Automated Image Tagging System

Graduation project report submitted to the
"AI & Data Science - AWS Machine Learning Engineer"
Program by Digital Egypt Pioneers Initiative - DEPI

Supervised By

Eng. Mohamed Bekheet

Prepared By

Abanoub Ayed Khalaf

Mohammed Mahmoud Ali Madany

Abdallah sobhi Abdellaty

Khaled Reda Abdelazim

Mostafa Ahmed Mohamed

Abdallah Labeb Salem

Content

1. **Introduction**
2. **System Design**
3. **System Development**
4. **System Deployment**
5. **Conclusion & Future work**

1. Introduction

Automated image tagging systems are becoming increasingly vital in today's data-driven world, where vast quantities of visual information are produced, stored, and utilized daily across numerous industries. These systems enable organizations to automatically assign descriptive tags to images, making it easier to catalog, search, and retrieve visual content. As the amount of unstructured data continues to grow exponentially, traditional manual tagging methods prove insufficient, time-consuming, and prone to human error. Thus, automated solutions that can accurately and efficiently classify images have become a necessity for businesses and organizations that rely heavily on visual data.

Image tagging, in its simplest form, refers to the process of assigning labels or "tags" to an image, identifying objects, scenes, or concepts within it. These tags make images searchable by keywords or categories, greatly enhancing the organization and usability of large image datasets. Manually tagging hundreds or thousands of images is a labor-intensive task, especially for companies that deal with constant inflows of new media, such as e-commerce platforms, digital content providers, social media networks, and even scientific fields like medical imaging or geology. To overcome these challenges, automated image tagging systems leverage machine learning (ML) and artificial intelligence (AI) technologies to understand and interpret visual content much like a human would.

The growing demand for automated image tagging can be attributed to its wide-ranging benefits. First, it vastly reduces the time and effort required to manage large visual datasets. Instead of dedicating hours or even days to manually label images, companies can rely on sophisticated algorithms that can process thousands of images in a fraction of the time. Additionally, machine learning models designed for image classification and tagging tend to offer higher consistency and accuracy in their results. By minimizing human error, organizations can improve the overall quality and reliability of their image databases. Moreover, automated systems can be trained to recognize an ever-increasing number of categories or specific items within an image, adapting to the unique needs of the business or the industry in which they operate.

Automated image tagging systems are especially beneficial in sectors where vast image collections must be processed regularly. E-commerce platforms, for example, handle immense volumes of product images, which need to be appropriately tagged for users to find them quickly. Accurate and relevant tags improve product discoverability, boosting user engagement and ultimately leading to higher sales conversions. In industries like fashion or automotive, where products are visually driven, proper image tagging allows customers to filter and search for specific items by color, style, brand, or even patterns. Automated systems can tag products with such detail and precision that user experiences are enhanced, leading to increased customer satisfaction and sales growth.

In addition to e-commerce, industries such as media and entertainment, healthcare, and scientific research are also leveraging the power of automated image tagging. Media companies that deal with digital content production and distribution rely on these systems to manage their libraries of photos, videos, and artwork efficiently. Automated tagging can accelerate content delivery processes, making it easier to categorize and retrieve visual media during content production or marketing campaigns. In healthcare, specifically in radiology and diagnostic imaging, automated tagging systems can assist medical professionals by labeling critical patterns in diagnostic scans, helping to detect conditions faster and more accurately. Furthermore, in fields like geology and mineral exploration, automated image classification systems are instrumental in identifying and tagging various mineral types, improving the efficiency of the analysis process.

As machine learning and AI technologies continue to advance, automated image tagging systems have evolved to become more sophisticated and versatile. One significant development in this space is the introduction of Microsoft's **Florence2 model**, a highly advanced model designed for visual understanding. Florence2's ability to interpret complex visual data with high accuracy has set a new benchmark in the industry, making it a critical tool for companies looking to improve their image classification and tagging processes. Built on cutting-edge architectures, Florence2 leverages multi-modal learning, enabling it to understand both visual and contextual elements within an image. This makes it highly effective in generating accurate and descriptive tags for even the most complex images. Moreover, its scalability and compatibility with modern cloud-based infrastructures, like **AWS (Amazon Web Services)**, make it an ideal choice for businesses seeking to automate their visual content management.

