Utility Copilot: Integrating Object Detection, Voice-to-Text, and Document Retrieval for Electric Utility Engineering

David Schiff
University of Central Florida
Orlando, Florida, USA

ABSTRACT

This paper presents Utility Copilot, an AI assistant for electric utility engineering that combines YOLOv7-based object detection, voice interaction, and document retrieval. The system achieves 89% precision in equipment identification across 11 classes and demonstrates a 49.3% improvement in documentation query accuracy over baseline LLMs. Through a web-based interface, utility workers can identify equipment, access technical documentation, and interact hands-free, streamlining field operations while maintaining safety standards. Evaluation results show significant improvements in both accuracy and efficiency, with statistical validation ($p < 0.05$) confirming the system's effectiveness for practical utility engineering tasks.

1 INTRODUCTION

While Large Language Models (LLMs) have revolutionized software development through code assistance and debugging tools, there remains a significant gap in their application to engineering disciplines, particularly in utility design. The electric utility industry, with its complex infrastructure and niche materials and documentation, presents unique challenges that traditional LLM applications have yet to fully address. This project explores the development of an LLM-powered assistant for utility engineers, combining visual analysis, natural language processing, and domain-specific knowledge.

Electric utility engineers face several significant challenges in their daily work that current technologies don't adequately address:

- Need for rapid equipment identification and documentation in the field
- Manual consultation of multiple resources causing workflow inefficiencies
- Time-consuming documentation processes and delayed access to critical information

This project implements an assistant that aims to streamline these workflows through three core functionalities:

(1) **Computer Vision:** Transformer-based object detection models for equipment identification
(2) **Voice Interaction:** Speech recognition capabilities for hands-free operation
(3) **Documentation Retrieval:** Context-aware access to technical documentation

The system employs an adaptable design that focuses on reliability and maintainability, allowing for integration of new features and updates to individual components without disrupting the overall system. While developed as a solo project, the implementation prioritizes practical utility and ease of use in field conditions. By combining visual analysis, natural language processing, and document retrieval in a single interface, this tool aims to assist utility engineers with their daily tasks, potentially improving efficiency while maintaining necessary safety standards.

2 RELATED WORK

AI and large language models (LLMs) have advanced utility engineering infrastructure management and maintenance. Both academic research and industry developments showcase solutions to challenges.

2.1 AI in Object Detection for Utility Engineering

AI-powered object detection has become vital for utility engineering inspection. Industry leader Viso.ai demonstrates computer vision for pole inspections, using YOLO and Faster R-CNN to identify equipment [7]. In vegetation management, Barron's reports detail systems that analyze aerial imagery to identify hazards near power lines [2].

Academic research validates these approaches. The "Insu-YOLO" algorithm [3] enhances YOLOv8 for insulator defect detection, achieving 95.9% mean average precision at 87 frames per second. Limberg et al. [5] evaluate YOLO-World and GPT-4V on aerial datasets, demonstrating effective person detection and scene description capabilities.

2.2 Applications of Large Language Models in Utility Engineering

Industry applications show LLMs transforming operational data management. Utility Analytics demonstrates equipment failure prediction and maintenance optimization [1], while Fowler shows LLM-based automation for technical documentation [4].

In academic work, Pu et al. [6] combine Set-of-Mark prompting with multimodal LLMs to automate inspection reports. Their method processes site images with GPT-4 to generate detailed reports while reducing manual effort.

2.3 Integrating Object Detection and LLMs

The synthesis of object detection and LLMs creates new opportunities in utility engineering. While industry shows practical applications, academic research provides validation through studies like Insu-YOLO [3] and Limberg et al.'s work [5]. Pu et al. [6] demonstrate how combining visual and textual processing can streamline inspection workflows.

These advances in research and implementation address challenges in inspection, maintenance, and information retrieval, improving utility operations' safety and efficiency.

3 SYSTEM ARCHITECTURE AND METHODS

The Utility Copilot application is designed as a multifunctional AI assistant to support electric utility workers by integrating object detection, voice-to-text transcription, and document retrieval. It is implemented as a web-based interface powered by Flask, OpenAI's GPT models, and LangChain. The complete implementation and setup instructions are available on GitHub[1]

3.1 System Overview

The architecture follows a modular design pattern that integrates multiple processing pathways into a cohesive system, as illustrated in Figure 1.

