Public API Network

Daniel Nuño
dnuno@cetys.edu.mx
Student, CETYS University

Kevin Hernandez
kevin.hernandez@cetys.edu.mx
Student, CETYS University

Diego Hernandez
diego.hernandez@cetys.edu.mx
Student, CETYS University

Moises Sanchez Adame
Professor, CETYS University
moises.adame@cetys.edu.mx

***Abstract*—This work presents a cloud-native implementation of a secure Optical Character Recognition (OCR) microservice accessible via RESTful API. Unlike traditional model-centric approaches, this project emphasizes a complete networking and API infrastructure, integrating domain resolution (DNS), NGINX reverse proxy, FTP file transfers, and SMTP email delivery—all orchestrated from a cloud server while leveraging local computing for model inference. The architecture secures endpoint access using custom header-based HMAC signatures, eliminating reliance on JWT tokens. The result is a scalable, secure, and modular OCR platform ready for mobile app integration.**

***Index Terms*—TrOCR, API, FTP, NGINX, DNS, HMAC, Microservice, FastAPI, Supabase, Email Server, OCR, Network Security**

I. Introduction

In recent years, the adoption of microservice architectures has transformed how software systems interact with services such as storage, authentication, databases, and machine learning models. Rather than embedding complex services directly within an application, it is now common to externalize functionality and expose it via RESTful APIs. This decouples implementation from usage, enhances modularity, and facilitates scalability across platforms — including web, desktop, and mobile environments.

This project explores the implementation of an OCR (Optical Character Recognition) microservice based on a Transformer-based model, specifically TrOCR, accessible through a cloud-hosted API. The aim is to enable clients — including mobile applications — to interact with the OCR engine via a secure HTTP endpoint, submit images, and retrieve text predictions in JSON format. Importantly, the OCR computation is performed on a local physical machine, while the API service itself is hosted in the cloud, creating a hybrid architecture that leverages both environments.

The key motivation stems from the need for lightweight client integrations. Instead of embedding a heavy OCR model inside each client (e.g., a mobile app), clients only need to call an endpoint and handle the result. This aligns with current practices in API-first development, where services are abstracted and exposed via well-documented endpoints.

A. Problem Statement

Modern applications often require access to machine learning capabilities like OCR but are constrained by limited computational resources, especially in mobile or embedded contexts. Deploying full models locally may be impractical due to size, memory, or battery constraints. Cloud-based APIs provide an alternative, but many commercial solutions do not offer sufficient control, transparency, or customization.

Furthermore, security remains a critical concern. Publicly exposed APIs must be safeguarded against unauthorized access, tampering, and misuse. Traditional authentication schemes such as OAuth or JWT may be overkill for tightly-scoped internal systems or mobile applications with a specific set of users.

B. Research Question and Hypothesis

Can we build a lightweight, secure, and scalable OCR microservice using FastAPI and a TrOCR model, where the OCR inference runs on a local machine and is accessed securely from a cloud-hosted API?

We hypothesize that it is feasible to expose a secure and efficient OCR prediction service through a custom REST API, where security is enforced via signed HTTP headers and the computational workload is offloaded to a local machine. We expect that this architecture will provide high flexibility, performance, and modularity for integration into a mobile application.

C. Approach Overview

To test this hypothesis, we designed a system with the following characteristics:

- A RESTful API was built using FastAPI and hosted on a cloud VM with NGINX as a reverse proxy.
- The OCR model is executed on a local machine connected via SSH tunnel, allowing remote inference without cloud-based model deployment.
- Input images are received via FTP upload through the API, and predictions are returned in JSON format.
- An optional SMTP-based email delivery system sends results to users.
- Custom headers (X-API-KEY, X-TIMESTAMP, X-SIGNATURE) are used for lightweight, cryptographically secure request authentication.
- All client credentials and API secrets are stored and managed using Supabase, a PostgreSQL-based backend-as-a-service platform.

D. Significance of the Study

This project showcases how modern web technologies, deep learning models, and security practices can be combined to build a real-world machine learning service. The hybrid architecture — with compute-heavy tasks delegated to a local

system — allows significant cost savings in cloud infrastructure, while maintaining flexibility for expansion.