This report focuses on the design, development, and deployment of an automated image tagging system built using the Florence2 model and AWS cloud services. The combination of Florence2's robust image classification capabilities and AWS's scalable cloud infrastructure enables the creation of a powerful, efficient, and scalable image tagging solution. The report also explores the system's deployment in two different environments: AWS for large-scale enterprise use and **Streamlit Cloud**, offering an intuitive interface for smaller projects or individual users.

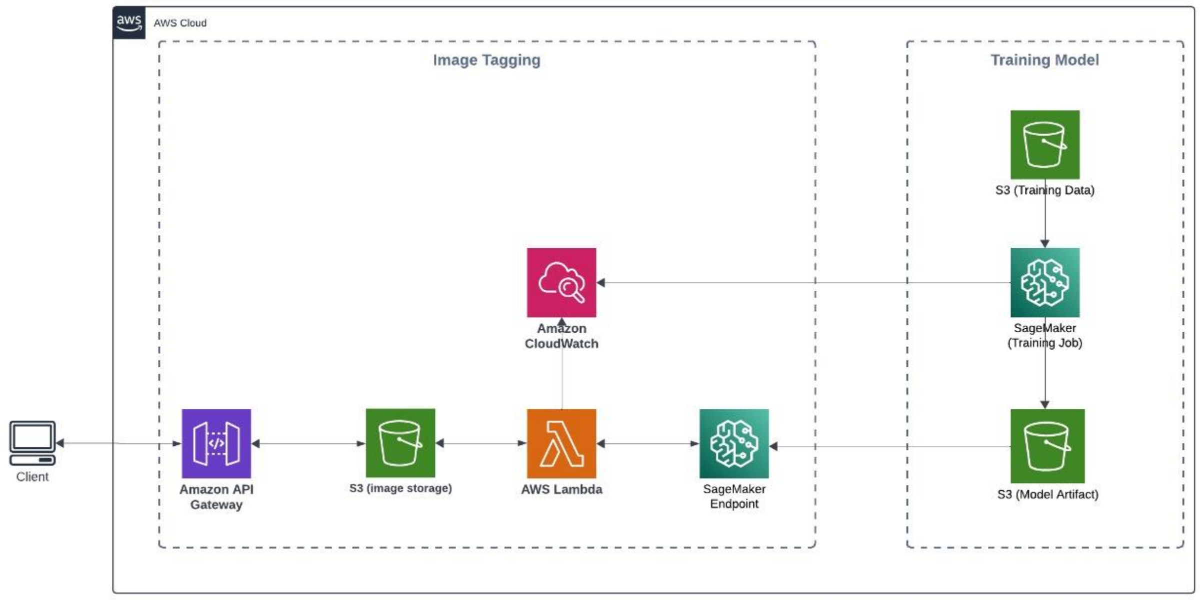
The design of the automated image tagging system aims to maximize performance and flexibility, leveraging a range of AWS services such as **API Gateway**, **S3 storage**, **Lambda functions**, and **SageMaker** for machine learning model hosting and deployment. These services form a cohesive pipeline that facilitates the seamless flow of data from image upload to tag generation. Users can interact with the system by uploading images via API, which are then processed and tagged by the Florence2 model hosted on SageMaker. The entire process is monitored and logged using **Amazon CloudWatch**, ensuring operational transparency and reliability. Furthermore, the system can be extended or modified to suit specific use cases, making it a versatile solution for various industries.

In the following sections, the report will delve into the system's importance and use cases, provide a detailed description of the design, highlight the capabilities of the Florence2 model, and discuss the steps involved in system development and deployment. Finally, the report concludes by summarizing the value of this automated image tagging system and its potential impact across various industries and use cases.