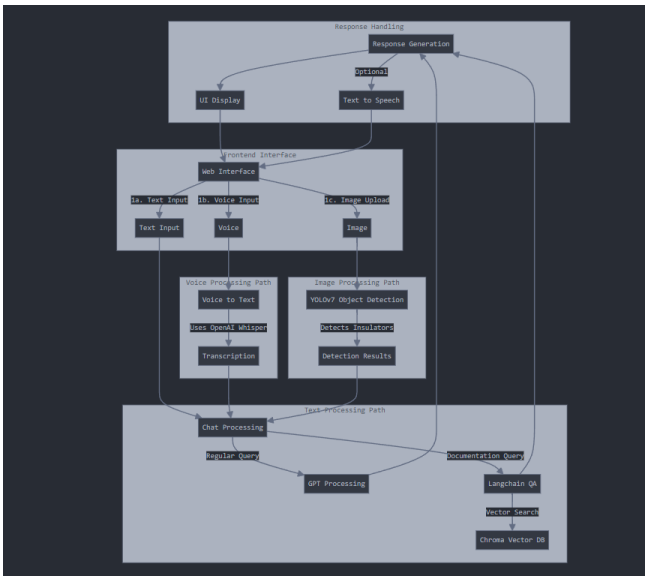

Figure 1: System Architecture

The system is organized into three main layers: the frontend interface, processing pathways, and response handling. Each layer serves a specific purpose while maintaining clear communication channels with other components.

3.1.1 Frontend Interface. The web interface provides three primary input methods for user interaction:

- Text Input: Traditional text-based chat interface
- Voice Input: Speech-to-text capability for hands-free operation
- Image Upload: Visual input for object detection

3.1.2 Processing Pathways. The system implements three distinct processing paths that handle different types of input:

Voice Processing Path: This pathway handles audio input through a sequence of operations:

- Voice-to-Text conversion using OpenAI's Whisper API
- Transcription processing and validation
- Integration with the chat processing system

Image Processing Path: For visual input, the system employs:

- YOLOv7 object detection for utility equipment identification
- Detection result processing and context integration

Text Processing Path: The system differentiates between two types of text queries:

- Regular queries handled by GPT processing
- Documentation queries routed through LangChain QA with vector search

3.1.3 Response Handling. The response generation system coordinates multiple output types:

1. Response Generation: Creates contextually appropriate responses based on input type and query classification
2. Output Methods:

- UI Display: Visual presentation of responses and detection results
- Text-to-Speech: Optional audio output for responses

This architectural design ensures that all components work together while maintaining separation. The system can handle multiple input types simultaneously and route them through appropriate processing paths while maintaining context and providing consistent response formatting.

3.2 Major Functional Components

Building on this architecture, the system implements three major functional components:

1. **Object Detection:** A YOLOv7-based model identifies utility-related objects from uploaded images and incorporates the results into conversation context.
2. **Voice-to-Text Transcription:** The voice interface enables users to input text via speech, processed using OpenAI's Whisper API.
3. **Document Retrieval:** LangChain retrieves and summarizes relevant information from company documents using a Chroma vector database and OpenAI embeddings.

Each of these components is integrated into the larger system architecture while maintaining its own specific processing pipeline and optimization parameters.

3.3 Object Detection Module

The object detection module serves as the visual understanding component of the Utility Copilot application. Think of it as the system's eyes - it can see and identify utility equipment in photographs, helping field workers quickly identify equipment, verify installations, and get information about components they encounter in their work. The module brings together several components that work together to provide this functionality while making it easy for the user to inference in a single interface.

3.3.1 System Workflow Overview. Before diving into the specific components, it's important to understand how the system orchestrates the detection process from start to finish. Figure 2 illustrates the complete workflow and interaction between components.

[1]The implementation code can be found at https://github.com/BilboBackends/Utility-Copilot

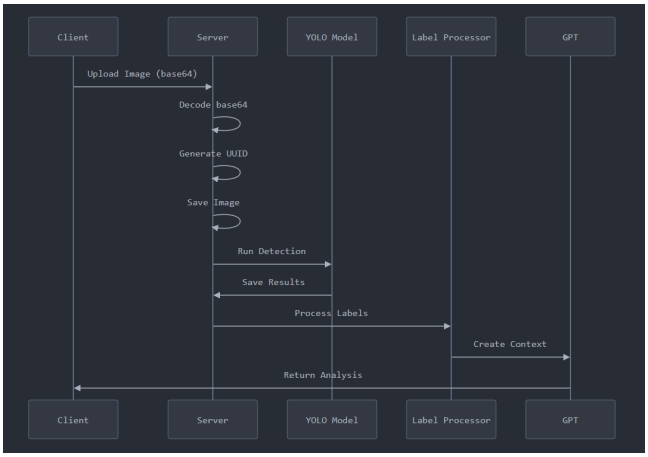

Figure 2: Object Detection Sequence Workflow

The sequence flows through five main components: Client, Server, YOLO Model, Label Processor, and GPT. Each plays a specific role in transforming a raw image into LLM context:

1. The process begins when the Client uploads an image, converting it to base64 format for efficient transmission. This encoding ensures reliable image data transfer across the network.