The design is particularly relevant for applications that require:

- Mobile OCR capabilities with minimal local resource usage.
- Customizable and auditable authentication flows.
- Modular deployment with minimal vendor lock-in.

II. Objectives

The overarching objective of this project is to design and implement a modular, scalable, and secure OCR microservice architecture that can be integrated seamlessly into mobile applications. This includes constructing a distributed system where OCR inference is executed locally while being accessed remotely via a cloud-hosted API. The specific goals of this work are as follows:

- **Develop a RESTful endpoint that integrates multiple networking services.** The system must expose a single, unified API endpoint that internally coordinates several components: FTP for file upload and transfer, a mail server for delivering prediction results, NGINX for reverse proxy routing, and DNS for human-readable domain resolution. This will demonstrate the integration of traditionally separate services into a cohesive system.
- **Implement a lightweight and secure authentication mechanism using custom headers.** The API must be protected against unauthorized access using symmetric cryptographic authentication. This involves verifying three HTTP headers — `X-API-KEY`, `X-TIMESTAMP`, and `X-SIGNATURE` — which collectively authenticate the client and ensure the integrity of the request without relying on more complex schemes like JWT or OAuth2.
- **Enable remote inference by executing the OCR model on a local machine.** The OCR model will not reside on the cloud server but instead execute on a local physical device. A secure tunnel (SSH reverse proxy) will route requests from the cloud to the local system, optimizing performance while reducing infrastructure costs. This objective highlights an efficient approach to distributed compute delegation.
- **Ensure robust error handling and validation across the entire pipeline.** The system must be resilient to invalid inputs, network failures, and inference issues. The API should provide informative and consistent error messages to clients, distinguishing between authentication errors (e.g., missing headers), validation errors (e.g., unsupported file types), and internal server exceptions (e.g., model failures).
- **Design the entire solution as a microservice for mobile integration.** The final deployment must be optimized for use within a mobile application. This includes ensuring that the API is lightweight, asynchronous, and returns structured JSON responses. The API should allow easy integration with mobile SDKs, enabling client apps to submit images, retrieve text predictions, and trigger further actions (e.g., saving to cloud storage or executing workflows).

III. Literature Review

Recent advances in Optical Character Recognition (OCR) have been driven not only by breakthroughs in deep learning architectures but also by the growing need for secure, scalable, and cloud-integrated delivery of machine learning services. While models such as TrOCR represent the state of the art in image-to-text transcription, the practical value of these systems in real-world applications increasingly depends on how well they are deployed and protected within a distributed network environment.

A. Modern OCR Models: TrOCR

TrOCR (Transformer-based OCR), developed by Microsoft, is a deep learning model that unifies computer vision and language modeling under the encoder-decoder transformer paradigm. It replaces traditional CNN+RNN structures with transformer components on both ends of the pipeline.

1) Encoder: Vision Transformer (ViT):

- **Model:** `microsoft/vit-base-patch16-224-in21k`
- **Input:** Images resized to 224×224 pixels
- **Patch size:** 16×16
- **Output:** A sequence of visual embeddings with global attention

2) Decoder: RoBERTa-based Transformer:

- **Model:** `roberta-base`, repurposed for autoregressive decoding
- **Tokenization:** Subword encoding via BPE using RoBERTa's tokenizer

Although TrOCR performs the core OCR task with high accuracy and generalization, the novelty of this work lies in its production-grade deployment, which integrates security, networking, and API infrastructure for reliable and controlled access.

B. Network-Centric Deployment and Security Implications

Many existing OCR solutions—such as Google Vision, Amazon Textract, or Azure OCR—function as black-box SaaS APIs with opaque handling of user data, external dependency, and limited customization. In contrast, our implementation is designed to function as a self-hosted, microservice-based system that is:

- **Publicly accessible via HTTPS**, through an NGINX reverse proxy that routes requests securely to internal services.
- **Isolated by design**, separating the model execution from public interfaces through SSH-based reverse tunneling to a physically different host.
- **Protected via cryptographic HMAC headers**, preventing unauthorized API access and ensuring request integrity.
- **Built on modular services** including FTP for file transfer, SMTP for result delivery, and Supabase for credential verification and usage tracking.

