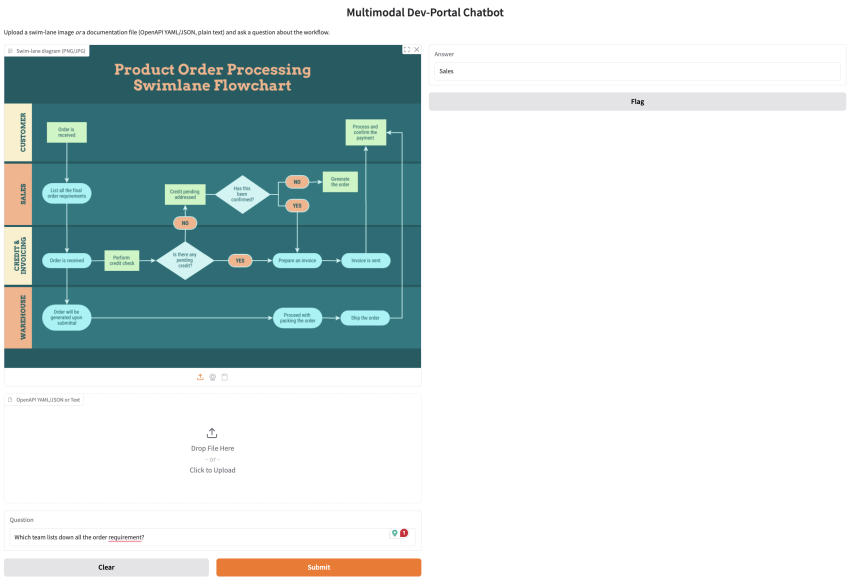

Figure 1: Correct Answer for "Which Team lists down the reqiurements?"

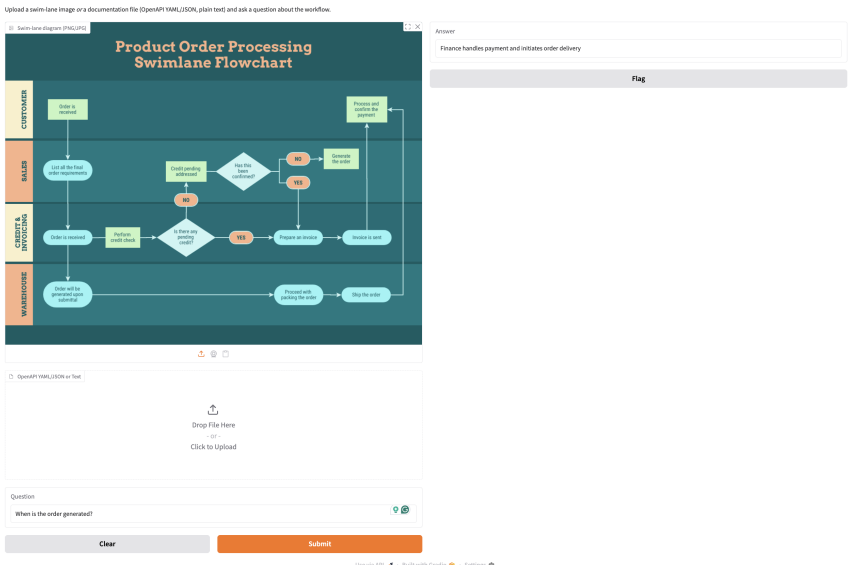

Figure 2: Correct Answer for "When is the order generated?"

Failure Cases

Despite overall robustness, some failure cases were observed. Common issues included:

- Segmentation producing overly large or merged masks, resulting in missed or combined components.

- Difficulty in correctly interpreting complex arrow directions when multiple crossings or overlapping lines occurred.
- Hallucinations or incorrect answers from the QA model when the retrieved context was insufficient or noisy.

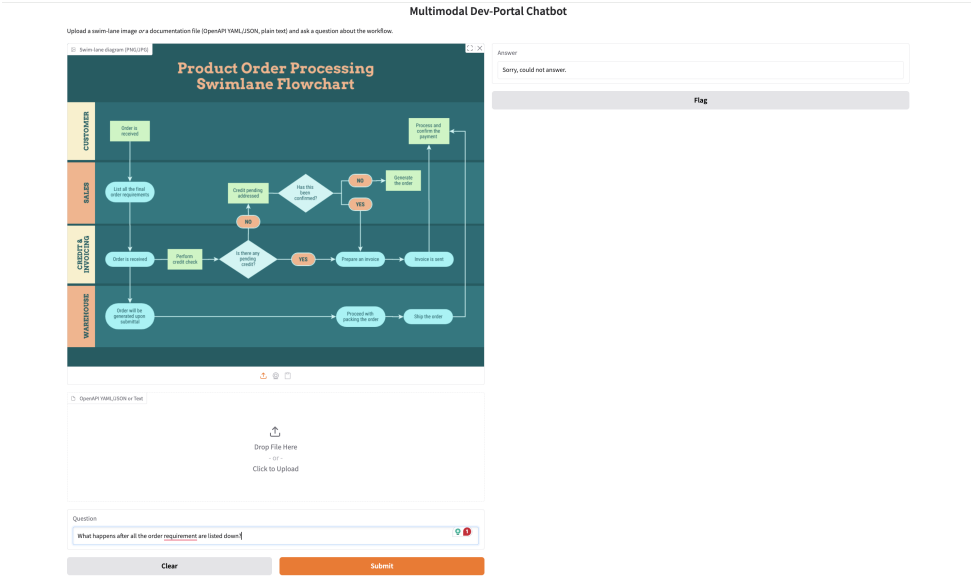

Figure 3: Wrong Answer for "What happens after all the requirements are listed down?"