2. The Server then performs three preliminary steps:

- Decodes the base64 image back to its original format
- Generates a UUID to uniquely identify this detection session
- Saves the image to the appropriate directory for processing

3. Next, the Server triggers the YOLO Model for detection, initiating the core computer vision analysis. The model processes the image and saves its results, including bounding boxes and class predictions.

4. The Label Processor then takes over, interpreting the raw detection data. This component translates model outputs into meaningful labels and organizes the detection results.

5. Finally, the GPT component creates a conversational context from the processed labels, enabling natural interaction about the detected objects.

This coordinated sequence ensures reliable processing while maintaining clear separation between components. Each step builds upon the previous ones, creating a robust pipeline from raw image to analysis.

3.3.2 Overview and User Experience. From a designers perspective, using the system is straightforward. When they encounter equipment in the field or in the office, they simply take a photo and upload it to the application. The system then begins its analysis, looking for familiar components and integrating what it finds into the conversation. This creates a natural workflow where workers can discuss what they're looking at and receive informed responses based on the system's visual analysis.

The interface provides two simple ways to submit images:

- Drag and drop an image directly onto the chat interface
- Click the upload zone to select an image from their device

3.3.3 System Architecture. Behind this simple interface lies a sophisticated architecture built around three core components:

Core System Files:

- app.py - Main Flask application and detection coordination
- script.js - Front-end interface and user interaction
- config.json - System configuration and parameters

These components work together to handle the complete detection process. The Flask application acts as the central hub, coordinating all the moving parts. When an image arrives, it's processed through a structured directory system:

Directory Structure:

- /inference/images/uploaded/ - Initial image storage
- /runs/detect/from_uploaded/ - Detection processing and results
- /static/images/ - Web-accessible processed images

3.3.4 Detection Process Flow. When someone uploads an image, the system recognizes its a photo through its file extension and analyzes it through a series of steps. First, the image is received and encoded for safe transmission, with a unique identifier assigned to prevent any confusion with other uploads. The system then channels this through the YOLOv7 model, which has been specially trained to recognize utility equipment.

Object Categories:

- Primary Infrastructure
  - Poles, Transformers, Capacitors
- Safety and Control Equipment
  - Fuses, Switches, Reclosers
- Support Components
  - Down Guys, Tags, Risers
- Protective Equipment
  - Arresters, Weatherheads, Trip Savers

The detection process is configured with specific parameters to ensure reliability - it operates at a 640x640 resolution and uses a confidence threshold of 0.5. This means the system only reports detections it's reasonably confident about, helping prevent false positives while maintaining high accuracy.

3.3.5 Integration and Output. After processing an image, the system provides multiple forms of feedback that integrates into the conversation. Users see an annotated image showing boxes around detected objects with clear labels, alongside a summary of what was found. This visual and textual information becomes part of the conversation context, allowing for natural follow-up questions about specific equipment.

3.4 Voice-to-Text Module

The voice-to-text module adds hands-free interaction capabilities to the Utility Copilot, allowing field workers to speak their queries and commands naturally. Using OpenAI's Whisper API for transcription, the system converts spoken words into text that integrates with the chat interface. This capability is particularly valuable for utility workers who may need to communicate with the system while their hands are occupied with equipment, forms, clipboards, etc.

3.4.1 System Workflow Overview. The voice-to-text functionality operates through a sequence of operations coordinated across multiple system components. Figure 3 illustrates this flow.

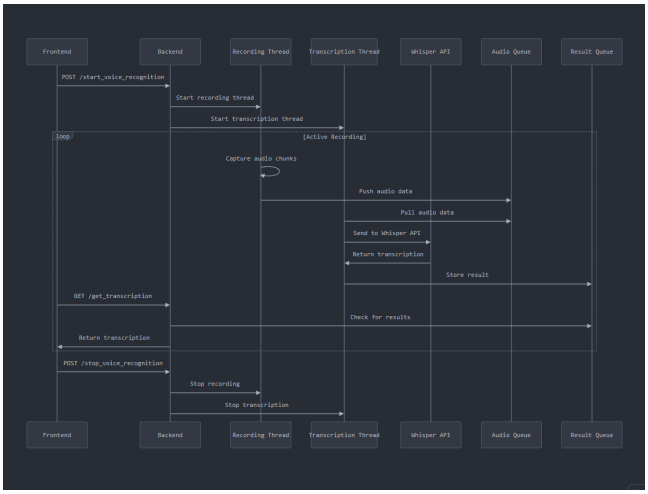

Figure 3: Voice-to-Text Processing Sequence

The system orchestrates communication between seven components that work together to enable voice recognition. The process begins when a user initiates voice recognition through the frontend interface. This triggers the backend to coordinate two parallel processes: a recording thread for capturing audio and a transcription thread for processing that audio into text.

The recording thread continuously captures audio in manageable chunks, handling the raw audio stream while maintaining active recording status. Meanwhile, the transcription thread manages the processing of this audio data, communicating with the Whisper API and handling the results. These processes are coordinated through two queues: an audio queue that buffers captured audio data and a result queue that stores processed transcriptions.