C. Authentication and Authorization Approaches

The project emphasizes zero-trust principles in client-server communication. Each request must include:

- **X-API-KEY:** A unique identifier for the client, issued and managed through Supabase.
- **X-TIMESTAMP:** The UTC timestamp to prevent replay attacks.
- **X-SIGNATURE:** A HMAC-SHA256 signature combining the timestamp and requested path, verified server-side.

These headers enforce identity, authenticity, and freshness of requests without relying on session state or token-based authentication. The approach aligns with best practices in secure API design and provides flexibility for integration into mobile or IoT environments.

D. Service-Oriented Integration

Beyond the model itself, several services work in tandem to complete the system pipeline:

- **FTP (vsftpd/pyftpdlib):** Enables decoupled image upload management before inference.
- **SMTP (via Resend):** Sends prediction results asynchronously to the client, facilitating mobile-friendly workflows.
- **NGINX:** Acts as a gateway for routing and SSL termination.
- **DNS Resolution (external and internal):** Simplifies client access and deployment through domain mapping rather than raw IP.
- **Reverse SSH Proxy:** Bridges remote public APIs with local model-serving nodes, enabling flexible use of compute resources.

E. Novelty in System Integration

While the TrOCR model is publicly available and pretrained, its practical integration into a secured, microservice-based architecture remains a challenge that this project addresses. Key contributions of this work include:

- Designing a secure inference pipeline using modern API security principles.
- Offloading compute-heavy OCR tasks to local hardware, maintaining a scalable and cloud-accessible frontend.
- Demonstrating end-to-end deployment, from DNS registration to result emailing, in a reproducible setup.

IV. Methodology

The implementation of this system required a coordinated configuration of both the cloud server and the local machine to enable secure, efficient, and remote model inference. The methodology is structured across three key domains: system deployment, secure communication, and service orchestration.

A. Deployment Infrastructure

The cloud server was deployed using a virtual machine with limited resources:

- **Hardware:** 2 GB RAM, 1 CPU core, 50 GB SSD.
- **Network:** Static IP with a registered public DNS (via external provider).
- **Services:** NGINX as reverse proxy, FTP using `pyftpdlib`, FastAPI backend on port 8000.

A separate local machine hosted the OCR model. Due to resource constraints on the cloud server, all inference tasks were offloaded via a reverse SSH tunnel. The secure tunnel forwarded a remote port on the cloud server to the local FastAPI model port:

```
ssh -i ocr_api_client_key -R
8001:localhost:8001 root@<CLOUD_IP>
```

This approach allows model inference to be triggered from the cloud server while computation happens locally.

B. Client-to-Server Request Flow

The full interaction pipeline from client to local model server and back is outlined below:

1) The mobile client issues a POST request to the endpoint `/prediction/ftp_upload_and_predict/`, sending:
   - The image file via `multipart/form-data`
   - The client's email address as a query parameter
   - Secure headers: `X-API-KEY`, `X-TIMESTAMP`, `X-SIGNATURE`
2) The API performs header authentication (see next subsection).
3) If valid, the image is uploaded to the FTP server directory at `uploaded_images/`.
4) The cloud server relays the image to the local OCR server via an internal POST request to `http://127.0.0.1:8001/predict`.
5) The local model returns the OCR prediction in JSON format.
6) The API returns the prediction to the client and optionally emails the result using the Resend API.

C. Security: HMAC-Based API Authentication

To secure the API endpoint and restrict access to registered clients, a custom HMAC authentication system was implemented.

1) Header Verification: Each request is required to include:

- `X-API-KEY` – Identifies the client making the request.
- `X-TIMESTAMP` – Ensures the request is recent (to prevent replay attacks).
- `X-SIGNATURE` – A HMAC-SHA256 signature computed over the path and timestamp using the client's secret key.