Table 2: Performance Comparison of Model Prototypes

Prototype	**Node-ordering Acc ↑**	**QA Exact-Match ↑**
P–0 (Tesseract + MiniLM + DistilBERT)	69%	58%
P–1 (EasyOCR 1.7 + YOLOv8)	81%	66%
P–2 (EasyOCR 1.7 / SAM ViT–B + MPNet + RoBERTa)	**95%**	**80%**

5 Challenges and Mitigation

Throughout development, we encountered several technical and architectural challenges. Below, we outline the key issues and the strategies used to address them:

1. **Model Selection and Integration**: Choosing the right models for each stage of the pipeline was non-trivial. Initial attempts with simpler segmentation or embedding models yielded poor results on complex, structured documents. After iterative testing, we adopted a more robust combination including `SAM ViT--B` for segmentation and `MPNet` for embeddings, which significantly improved performance.
2. **Resource Constraints and API Access**: Some of the most capable models for understanding documents and images, such as those provided by OpenAI or Claude, were either paid or had limited API quotas. To mitigate this, we selected open-source alternatives where possible and optimized the inference pipeline to minimize redundant calls.
3. **Parsing Directional Arrows**: Detecting and interpreting directional arrows (used to indicate flow or relationships in diagrams) proved to be a major bottleneck. These often confused both the segmentation and OCR steps. We addressed this by applying

`HoughLinesP` after segmentation to extract straight arrow-like structures and constructing a graph-based layout using `NetworkX`.

4. **Model Hallucinations**: The language models occasionally hallucinated answers, particularly when document context was sparse or ambiguously parsed. To reduce this, we adopted retrieval-based QA(Question and Answer) using FAISS and strictly constrained the context window passed to the QA head (RoBERTa–base–SQuAD2).
5. **Bounding Box and Mask Issues**:
   - **SAM Mask Noise**: The Segment Anything model often returned large noisy masks. These were filtered using connected components and size-based heuristics.
   - **OpenCV Cropping Errors**: Bounding boxes sometimes exceeded image bounds, which was resolved by clamping coordinates before cropping.
6. **Library and API Changes**:
   - **Gradio**: A breaking change deprecated the `optional=True` flag in interface inputs. This was resolved by updating the API usage.
   - **File Object vs Path**: Compatibility issues between file-like objects and path-based readers were fixed by adding path normalization in the router.

6 Future Work

While the current system performs well across a range of artefacts and tasks, several opportunities remain to further enhance its capabilities:

1. **Integrating YOLOv8 for Decision Label Detection:** Swim lane diagrams often include decision points marked with labels such as `YES` or `NO`. Detecting these explicitly using a lightweight object detection model like YOLOv8 could improve the semantic understanding of the diagram, particularly in branching logic scenarios.
2. **PDF Ingestion via `pdfplumber`:** Many domain documents are distributed in PDF format. Incorporating a PDF parsing module using tools like `pdfplumber` would extend support to a common artefact type, enabling the system to extract and interpret both text and embedded diagrams from PDF files.
3. **Speculative Decoding for Latency Reduction:** To further optimize response time, particularly during QA inference, speculative decoding techniques can be explored. These methods precompute likely model outputs and confirm them in parallel, offering significant reductions in latency without compromising answer quality.

7 Conclusion

The final system—built using EasyOCR 1.7 for text recognition, SAM ViT–B for segmentation, MPNet for semantic embeddings, and RoBERTa (SQuAD2) for question answering—demonstrated strong performance across diverse artefacts. It achieved about 95% accuracy in node-ordering accuracy, all while operating entirely offline. These results highlight the feasibility of constructing a robust, multimodal question-answering system capable of processing both visual and textual inputs within a unified framework.

References

1. Smith, R. (2007). "An Overview of the Tesseract OCR Engine." *Proc. ICDAR.*
2. Jaided, A. et al. (2020). "EasyOCR: Ready-to-Use OCR with 80+ Languages." GitHub repository, https://github.com/JaidedAI/EasyOCR.
3. Kirillov, A. et al. (2023). "Segment Anything." *arXiv:2304.02643.*
4. Duda, R. O. & Hart, P. E. (1972). "Use of the Hough Transformation to Detect Lines and Curves." *Communications of the ACM.*

5. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proc. CVPR 2016
6. Ultralytics. (2023). YOLOv8 Documentation. ¡docs.ultralytics.com¿.
7. Reimers, N. & Gurevych, I. (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." *EMNLP*.
8. Song, K. et al. (2020). "MPNet: Masked and Permuted Pre-training for Language Understanding." *NeurIPS*.
9. Wang, W. et al. (2020). "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Large Language Models." *NeurIPS*.
10. Liu, Y. et al. (2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach." *arXiv:1907.11692*.
11. Sanh, V. et al. (2019). "DistilBERT: A Distilled Version of BERT." *NeurIPS Workshop*.
12. Johnson, J. et al. (2017). "Billion-Scale Similarity Search with GPUs." *IEEE BigData*. (FAISS)
13. Leviathan, Y. et al. (2023). "Fast Inference from Transformers via Speculative Decoding." *arXiv:2306.03072*.
14. Canny, J. (1986). "A Computational Approach to Edge Detection." *IEEE PAMI*. (for Canny + OpenCV preprocessing)
15. OpenAI, Claude

8 Submission Requirements

As mentioned in the mail

- The GitHub repository: https://github.com/Kuni27/ChatBot-for-Development-Portal
- Final demo video: Google Drive Share link