This sequence continues until the user chooses to stop voice recognition, at which point the system shuts down all active processes, so no data is lost and all resources are properly released.

3.4.2 Technical Implementation. The voice processing system is architected around several components that work to deliver reliable voice recognition. The system is built on four main components:

Core Components:

- VoicetoText.py - Primary voice processing engine
- voice.py - Audio file operations and management
- config.py - State management and configuration
- app.py - Flask routes for voice control

The system's audio capture is tuned for optimal speech recognition. It operates at a 16000 Hz sample rate, providing a balance between audio quality and processing efficiency. The energy threshold is set at 500 to effectively filter out background noise, while a pause threshold of 0.8 seconds helps detect natural breaks in speech. Perhaps most importantly, the system employs dynamic energy adjustment, automatically adapting to ambient noise levels in different environments.

3.4.3 Thread Management and Processing. When a user activates voice recognition, the system spawns two threads that work in parallel. The recording thread takes on the responsibility of monitoring audio input continuously. It processes the incoming signal to reduce noise and captures the audio in chunks, while ensuring efficient queue management for the audio data.

Running alongside this, the transcription thread performs the task of converting speech to text. It retrieves audio data from the queue as it becomes available, handles all communication with the Whisper API, and processes the returned transcriptions. This parallel processing approach allows the system to maintain real-time performance while ensuring no audio data is lost.

3.4.4 Whisper API Integration. The integration with OpenAI's Whisper API provides the backbone of the speech recognition capabilities. The API handles different accents and speech patterns, maintaining high accuracy even in noisy environments. Its ability to understand technical vocabulary makes it particularly well-suited for utility work, where specialized terminology is common.

The audio processing pipeline ensures efficient handling of speech data. Raw audio is converted into the format required by the API, then processed in chunks to maintain responsive performance. The system includes robust error handling and retry logic.

3.4.5 State Management and Error Handling. Maintaining reliable operation requires careful state management across the system. Three critical state variables coordinate the voice recognition process:

State Variables:

- voice_recognition_active - Controls recording state
- recognitionStarted - Tracks UI button state
- recognitionIntervalId - Manages polling intervals

The system implements error handling at multiple levels. Before recording begins, it verifies microphone access and monitors audio quality to ensure viable input. During processing, it prevents queue overflows and handles API errors.

3.4.6 User Interface Integration. The voice recognition feature has been integrated into the chat interface to provide a coherent user experience. Users interact with a microphone button that indicates when voice recognition is active. As they speak, they see their words appear in real-time, with visual indicators showing recording and processing status.

The interface gives users full control over their voice input. They can review transcriptions before sending, make any necessary edits, or cancel and retry if needed.

3.4.7 Practical Benefits. In practice, this voice-to-text capability transforms how utility workers interact with the system. The hands-free operation is invaluable when workers are handling equipment, allowing them to maintain focus on their physical tasks while still accessing the information they need. Voice input is often faster than typing, and the system's natural language processing makes it accessible to all users, regardless of their typing proficiency.

3.5 Document Retrieval Module with LangChain

The document retrieval module acts as the system's memory and knowledge base. Using LangChain's capabilities, it enables the Utility Copilot to search through company documentation and provide accurate, sourced answers to user queries. This module transforms static PDF documents into an interactive knowledge resource, making it easy for workers to access the information they need without manually searching through long manuals.

3.5.1 Overview and Architecture. The system uses a technique called Retrieval-Augmented Generation (RAG) to provide accurate responses. Think of it as having an intelligent assistant that can instantly read through all your documentation, find the most relevant sections, and craft a helpful response based on exactly what it finds. Figure 4 illustrates this workflow.