2. System Design

The system design for the automated image tagging solution leverages a combination of AWS services and Microsoft's Florence2 model to create an efficient, scalable architecture. This design ensures seamless interaction between the components responsible for image storage, model inference, and result generation. The diagram provided gives an overview of how these services interact within the system, from receiving user inputs to generating and returning image tags.

The core design comprises two primary parts: **Image Tagging** and **Model Training**. Each of these components is built using different AWS services to perform specific tasks that contribute to the overall functionality of the system.



Components:

1. Client Interface (Image Upload)

The starting point for the system is the **Client Interface**, where users upload images to be tagged. This interface could be anything from a web application or a mobile app, allowing users to upload images via an API endpoint. These images are sent to the system for classification.

- **Amazon API Gateway**: The client interacts with the system through the **Amazon API Gateway**, which acts as the front-end API management service. It handles HTTP(S) requests and forwards them to the appropriate AWS services, ensuring that the image data is securely and efficiently transmitted. API Gateway allows for scalability and real-time processing of multiple concurrent image uploads.

2. S3 (Image Storage)

Once the image is uploaded via the API Gateway, it is stored in **Amazon S3 (Simple Storage Service)**, a highly durable and scalable object storage service. The image file is saved in a designated S3 bucket, which acts as a temporary holding area before further processing.

- **Image Storage in S3**: The S3 bucket for image storage ensures that large quantities of image files can be securely stored and accessed by the system. S3's highly available architecture means that images are always accessible for further processing, regardless of scale.
- **Triggering AWS Lambda**: As soon as a new image is uploaded to the S3 bucket, an event notification is triggered, which invokes an **AWS Lambda** function for further processing.

3. AWS Lambda (Serverless Compute)

Once an image is uploaded and stored, the **AWS Lambda** function is invoked. Lambda is a serverless compute service that runs code in response to events such as S3 uploads. The Lambda function fetches the image from the S3 bucket, prepares it for classification, and then forwards it to the model for inference.

- **Model Invocation**: The Lambda function interacts with the **SageMaker Endpoint** to make the actual inference call to the Florence2 model. This is a key part of the system's design, as Lambda allows for dynamic, on-demand invocation of the model without needing a dedicated server running full-time.

4. Amazon SageMaker Endpoint (Model Hosting)

The core of the image tagging system revolves around **Amazon SageMaker**, AWS's fully managed service that allows developers to build, train, and deploy machine learning models at scale. In this system, SageMaker hosts the **Florence2 model** used for image classification.

- **SageMaker Endpoint**: A **SageMaker Endpoint** is deployed, which serves as the hosting infrastructure for the Florence2 model. This endpoint receives the image from Lambda and processes it using the model to generate tags based on the image's content.
- **Florence2 Model Integration**: Florence2, being a sophisticated model designed for visual recognition tasks, is pre-trained on large-scale datasets. In this system, it has been fine-tuned using specific training data relevant to the use. The endpoint uses this fine-tuned model to provide accurate image tagging results. The tags are then returned to the Lambda function for post-processing

5. Amazon CloudWatch (Monitoring and Logging)

Throughout the system's operation, **Amazon CloudWatch** is used for monitoring and logging. CloudWatch is a monitoring service designed to provide data and insights into AWS services and applications.

- **Logging Lambda and API Gateway Activities**: CloudWatch logs are used to monitor the performance of API Gateway, Lambda functions, and the SageMaker endpoint. It tracks metrics such as the number of requests, response times, and error rates.
- **Performance Monitoring**: CloudWatch helps ensure that the system remains healthy and performs optimally by providing real-time metrics and alarms for potential issues, such as increased latency or processing errors. This data allows for quick troubleshooting and system scaling when necessary.