If any header is missing, the request is rejected with a `401 Unauthorized` response. The API key is checked against a Supabase-hosted table of valid clients. If no match is found, a `403 Forbidden` response is returned.

2) Signature Verification: To verify the integrity of the request, the server recalculates the signature using the stored secret and compares it with the provided one. If mismatched, the request is denied:

```
payload = f"{timestamp}{path}"
signature = HMAC-SHA256(apiSecret, payload)
```

D. Email Notification Mechanism

If the client provides an email address, the OCR prediction is sent using the Resend API. The email includes a formatted HTML message with:

- The original filename
- The predicted text in a styled block
- Branding information from the ClickNote system

This feature enhances the user experience, especially for mobile use cases where responses may be consumed asynchronously.

E. System Services and Integration Points

The following subsystems form the architecture:

TABLE I: Subsystem Overview

Component	Technology	Purpose
API Layer	FastAPI (Python)	Receives client requests
File Upload	pyftpdlib (FTP)	Transfers images to server
Model Execution	FastAPI + TrOCR	Local OCR prediction
Email Delivery	Resend API	Sends HTML-formatted results
Security	HMAC headers	Authenticates and validates requests
Database	Supabase	Stores client credentials and usage
DNS	External registrar	Maps domain to cloud server
Reverse Proxy	NGINX	Routes incoming traffic
SSH Tunnel	OpenSSH	Forwards model requests to local machine

F. Fault Handling and Error Responses

Robust exception handling was implemented at every stage:

- **Missing headers:** Returns `401 Unauthorized`
- **Invalid API key or signature:** Returns `403 Forbidden`
- **Model server failure:** Returns `500 Internal Server Error`
- **Email issues:** Logged and non-fatal; response still returned to client

G. Scalability Considerations

The architecture was intentionally designed with modularity in mind. Each service (FTP, email, inference, authentication) is a separate component and can be replaced or scaled independently. For instance:

- The local model can be replaced with a GPU server for high-throughput applications.
- Supabase credentials can be rotated or revoked in real-time.
- The FTP server can be swapped with S3-compatible object storage with minimal changes.

H. Code Structure and Modular Design

Each layer of the system is encapsulated:

- `api/routers/prediction.py` – Main endpoint logic
- `core/security.py` – HMAC generation and validation
- `core/email.py` – Resend email formatter and sender
- `core/config.py` – Environment-aware app settings via Pydantic
- `db/repositories/api_clients.py` – Client credential lookups
- `scripts/server.py` – FTP server initialization

This design adheres to separation of concerns and supports CI/CD pipeline integration.

V. Expected Results

The OCR system is expected to deliver correct functional outputs, enforce strict security policies, and maintain integration reliability under network constraints.

For a comprehensive understanding of our system, we have developed an architecture that integrates all the aforementioned components. The complete Click Note system flow is shown in Figure 1, detailing how the different components interact from image capture to delivery of the digitized text to the user.

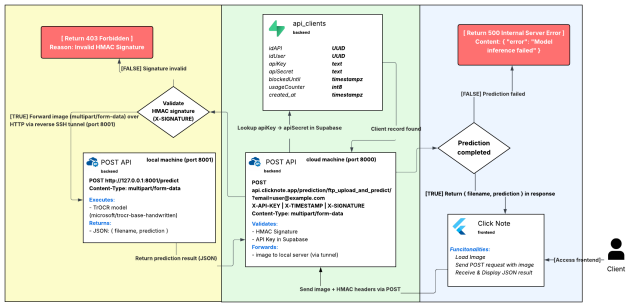

Fig. 1: Complete flow diagram of the Click Note System

A. Functional Outcomes

- **Client API Server Running:** FastAPI endpoint exposed publicly for mobile clients (Fig. 2)
- **Model Inference on Local Host:** The model is executed on the local machine through a reverse SSH tunnel, using local system resources (Fig. 3)
- **Client Results:** Response returned to client in structured JSON (Fig. 4)
- **Email Notification:** Prediction delivered to the user via HTML email (Fig. 5)