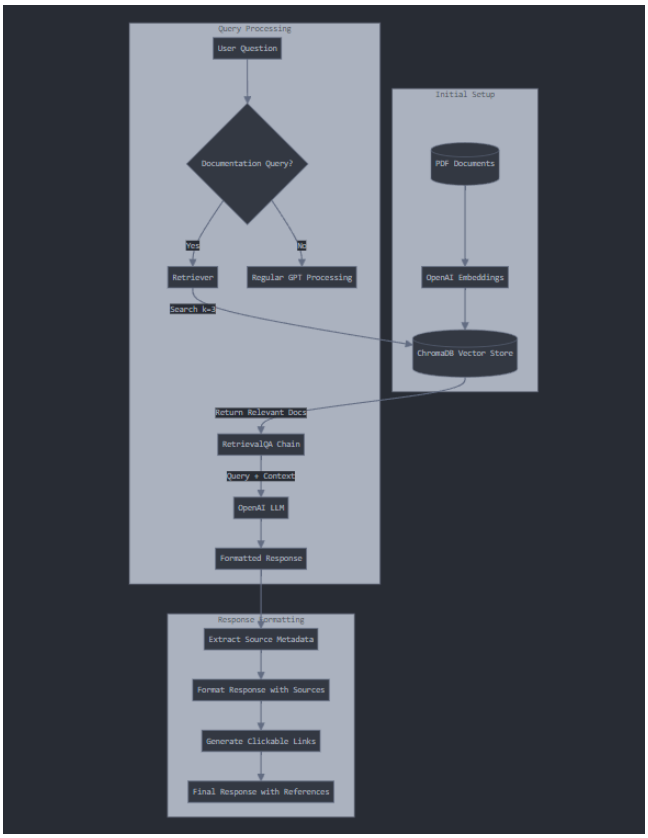

Figure 4: Document Query Processing and Response Generation Flow

As shown in the diagram, the process combines two main pathways: initial setup for document processing and the dynamic query handling flow. When a user asks a question, the system first determines if it's a documentation query. Based on this decision, it either routes the question through the retrieval system or handles it as a regular chat interaction. For documentation queries, the system leverages the ChromaDB vector store to find relevant content, processes it through a retrieval chain, and generates a response complete with source references and clickable links.

The module is built around several LangChain components that work together:

Core Components:

- OpenAI Embeddings - Converts text into semantic vectors
- Chroma Vector Store - Manages and searches document embeddings
- RetrievalQA Chain - Coordinates the question-answering process
- RecursiveCharacterTextSplitter - Breaks documents into manageable chunks

3.5.2 Document Processing Pipeline. The transition from static PDF to searchable knowledge happens in several stages. First, load the company's PDF documentation using LangChain's PyPDFLoader. This loader was chosen because it is good at maintaining document structure and metadata, which helps us keep track of where information comes from, which we will use later when generating the sources.

Once a document is loaded, we need to break it down into smaller, manageable pieces. This is where the RecursiveCharacterTextSplitter comes in, using these parameters:

Splitting:

- Chunk Size: 1000 characters
- Chunk Overlap: 200 characters

3.5.3 Semantic Search. The main part of the retrieval system lies in how we transform text into searchable vectors. Each chunk of text is processed through OpenAI's embedding model. These vectors are then stored in a Chroma database, creating a searchable index of all the documentation.

When a user asks a question, we follow this retrieval process:

1. The user's question is converted into the same vector space as the stored documents 2. The system searches for the most similar vectors in the database 3. Relevant document chunks are retrieved and ranked by similarity 4. The context is assembled and passed to the language model

Search:

- Top k Results: 3 most relevant chunks
- Similarity Metric: Cosine similarity
- Response Format: Includes both answer and sources

3.5.4 Query Processing and Response Generation. The system is dynamic about how it handles queries. Implemented is a keyword-based detection system that identifies when a user is likely asking for documentation-related information. This helps the system decide whether to engage the document retrieval process or handle the query through standard conversation.

Documentation Keywords (partial example sections):

- Process terms: "how to," "steps," "procedure"
- Reference terms: "manual," "guide," "specifications , documentation"
- Inquiry terms: "what is," "explain," "describe"

When a documentation query is identified, the RetrievalQA chain gets called. It coordinates between the retriever and the language model, ensuring that responses are both accurate and natural.

3.5.5 Source Attribution and Verification. One of the most valuable features of the implementation is how it handles sources. Every response includes references to where the information was found, complete with page numbers and section titles. Users see these as clickable links that open the relevant sections of the PDF documentation:

For example, a response might look like: "The standard clearance for overhead lines is 18 feet [Page 12]. However, this increases to 21 feet when crossing roadways [Page 15]."

This source tracking serves multiple purposes:

- Allows users to verify information
- Provides context for further reading
- Maintains transparency in responses

3.5.6 User Interface Integration. The document retrieval capabilities are integrated into the chat interface. When users receive responses that include source references, they can click on the page numbers to view the original documentation. This opens in a separate window, allowing users to refer to both the chat and the source material simultaneously.

4 EVALUATION

This section provides an evaluation of the system's performance across two principal components: (1) the LangChain-enhanced LLM's performance compared to the baseline LLM, and (2) the object detection system's ability to identify utility equipment. The evaluation employs a few performance metrics, statistical tests, and visualizations to validate the system's effectiveness.