6. SageMaker Training Job (Model Training Pipeline)

In the development phase, the **training of the Florence2 model** was performed using the SageMaker Training Job. SageMaker allows for scalable training of machine learning models using a wide range of algorithms and frameworks.

- **S3 (Training Data)**: The model is trained using a dataset of labeled images stored in an S3 bucket. This dataset consists of a wide range of images relevant to the domain the model is being trained on (e.g., product images, mineral samples, etc.).
- **Training Job**: SageMaker's Training Job pulls the training data from the S3 bucket and runs the training algorithm to produce a fine-tuned version of the Florence2 model. The training process adjusts the model’s weights and parameters to ensure that it can accurately classify images based on the target dataset.
- **S3 (Model Artifacts)**: Once training is complete, the resulting model artifacts (the trained model) are stored in another S3 bucket. These artifacts are then used to deploy the model to the SageMaker endpoint for inference.

7. S3 (Model Artifacts)

The trained Florence2 model, after completing the training process, is saved in an S3 bucket. This allows for persistent storage of the model, which can be retrieved for future use or further fine-tuning. Storing model artifacts in S3 ensures that the model is accessible for re-deployment or retraining at any time, adding to the system's flexibility.

System Workflow Overview:

1. **Client Uploads Image**: The client uploads an image via the API Gateway.
2. **Image Stored in S3**: The image is stored in an S3 bucket, triggering a Lambda function.
3. **Image Pre-processing**: The Lambda function retrieves and pre-processes the image for inference.

4. **Inference via SageMaker Endpoint**: The processed image is sent to the SageMaker endpoint, where the Florence2 model generates tags based on the image content.
5. **Tags Returned**: The generated tags are returned to the client via Lambda and API Gateway.
6. **Monitoring and Logging**: CloudWatch logs system performance, while metrics are used to track API and Lambda activity.

This detailed system design ensures that the image tagging process is seamless, scalable, and highly efficient, capable of handling large datasets and high volumes of image uploads in real-time. By leveraging AWS services, this system can automatically scale to accommodate more users or higher workloads without requiring additional infrastructure adjustments.

3. System Development

The development of the automated image tagging system centers around integrating advanced machine learning capabilities (via the **Florence2 model**) with AWS's powerful, scalable infrastructure. The primary goal is to develop a system that can automatically process images, generate accurate tags, and seamlessly scale to meet fluctuating demands. The development process consists of multiple stages, including dataset preparation, model training and fine-tuning, infrastructure setup, and application logic implementation. Here's a detailed look into each component:

1. Dataset Preparation

Before the Florence2 model can be used effectively, it must be trained and fine-tuned with relevant datasets. This stage is crucial as the quality and diversity of the training data directly impact the model's ability to generate accurate tags.

- **Data Collection**: The initial step involves collecting a dataset of labeled images relevant to the use case. For example, in a mineral classification system, images of various minerals would be sourced and appropriately labeled (e.g., quartz, feldspar, etc.). In an e-commerce application, product images would be categorized (e.g., electronics, clothing, accessories, etc.).
- **Data Labeling**: Once the dataset is gathered, it is labeled with specific tags that the model will learn to associate with particular visual patterns. Labeling can be done manually by domain experts or through semi-automated systems that leverage pre-trained models to assist in labeling.
- **Data Storage**: The dataset, along with its labels, is stored in an **Amazon S3** bucket. S3 is used due to its scalability and durability, making it the ideal storage solution for large datasets.
- **Data Pre-processing**: The raw images must be pre-processed before feeding them into the model. This includes resizing images to a standard size, normalizing pixel values, augmenting the data (e.g., flipping, rotating, adjusting brightness), and converting images to formats suitable for

training. This step enhances the model's generalization ability, allowing it to perform better in real-world applications.

