CaficulBot - Offline AI Technical Assistant for Coffee Farmers

---

By: Sergio Quintero

August 6 - 2025

Overview

CaficulBot is an AI multimodal assistant built for Colombia’s coffee farmers, fully operational without the internet. It brings expert, real time support on crop care, pests, and farm management directly to the most remote plantations, empowering farmers with decades of technical knowledge in the palm of their hand

Context and motivation

Colombia is the world's third largest coffee producer after Brazil and Vietnam, making coffee a fundamental driver of the national economy. Thus, developing technological tools tailored to local conditions to improve coffee cultivation efficiency is essential. One significant challenge faced in Colombia's rural coffee regions is limited internet access, with only 28 out of every 100 households having connectivity, which is virtually nonexistent in remote coffee plantations far from residential areas.

To address these challenges, CaficulBot emerges as an innovative solution. It provides Colombian coffee farmers real time access to extensive technical information through an AI powered conversational assistant. This assistant leverages the gemma-3n-E2B language model, fine-tuned with over 1,000 technical coffee related documents covering topics such as pest and disease management, cultivation processes, agricultural best practices, and agroecological approaches. Additionally, the assistant is enhanced with labeled image datasets of coffee pests and diseases, alongside function calling capabilities related to administrative farm management.

CaficulBot is designed not merely as a conversational tool but as an expert copilot for coffee farmers. Its goal is to deliver advanced, actionable knowledge to assist farmers directly in their daily agricultural activities such as diagnosing plant diseases and recommending treatments, and managing farm finances, administrative tasks, and daily operations. Farmers can access personalized, real time information, such as expected harvest yields or estimated profits for specific farm plots, facilitating informed and strategic decision making.

An important source of technical information

Colombia hosts one of the largest coffee research institutions globally. The National Federation of Coffee Growers' Research Center (CENICAFE). This center conducts extensive research on coffee production, quality improvement, pest and disease management, and conservation of natural resources within Colombia’s coffee growing regions. Annually, CENICAFE produces a vast amount of publicly accessible technical documentation in physical and digital formats, aiming to democratize knowledge among individuals involved in coffee cultivation. (https://www.cenicafe.org)

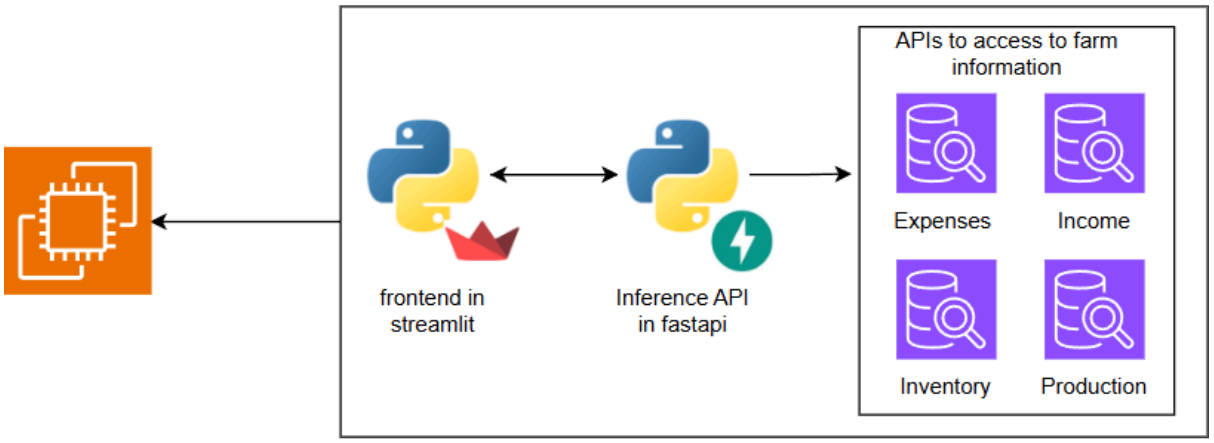
However, due to limited internet connectivity and the complexity of navigating extensive documentation, many Colombian coffee farmers rarely access this valuable information. CaficulBot bridges this gap by offering farmers an intuitive, streamlined, and practical interface to critical technical insights. It not only supports farmers in enhancing their agricultural productivity but also serves as a means to preserve and democratize decades of

accumulated technical knowledge generated by the National Federation of Coffee Growers of Colombia.