4.1 Evaluation Methodology

The evaluation strategy focuses on three dimensions for assessing the LangChain integration:

- **Pass Rates:** Assessment of query response accuracy across different categories of utility engineering tasks
- **Efficiency Metrics:** Analysis of the trade-offs between response speed, verbosity, and accuracy (important if it's used company wide by a ton of users)
- **Response Characteristics:** Measurement of system usability through response times and output verbosity

For the object detection component, the evaluation examines:

- **Classification Accuracy:** Performance in correctly identifying 11 different types of utility equipment
- **Detection Confidence:** Analysis of precision and recall across different confidence thresholds
- **Model Reliability:** Assessment of consistency across different equipment classes

4.2 LangChain Performance Analysis

The evaluation of the LangChain-enhanced system against the baseline LLM focused on three metric categories: pass rates across different query types, efficiency metrics for system performance, and response characteristics for usability assessment. The pass rates were a binary pass/fail determined by an experienced engineer. The qualifications for pass was if they would approve the logic if it were given to them for review by a designer, and anything else was a fail. Detailed test cases and raw results can be found in the project repository[2].

4.2.1 Pass Rate Analysis. Pass rates were calculated using:

$$\text{Pass Rate (\%)} = \frac{\text{Number of Correctly Answered Queries}}{\text{Total Number of Queries}} \times 100$$

Table 1 summarizes the performance across nine utility engineering categories, comparing the baseline LLM against the LangChain-enhanced system:

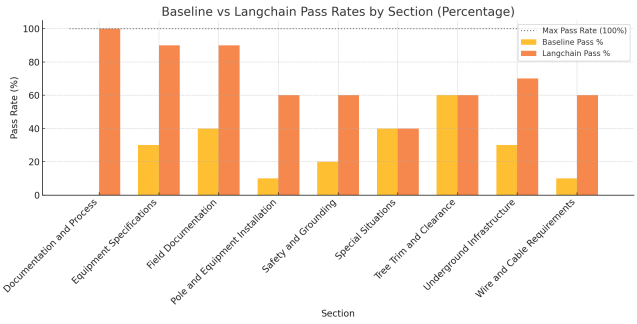

Figure 5: Pass Rates by Section (Baseline vs LangChain)

4.2.2 Efficiency Metrics. Three efficiency metrics were derived to evaluate system performance. Higher values are better for *Efficiency* and *Pass-to-Words Ratio*, while lower values are better for *Time-to-Pass Ratio*.

- **Efficiency (Words per Second):**

$$\text{Efficiency} = \frac{\text{Average Word Count}}{\text{Average Time Taken}}$$

  - Baseline: 81.16 words/second
  - LangChain: 35.66 words/second (−56.1% decrease)
- **Pass-to-Words Ratio:**

$$\text{Pass-to-Words Ratio} = \frac{\text{Pass Rate (\%)}}{\text{Average Word Count}}$$

  - Baseline: 0.223
  - LangChain: 1.877 (+741.3% increase)
- **Time-to-Pass Ratio:**

$$\text{Time-to-Pass Ratio} = \frac{\text{Average Time Taken}}{\text{Pass Rate (\%)}}$$

  - Baseline: 0.0551 seconds per pass rate percentage
  - LangChain: 0.0149 seconds per pass rate percentage (−72.9% decrease)

4.2.3 Response Characteristics. Response time and word count measurements were used to evaluate system usability:

The LangChain-enhanced system demonstrated significant improvements in response characteristics, with a 31.3% reduction in response time and a 69.0% reduction in word count while maintaining or improving accuracy across most categories.

[2]https://github.com/BilboBackends/Utility-Copilot/blob/main/test_results/Model%20Comparison%20Markdown.md

Table 1: Pass Rates by Section (Baseline vs LangChain)

Section	Total Questions	Baseline Pass Rate (%)	LangChain Pass Rate (%)
Documentation and Process	10	0.0	100.0
Equipment Specifications	10	30.0	90.0
Field Documentation	10	40.0	90.0
Pole and Equipment Installation	10	10.0	60.0
Safety and Grounding	5	20.0	60.0
Special Situations	5	40.0	40.0
Tree Trim and Clearance	5	60.0	60.0
Underground Infrastructure	10	30.0	70.0
Wire and Cable Requirements	10	10.0	60.0

4.3 Statistical Analysis

To validate that the improvements observed with LangChain integration represent genuine enhancements rather than random variation, we conducted a statistical analysis of the performance differences.

4.3.1 Methodology. A paired t-test was performed on the pass rates across all nine evaluation sections to determine the statistical significance of the improvements. The test evaluates the null hypothesis that there is no significant difference between the baseline and LangChain pass rates.

The t-statistic was calculated using:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}}$$

where:

- $\bar{d}$ is the mean difference in pass rates between LangChain and Baseline
- s_d is the standard deviation of the differences in pass rates
- n is the number of paired observations (sections)

4.3.2 Results. The analysis yielded a t-statistic of -4.27 and a p-value of 0.0027. Given that this p-value is well below the conventional significance threshold ($p < 0.05$), we reject the null hypothesis of no difference between the systems. This provides strong statistical evidence that the LangChain enhancements represent genuine improvements in system performance.