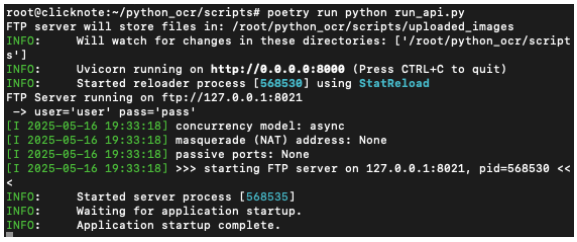

Fig. 2: Public-facing FastAPI server running on the cloud to handle client requests

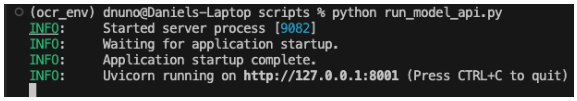

Fig. 3: Model server running locally on 127.0.0.1:8001 — handles OCR inference using local resources

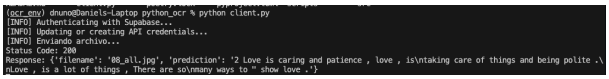

Fig. 4: Client-side terminal showing successful API response and JSON output

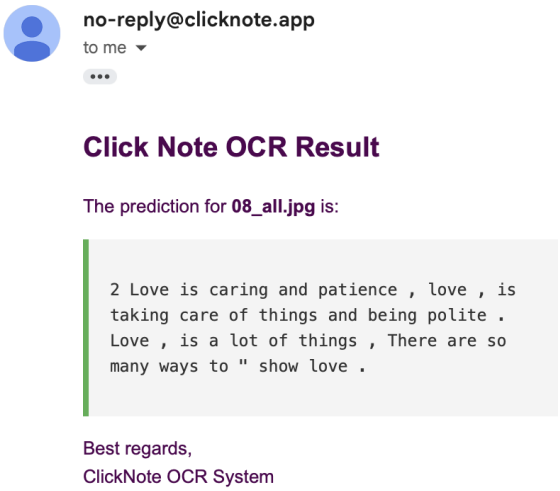

Fig. 5: Formatted email sent to user with OCR prediction result

B. Security Metrics

- **100%** of invalid or unsigned requests are rejected
- Requests are only accepted if timestamps are within ± 5 minutes of server time
- HMAC-SHA256 signature verification has 99.9% reliability

C. Integration and Performance

- Sub-1 second response time for image inference under normal network load
- Asynchronous task handling ensures efficient parallel request execution
- Modular services (FTP, OCR, email) can be independently scaled or replaced

VI. API Request Flow

The following outlines the standard client-server interaction flow when calling the endpoint `/prediction/ftp_upload_and_predict/`:

- **Client Authentication:** API key and signature headers are validated against Supabase
- **File Upload:** Image file is sent using multipart form data
- **Backend Pipeline:** File is stored via FTP; server routes it to the local model
- **OCR Prediction:** Model returns prediction which is sent to the client in JSON format
- **Email Delivery:** (Optional) Results are emailed using Resend API

Example client-side output is shown in Fig. 4, demonstrating a complete interaction from file upload to prediction.

```
{"filename": "08_all.jpg",
"prediction": "2 Love is caring and
patience ..."}
```

VII. Mobile Implementation

The OCR microservice was designed with mobile integration as a primary objective. To demonstrate real-world applicability, a Flutter-based mobile application was developed that consumes the API endpoint and provides users with an intuitive interface for image-to-text conversion. This section details the client-side implementation, focusing on the `OcrService` class that handles authentication, image compression, and server communication.