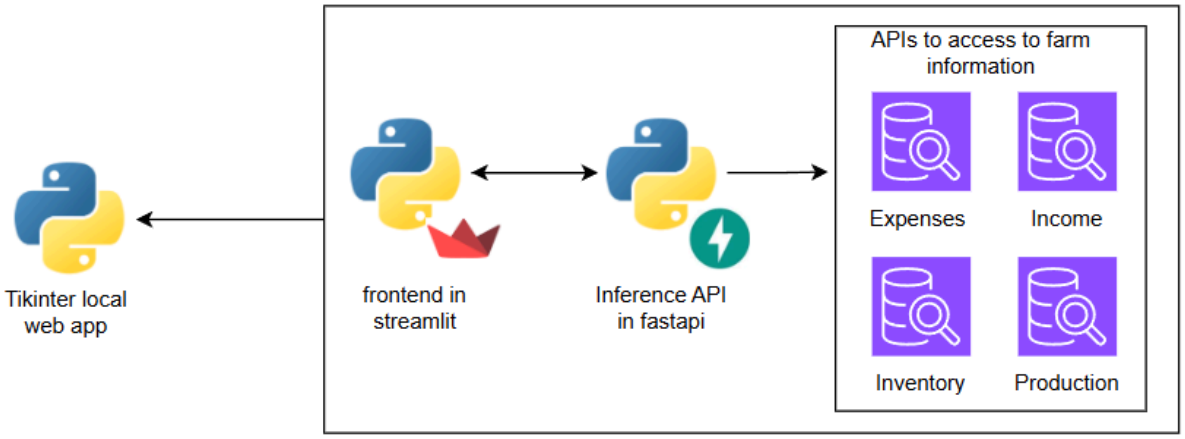
Technical Solution

CaficulBot is engineered for offline operation, leveraging the multimodal capabilities of Gemma-3N (audio-text-image). It is orchestrated by a modular, microservices based architecture, creating a lightweight and efficient solution tailored to the needs of the Colombian agricultural sector. The solution was developed to operate entirely in Spanish, as it is specifically intended for use by Colombian coffee growers. Consequently, the entire training dataset and all model generated responses are in Spanish.
The following section outlines the technical details of this solution

Architecture



Local deployment architecture



External deployment architecture

The application's architecture is built around a Python based inference API developed with the FastAPI framework. This API leverages a fine tuned Gemma-3N model, which is hosted locally within the same application and has a size of 10GB. The inference API also interacts with four local APIs, each connected to a separate SQLite database containing information on the farm's expenses, income, inventory, and production. These APIs support not only querying data but also inserting and modifying records.

The frontend of the MVP is also developed in Python, using the Streamlit framework. It allows for user interaction via voice, text, and image uploads or direct photo capture. For audio processing, the same Gemma-3N model is utilized due to its inherent audio processing capabilities. However, to address occasional technical issues during voice inference, a fallback mechanism was implemented. This fallback uses a self hosted Whisper small model from fast-whisper, approximately 100MB in size, which transcribes spoken questions into text for processing by the LLM.

For local usage the application is packaged inside a local app with the python framework tkinter, which allows the entire application to be displayed only when the executable is run.

For external usage (for demonstration purposes), the application is deployed on an EC2 g4dn.xlarge instance, which is equipped with a 10GB VRAM NVIDIA GPU. It is important to note that this cloud deployment is solely for the demo. The ultimate intention is for the application to be deployed locally on users' devices, enabling offline usage, as reflected in the GitHub repository. (url: http://52.205.44.176:8501/)

Dataset Preparation

The fine-tuning process of CaficulBot involved curating and transforming three highly specialized datasets to fully exploit the multimodal capabilities of the Gemma-3n-E2B model. These datasets included technical Q&A pairs, labeled pest and disease imagery, and function calling scenarios, each with specific preprocessing tailored to its modality.

1. **Textual Question-Answer Dataset**

The primary corpus consisted of 5,000 question-answer pairs derived from over 1,000 technical documents from CENICAFE (https://www.cenicafe.org/es/index.php/nuestras_publicaciones/index.php). The dataset was created using OpenAI model to generate the technical questions based on the documents and the the respective response, then the final dataset was loaded using the HuggingFace datasets library and mapped into a conversational format, where each sample was restructured into a messages format with user (human) and assistant (gpt) roles. This structure adheres to instruction-tuned templates required for chat based LLM fine tuning.