4.3.3 Implications. The statistical significance of these results has several important implications:

- The improvements in pass rates are systematic rather than due to chance
- The effectiveness of LangChain integration is consistent across different query categories
- The performance gains can be reliably expected in operational settings

This statistical validation reinforces the practical significance of the performance improvements observed in both accuracy and efficiency metrics.

4.4 Object Detection Performance

Training a robust object detection model typically requires massive datasets, yet the YOLOv7-based system achieved decent results with just 700 training images. The system identifies 11 different types of utility equipment with the following performance:

4.4.1 Core Performance Metrics.

- **Precision:** 89%
- **Recall:** 66%
- **Mean Average Precision (mAP) at 50% Confidence:** 72%

These numbers are particularly encouraging given the limited training data, suggesting the model learned effectively from the available examples.

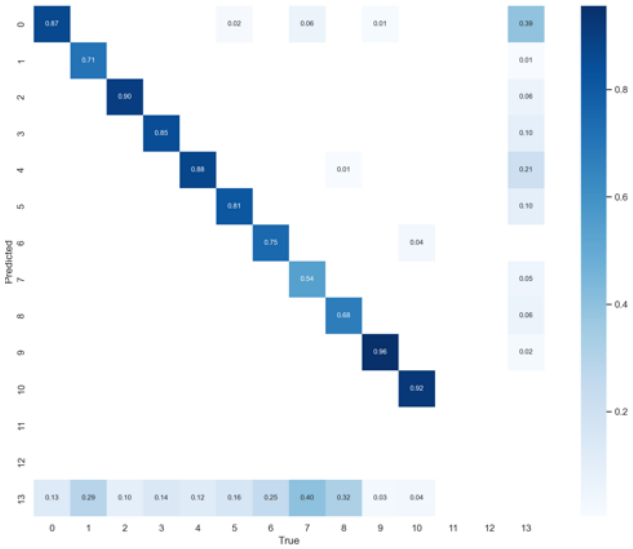

Figure 6: Object Detection Confusion Matrix showing classification accuracy across object classes. The strong diagonal pattern tells us the model rarely confuses different equipment types.

4.4.2 Classification Analysis. Looking at the confusion matrix (Figure 6), we can see:

- A clear diagonal pattern showing the model consistently gets it right
- Very few off-diagonal spots, meaning it rarely mistakes one piece of equipment for another

- Solid performance across all equipment types, even with limited examples

4.4.3 Confidence Analysis. The precision-recall curves (Figure 7) tell us two important stories:

Precision:

- The model gets more precise when it needs to be more confident
- Most equipment types maintain high precision
- Performance stays stable despite the small dataset

Recall:

- As expected, we catch fewer instances when requiring higher confidence
- Some equipment types are easier to spot than others, where the viper recloser lags behind, which would be solved with more training samples

Overall, the model demonstrates reliable utility equipment identification in the field, even with the modest training dataset. While there's room for improvement, particularly with challenging equipment types (viper recloser), the results show we're on the right track.

5 CONCLUSION AND FUTURE WORK

This paper presented Utility Copilot, a system that integrates object detection, voice interaction, and documentation retrieval to support utility engineering tasks. The evaluation demonstrates the effectiveness of both major system components, showing significant improvements over baseline approaches.

The object detection system achieves 89% precision in identifying utility equipment across 11 different classes, despite training on only 700 images. While some equipment types like the Viper Recloser show room for improvement, the overall detection performance supports reliable field use.

The LangChain enhancement demonstrates even more substantial improvements, increasing the overall query pass rate from 24.0% to 73.3%. Particularly notable is its perfect performance in documentation tasks and 90% accuracy in equipment specifications. The system not only improves accuracy but also efficiency, with 31.3% faster response times and 69.0% reduction in word count. Statistical analysis ($p = 0.0027$) confirms these improvements are significant.

Future work should focus on expanding the training dataset for challenging equipment types and improving the system's performance in areas where the pass rates remain lower, such as special situations and tree trim clearance.

The combination of visual detection and enhanced language processing enables field workers to quickly identify equipment and access relevant information, streamlining their workflow while maintaining accuracy. While both components show areas for potential improvement, the current performance levels demonstrate practical utility for field operations.

6 APPENDIX

This section provides additional technical details and parameters necessary for reproducing the system implementation.

6.1 YOLOv7 Training Parameters

The object detection model was trained with the following configuration:

- Model Base: YOLOv7
- Input Resolution: 640x640 pixels
- Confidence Threshold: 0.5
- Training Dataset: 700 annotated utility pole images
- Batch Size: 16
- Learning Rate: 0.01

6.2 Voice Processing Configuration

Voice-to-text processing uses the following parameters:

- Sample Rate: 16000 Hz
- Energy Threshold: 500
- Pause Threshold: 0.8 seconds
- Dynamic Energy Threshold: Enabled
- API Model: OpenAI Whisper-1

6.3 Document Processing Parameters

LangChain document processing configuration:

- Chunk Size: 1000 characters
- Chunk Overlap: 200 characters
- Vector Database: ChromaDB
- Embedding Model: OpenAI text-embedding-ada-002
- Top k Retrieved Documents: 3
- Similarity Metric: Cosine similarity

6.4 Chat System Configuration

The chat interface uses the following parameters:

- LLM Model: GPT-3.5-turbo
- Temperature: 0
- System Message Prompt: "You are a helpful chatbot for Electrical Utility Engineering. You can retrieve documentation and if a user greets you, reply to the user with an appropriate response including a description of yourself. One of the cool abilities that you will have is the ability to read the output of an object detection model (YOLO) from an image that a user uploads..."