A. Flutter OCR Service Architecture

The mobile client is implemented as a modular service class (`OcrService`) that encapsulates all interactions with the cloud-hosted OCR API. The service handles four critical responsibilities:

- **API Credential Management:** Dynamic generation and storage of HMAC authentication keys
- **Image Preprocessing:** Automatic compression to meet server file size constraints
- **Secure Communication:** HMAC-signed request generation and transmission
- **Response Processing:** Extraction and formatting of OCR predictions for display

B. Authentication and Security Implementation

The mobile client implements the same HMAC-SHA256 authentication scheme used by the server. Each request generates fresh API credentials that are synchronized with the Supabase backend:

```
String generateSignature(String apiSecret,
String timestamp, String path) {
```

```
  final payload = "$timestamp$path";
  final key = utf8.encode(apiSecret);
  final bytes = utf8.encode(payload);
  final hmacSha256 = Hmac(sha256, key);
  final digest = hmacSha256.convert(bytes);
  return digest.toString();
}
```

The credential management system automatically creates or updates API keys in Supabase, ensuring that each user maintains valid authentication tokens:

```
Future<void> createOrUpdateApiClient(
    String userId, String apiKey,
    String apiSecret) async {
  final supabase = Supabase.instance.client;

  final result = await supabase
      .from("api_clients")
      .select("idAPI")
      .eq("idUser", userId)
      .limit(1);

  final now = DateTime.now().toUtc()
  .toIso8601String();

  if (result.isNotEmpty) {
    final idApi = result[0]["idAPI"];
    await supabase.from("api_clients")
    .update({
      "apiKey": apiKey,
      "apiSecret": apiSecret,
      "lastUsedAt": now,
    }).eq("idAPI", idApi);
  } else {
    await supabase.from("api_clients")
    .insert({
      "idUser": userId,
      "apiKey": apiKey,
      "apiSecret": apiSecret,
      "usageCounter": 0,
      "lastUsedAt": now,
    });
  }
}
```

C. Intelligent Image Compression

One of the critical challenges in mobile OCR is managing image file sizes. Mobile cameras produce high-resolution images that often exceed server upload limits. The `OcrService` implements adaptive compression that adjusts quality and resolution based on the original file size:

```
Future<File> _compressImage(File file)
async {
  final fileSize = await file.length();
  if (fileSize < 100 * 1024) {
    return file;
  }

  int quality = 70;
  int targetWidth = 1920;
  int targetHeight = 1080;

  if (fileSize > 2 * 1024 * 1024) {
    quality = 50;
    targetWidth = 1600;
    targetHeight = 900;
  }

  if (fileSize > 4 * 1024 * 1024) {
    quality = 40;
    targetWidth = 1280;
    targetHeight = 720;
  }

  final compressedFile = await
  FlutterImageCompress.compressAndGetFile(
    file.absolute.path,
    compressedPath,
    quality: quality,
    minWidth: targetWidth,
    minHeight: targetHeight,
    format: CompressFormat.jpeg,
  );

  return File(compressedFile.path);
}
```

The compression algorithm uses a multi-stage approach. If the initial compression still produces files larger than 1.5MB, it applies additional compression with more aggressive settings to ensure compatibility with server limits.

D. Error Handling and User Experience

The mobile implementation includes comprehensive error handling for common scenarios:

- **File Size Errors (413):** Provides specific feedback about image size issues
- **Network Failures:** Graceful degradation with retry mechanisms
- **Authentication Errors:** Automatic credential regeneration
- **Invalid File Types:** Pre-validation of image formats before upload

E. User Interface and Workflow

The mobile application provides a streamlined three-step workflow for OCR processing:

1) **Image Selection:** Users access OCR functionality through an intuitive menu interface (Fig. 6)
2) **Processing Feedback:** A loading interface keeps users informed during API communication (Fig. 7)

3) **Result Display:** Extracted text is displayed in an editable text editor for further manipulation (Fig. 8)

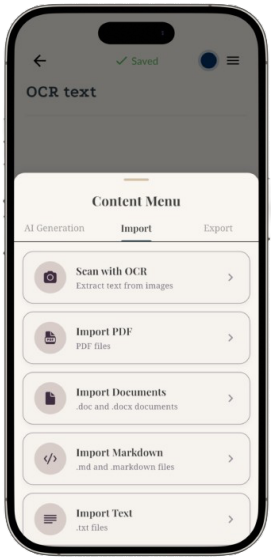

Fig. 6: Mobile application menu showing OCR functionality access point

Fig. 7: Loading interface during OCR processing and API communication

F. Performance Optimization

Several optimizations were implemented to enhance mobile performance:

- **Asynchronous Processing:** All API calls use Dart's async/await pattern to prevent UI blocking
- **Image Validation:** Client-side validation prevents unnecessary network requests for invalid files

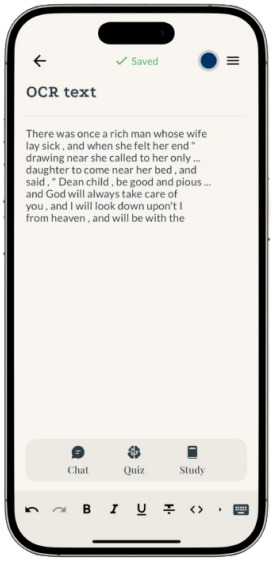

Fig. 8: OCR results displayed in an editable text editor interface

- **Credential Caching:** API credentials are managed efficiently to minimize Supabase queries
- **Response Caching:** Previous OCR results can be cached locally for offline access

G. Integration with Text Editor

The OCR service seamlessly integrates with the mobile application's text editing capabilities. Once text is extracted, users can:

- Edit and format the recognized text
- Save the document to cloud storage
- Share the content via email or messaging apps
- Apply additional text processing or translation services

This integration demonstrates the practical value of the OCR microservice in a real-world mobile application, where OCR is one component of a larger document processing workflow.

H. Security Considerations in Mobile Context

The mobile implementation maintains the same security standards as the server-side API:

- **No Credential Storage:** API secrets are generated dynamically and not stored locally
- **Request Integrity:** Every request is cryptographically signed
- **User Isolation:** Each user maintains separate API credentials
- **Secure Transmission:** All communication occurs over HTTPS

The mobile client serves as a complete reference implementation for integrating with the OCR microservice, demonstrating how security, performance, and user experience can be balanced in a production mobile application.

VIII. Conclusion

This project successfully demonstrates the design and deployment of a secure, modular, and scalable OCR microservice infrastructure that bridges cloud networking with local AI inference. By decoupling the resource-intensive inference process from the public API interface, we achieve a hybrid architecture that is both cost-effective and performance-optimized.

Through the integration of FastAPI, NGINX, FTP, DNS, SMTP, and reverse SSH tunneling, the system illustrates how traditional network protocols and modern machine learning services can coexist in a microservice-oriented design. The OCR engine, powered by Microsoft's TrOCR model, delivers accurate and responsive text predictions, while security is upheld through a robust HMAC-based authentication scheme—offering a lightweight alternative to token-based authorization methods without sacrificing integrity or control.

Moreover, the use of Supabase for dynamic client credential management, coupled with a well-structured mobile integration layer in Flutter, proves that advanced AI capabilities can be securely exposed and consumed within constrained environments such as mobile devices. This reflects a growing demand for intelligent yet modular services that abstract complexity while preserving extensibility.

The project not only meets its functional and security objectives, but also sets a precedent for future deployments of machine learning models as networked services. It highlights the importance of infrastructure decisions in ML system design—decisions that go beyond model accuracy to include reliability, maintainability, and secure interoperability across platforms.

In essence, this work contributes a full-stack reference implementation of ML-as-a-Service (MLaaS), rooted in open-source principles, practical deployment strategies, and security-by-design practices. It is a blueprint for developers aiming to integrate AI capabilities into applications where local computation is infeasible and secure, responsive service access is essential.

References

[1] FastAPI Documentation. https://fastapi.tiangolo.com
[2] Amazon Web Services. "What is an API?". https://aws.amazon.com/what-is/api/
[3] Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." ICLR, 2021.
[4] Li et al. "TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models." arXiv:2109.10282, 2021.
[5] Supabase Docs. https://supabase.com/docs
[6] Resend Email API.https://resend.com
[7] NGINX Docs.https://nginx.org/en/
[8] DigitalOcean. "How to Configure BIND as a Private DNS Server."https://www.digitalocean.com/community/tutorials

Appendix

The following link contains supplementary material for the project, including relevant files and resources:

- Code - GitHub