To adapt the text data for a multimodal model expecting image input, a placeholder dummy image was appended to each user message, ensuring all samples conformed to the expected multimodal structure while preserving purely textual semantics. This also enabled smoother training convergence under a unified input format. (URL: https://huggingface.co/datasets/sergioq2/coffe

2. Function Calling Dataset

For enabling structured actions (e.g., retrieving inventory data or calculating farm expenses), a dataset with 2,700 samples was used. Each sample included a natural language query and a corresponding structured function call. Similar to the text dataset, these were converted into the same message based chat format and appended with a dummy image for consistency in multimodal input encoding.

The function calling dataset was designed to equip the model with the ability to interact with its environment by executing predefined functions. While Gemma-3n-E2B is not inherently equipped with advanced reasoning or tool use capabilities, the goal was to extend its practical utility by enabling it to generate structured outputs corresponding to specific function calls. These functions, such as querying inventory data or calculating farm expenses, allow the model to go beyond passive Q&A and actively assist with real world tasks. By training the model to map user intents to function signatures with appropriate parameters, CaficulBot gains the ability to trigger external operations and deliver actionable outcomes, effectively bridging the gap between static knowledge and interactive system behavior. (URL: https://huggingface.co/datasets/sergioq2/functioncalling_coffedata)

3. Labeled Image Dataset for Pest and Disease Diagnosis

To enable visual recognition, a multiclass image classification dataset was integrated. This dataset consisted of 2,616 coffee plant images annotated with five specific diseases: Roya, Broca, Ojo de gallo, Mancha de hierro, and Mal rosado. Labels were converted into natural language diagnostic outputs (e.g., “La enfermedad que tiene la planta es Roya”) and embedded into message format for training. The images were loaded using the HuggingFace Image feature, and label texts were dynamically generated using class mappings to ensure readability and consistency.

All three datasets were unified into a single multimodal training set composed of vision+text pairs in the format expected by Unsloth's FastVisionModel. This composite dataset was then shuffled to balance exposure to each modality and training objective. (URL: https://app.roboflow.com/detection-3nbwx/coffe-mw9n0/2/export)

Dataset preparation challenges

During the preparation of the textual datasets, it was necessary to inject placeholder images into each sample to ensure compatibility with the Gemma-3n-E2B model’s multimodal input requirements. Although the textual question-answer pairs did not rely on visual information, the model expects every input to include both a text and an image component during fine tuning. To avoid input formatting errors and maintain a consistent training structure across all datasets, a dummy image (a blank image) was programmatically added to each user

prompt. This allowed the model to process textual only data within a multimodal framework without affecting the semantic integrity of the training samples.

Python

```
def create_dummy_image():
    dummy_array = np.zeros((224, 224, 3), dtype=np.uint8)
    return Image.fromarray(dummy_array)

def create_text_placeholder_image():
    img = Image.new('RGB', (224, 224), color='white')
    from PIL import ImageDraw, ImageFont
    draw = ImageDraw.Draw(img)
    draw.text((50, 100), "PREGUNTA DE TEXTO", fill='black')
    return img

DUMMY_IMAGE = create_dummy_image()

def add_dummy_image_to_text_dataset(text_dataset):
    modified_dataset = []

    for sample in text_dataset:
        new_sample = {
            "messages": [
                {
                    "role": "user",
                    "content": [
                        {"type": "text", "text":
sample["messages"][0]["content"][0]["text"]},
                        {"type": "image", "image": DUMMY_IMAGE}
                    ]
                },
                {
                    "role": "assistant",
                    "content": sample["messages"][1]["content"]
                }
            ]
        }
        modified_dataset.append(new_sample)

    return modified_dataset
```

Fine Tuning Strategy

The fine tuning was performed on the unsloth/gemma-3n-E2B-it variant using the Unsloth library, which provides efficient low rank adaptation (LoRA) with 4-bit quantization and full multimodal support. LoRA (Low-Rank Adaptation) was chosen as the fine tuning method because it offers an efficient way to adapt large language models without overwriting their existing knowledge, a phenomenon known as catastrophic forgetting. Instead of updating all model parameters, LoRA introduces a small set of trainable parameters that modify the behavior of specific layers while keeping the original weights frozen. This approach allows the model to retain its general language and reasoning abilities while integrating new, domain specific knowledge, in this case, technical expertise related to coffee. Additionally, LoRA does not require massive datasets to be effective, making it ideal for scenarios like this where carefully curated but relatively small datasets are available. It enables a balanced fusion between the model's pre-trained capabilities and the specialized information introduced during fine tuning.