7 CODE

The complete implementation of the Utility Copilot system, including all components described in this paper, are available here:
https://github.com/BilboBackends/Utility-Copilot

REFERENCES

[1] Utility Analytics. 2024. *Large Language Models in Grid Analytics.* https://utilityanalytics.com/2024/09/large-language-models-grid-analytics/
[2] Barron's. 2024. *Electric Grid Faces Challenges as Demand Grows.* https://www.barrons.com/video/electric-grid-faces-challenges-as-demand-grows/2531F557-A89F-4CBD-B136-3B7C285A23A8.html
[3] Xiang Chen, Yujie Wang, and Meng et al. Li. 2022. Insu-YOLO: An Insulator Defect Detection Algorithm Based on Multiscale Feature Fusion. *Electronics* (2022). https://www.mdpi.com/2079-9292/12/15/3210

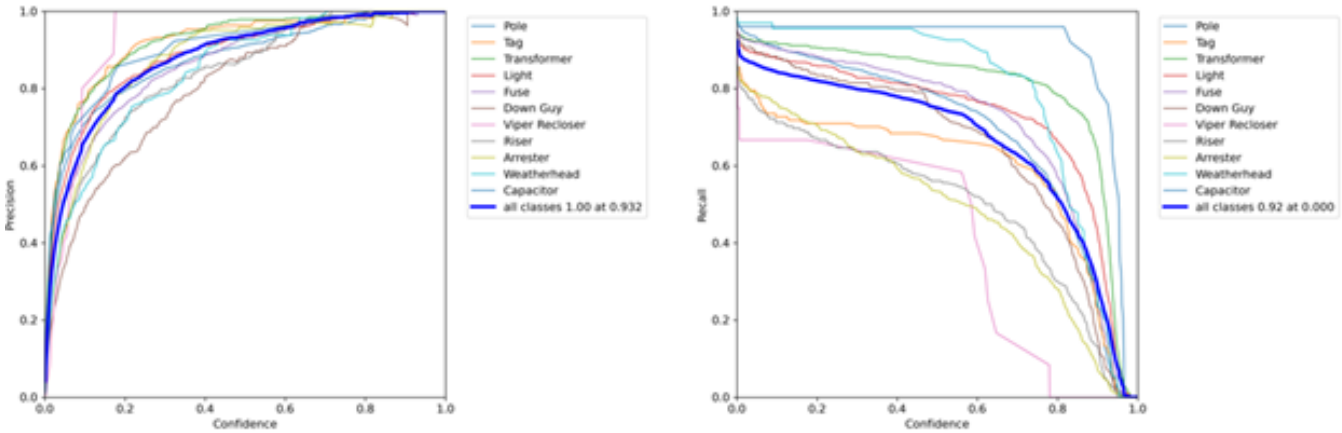

Figure 7: Precision and Recall vs. Confidence plots showing how sure the model needs to be to make accurate predictions. Left: Higher confidence means better precision. Right: Trade-offs between confidence and recall vary by equipment type.

[4] Martin Fowler. 2024. *Engineering Practices Enhanced by Large Language Models.* https://martinfowler.com/articles/engineering-practices-llm.html

[5] Jakob Limberg and Rafael et al. Gonçalves. 2024. Leveraging YOLO-World and GPT-4V LMMs for Zero-Shot Person Detection and Action Recognition in Drone Imagery. *Semantic Scholar Preprint* (2024). https://www.semanticscholar.org/paper/Leveraging-YOLO-World-and-GPT-4V-LMMs-for-Zero-Shot-Limberg-Gon%C3%A7alves/fba0aab38a140fe79baa6782b952070bab57eb21

[6] Zhao Pu, Ming Wu, and Han et al. Zhao. 2024. Automated Inspection Report Generation Using Multimodal Large Language Models and Set-of-Mark Prompting. In *Proceedings of the 41st International Symposium on Automation and Robotics in Construction (ISARC).* https://www.iaarc.org/publications/fulltext/129_ISARC_2024_Paper_115.pdf

[7] Viso.ai. 2024. *Applications of Computer Vision in Energy and Utilities Industry.* https://viso.ai/applications/computer-vision-in-energy-and-utilities-industry/