The selection of the Unsloth library was a strategic decision driven by its profound impact on training efficiency. Unsloth offers a dramatic acceleration in fine-tuning speed, up to 30x faster, and a reduction in memory usage by up to 60% compared to standard Hugging Face implementations, all without any loss in accuracy. This performance gain is achieved through sophisticated low level optimizations, including a manual autograd engine tailored for LoRA and the rewriting of all compute kernels in OpenAI's Triton language. For this project, these advantages were critical, enabling rapid experimentation and iteration. The reduced memory footprint allowed for training with larger batch sizes on the available hardware, maximizing GPU utilization and leading to a more efficient and effective fine tuning workflow.

Training process

Gemma-3n is a highly optimized multimodal architecture designed for efficient deployment across diverse devices, from mobile phones to edge computing environments. Its internal structure is composed of distinct parameter groups: text, vision, audio, and Per-Layer Embedding (PLE), which can be selectively activated to optimize for task-specific requirements. For visual processing, the model integrates a high-performance MobileNet-V5 encoder, enabling rapid and accurate interpretation of image data, which is essential for tasks like pest and disease recognition in CaficulBot. One of its standout features is the MatFormer (Matryoshka Transformer) architecture, which nests smaller models inside a larger framework. This allows selective activation of sub models, reducing memory and compute consumption depending on the complexity of the request. Furthermore, PLE caching enables the model to offload layer-specific embedding computations to fast external storage, dramatically lowering runtime memory usage. Conditional parameter loading extends this flexibility by allowing the model to bypass loading audio or vision modules when not needed, making it possible to operate with as few as 1.91 billion effective parameters out of a total that exceeds 5 billion. This modular and efficient design was instrumental in fine tuning CaficulBot with a mix of vision, text, and structured task datasets, all while maintaining low resource requirements and high inference performance.

Model Configuration

- Base Model: gemma-3n-E2B-it (Instruct-tuned, multimodal, 6B parameters)
- Quantization: 4-bit (load_in_4bit=True), reducing memory usage significantly while preserving performance.
- Adapter Method: LoRA with r=32, alpha=64, dropout=0.03, enabling lightweight, scalable parameter-efficient training.
- Layers Fine-Tuned:
    - Vision layers (ViT encoder)
    - Language layers (decoder)
    - Attention and MLP modules
- PEFT Target Modules: All linear layers, plus explicitly saving lm_head and embed_tokens.

Python

```python
model, tokenizer = FastVisionModel.from_pretrained(
    "unsloth/gemma-3n-E2B-it",
    load_in_4bit = True,
    use_gradient_checkpointing = "unsloth",
)


model = FastVisionModel.get_peft_model(
    model,
    finetune_vision_layers     = True,
    finetune_language_layers   = True,
    finetune_attention_modules = True,
    finetune_mlp_modules       = True,

    r =32,
    lora_alpha = 64,
    lora_dropout = 0.03,
    bias = "none",
    random_state = 3407,
    use_rslora = True,
    loftq_config = None,
    target_modules = "all-linear",
    modules_to_save=[
        "lm_head",
        "embed_tokens",
    ],
)
```

Training Configuration

- Trainer: SFTTrainer from TRL (transformers + PEFT integration)
- Batch Size: 4 per device with gradient accumulation of 8 steps
- Optimizer: adamw_torch_fused with cosine learning rate scheduler
- Learning Rate: 2e-4
- Epochs: 3
- Max Gradient Norm: 0.3
- Gradient Checkpointing: Enabled to reduce memory overhead
- Warmup: 5% of total steps
- Total Steps: 100 (focused training phase for quick iteration and evaluation)

During training, the UnslothVisionDataCollator ensured correct batching of multimodal inputs. The model was trained to simultaneously perform:

- Natural language understanding and generation from technical queries,
- Visual diagnosis based on pest and disease images, and
- Structured function calling from user intents.

Final Model Publishing

After training convergence, the fine-tuned model was merged and pushed to the Hugging Face Hub under the name: sergioq2/genna-3N-finetune-coffe_q4_off

```python
if True:
    model.push_to_hub_merged(
        "sergioq2/gemma-3N-finetune-coffe_q4_off", tokenizer,
        token = HUGGINGFACE_HUB_TOKEN
    )
```

This makes it publicly accessible for inference, benchmarking, and future incremental tuning.

Model Consumption and Inference API

With the fine-tuned model (sergioq2/gemma-3N-finetune-coffe_q4_off) published on the Hugging Face Hub, the next step was to operationalize it through a robust and scalable inference API. The first approach was to try with Ollama to run it locally, but it was not possible since the multi modal capabilities were lost with this specific model, so the approach was to develop a custom API using FastAPI, designed to handle both text and image inputs, thereby exposing the full multimodal and tool-use capabilities of Caficulbot.

Inference API Architecture

The core of the service is a FastAPI application that orchestrates model loading, request handling, and interaction with external tools.

1. Model Loading: To ensure high availability and low latency, the model is loaded into GPU memory upon application startup using a startup event handler. A transformers pipeline of type "image-text-to-text" is instantiated, pointing to the local path of the downloaded model. Key configurations for performance include:
    - device="cuda": Ensures all computations are performed on the GPU.
    - torch_dtype=torch.bfloat16: Utilizes 16-bit brain floating-point precision, which significantly reduces the memory footprint and accelerates inference speed with minimal impact on accuracy.
    - local_files_only=True: Guarantees that the model is loaded from the local disk, preventing unexpected downloads and ensuring deployment consistency.

2. API Endpoint: A single POST endpoint, /ask, serves all user requests. It is designed to accept multipart form data, which can include a text question and an optional image file. This unified endpoint simplifies client-side implementation by routing all interactions through one interface.

Multimodal and Tool-Use Logic

API’s versatility lies in its ability to dynamically adapt its behavior based on the input modality and the user's intent, a process governed by sophisticated prompt engineering and conditional logic.

1. Dual-Prompt Strategy: Two distinct system prompts are employed to guide the model's responses:
    - SYSTEM_PROMPT_IMAGE: Used exclusively for requests containing an image. It instructs the model to act as a coffee expert, analyze the provided image in detail, and respond in natural language without attempting to use any tools. This isolates the visual diagnosis task.
    - SYSTEM_PROMPT: Used for text-only requests. This prompt is more complex, defining the model's persona and providing strict instructions on tool usage. It explicitly directs the model to respond in natural language for general queries (e.g., greetings, pest control advice) and to generate a specific JSON structure only when questions related to inventory or expenses are detected (e.g., {"tool": "inventario_consulta", "argumentos": "producto=fertilizante"}). This structured output is the key to enabling function calling.

2. Request Handling Flow
    - Image Requests: If an image is detected in the request, the API reads the image bytes, converts it into a Pillow (PIL) Image object, and pairs it with the user's question. It then sends this multimodal payload to the pipeline using the SYSTEM_PROMPT_IMAGE.
    - Text and Tool-Use Requests: If no image is provided, the API uses the SYSTEM_PROMPT to process the text question. After receiving the model's generated text, it passes the response to a custom parser, parse_tool_call.

3. Custom Tool-Calling Mechanism: Since the model was fine tuned to generate specific JSON strings for function calls, a lightweight, custom mechanism was implemented to interpret and act on these outputs:
    - Parsing: The parse_tool_call function attempts to decode the model's response as a JSON object. If successful, it extracts the tool name and arguments.
    - Execution: The API then routes the request to the appropriate function. For instance, if tool_name is "inventario_consulta", it calls the consultar_inventario_api function, which in turn makes an HTTP GET request to a separate microservice (e.g., at http://localhost:8001).
    - Response Synthesis: The data retrieved from the external API is then formatted into a user friendly, natural language sentence (e.g., "Quedan disponibles: 50 unidades de fertilizante."). If no tool call is detected, the model's original natural language response is returned directly.

This architecture creates an efficient system where the fine-tuned model acts not just as a knowledge base but as a reasoning engine that can delegate tasks to external tools, seamlessly integrating visual analysis, technical Q&A, and real time data retrieval into a single, cohesive user experience.