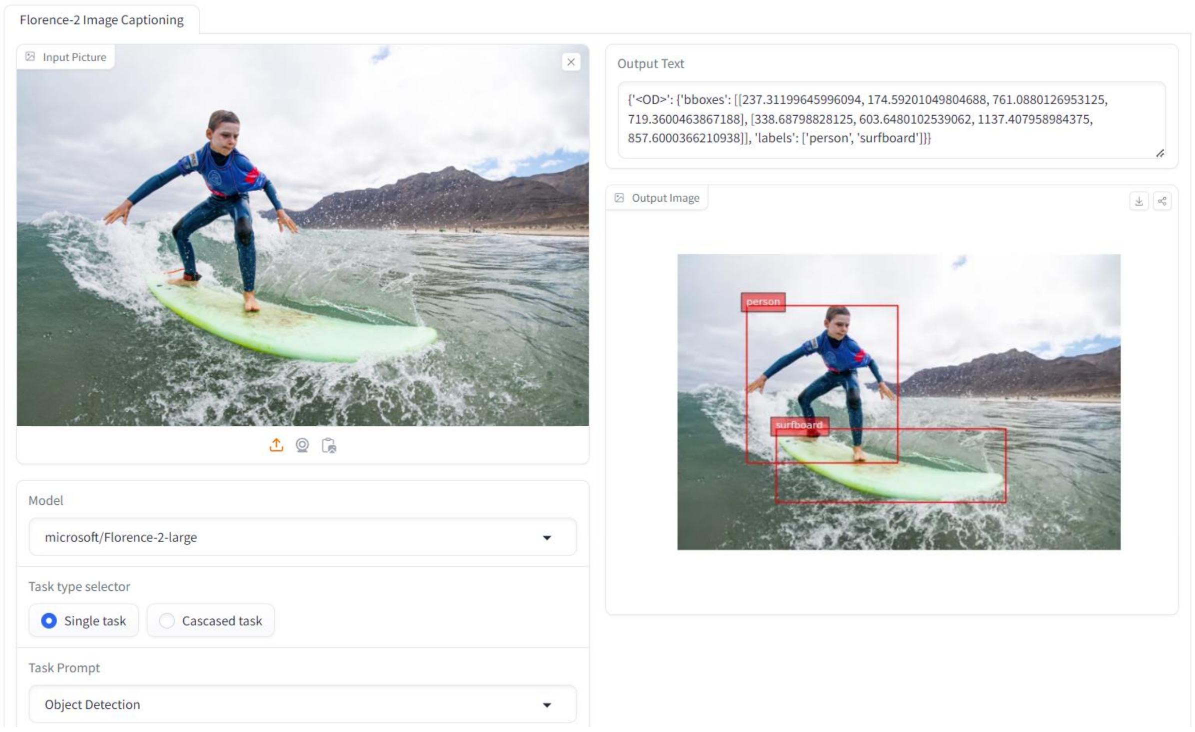
2. Florence2 Model Training and Fine-Tuning

Florence2 Model Overview:

- The intelligence driving the system is powered by Microsoft's Florence2 model, a cutting-edge vision foundation designed for image recognition tasks. Florence2 excels at understanding complex visual data and is versatile enough to handle a wide range of applications. It belongs to a new generation of visual models that utilize multi-modal learning, allowing it to interpret images alongside contextual information, such as text descriptions. This capability enables the model to generate highly accurate and descriptive tags for images, surpassing basic object detection by recognizing complex scenes, relationships, and even abstract concepts. Additionally, Florence2 boasts impressive zero-shot capabilities, allowing it to tag objects it hasn't been explicitly trained on, thanks to its extensive pre-trained knowledge. The model leverages a prompt-based approach to handle diverse vision and vision-language tasks like captioning, object detection, and segmentation. Built on the robust FLD-5B dataset with 5.4 billion annotations across 126 million images, it excels in multi-task learning. Its sequence-to-sequence architecture ensures exceptional performance in both zero-shot and fine-tuned settings, making Florence2 a powerful and competitive model for visual understanding tasks.

Examples using Florence-2 model API

Florence-2 Demo



Output Text

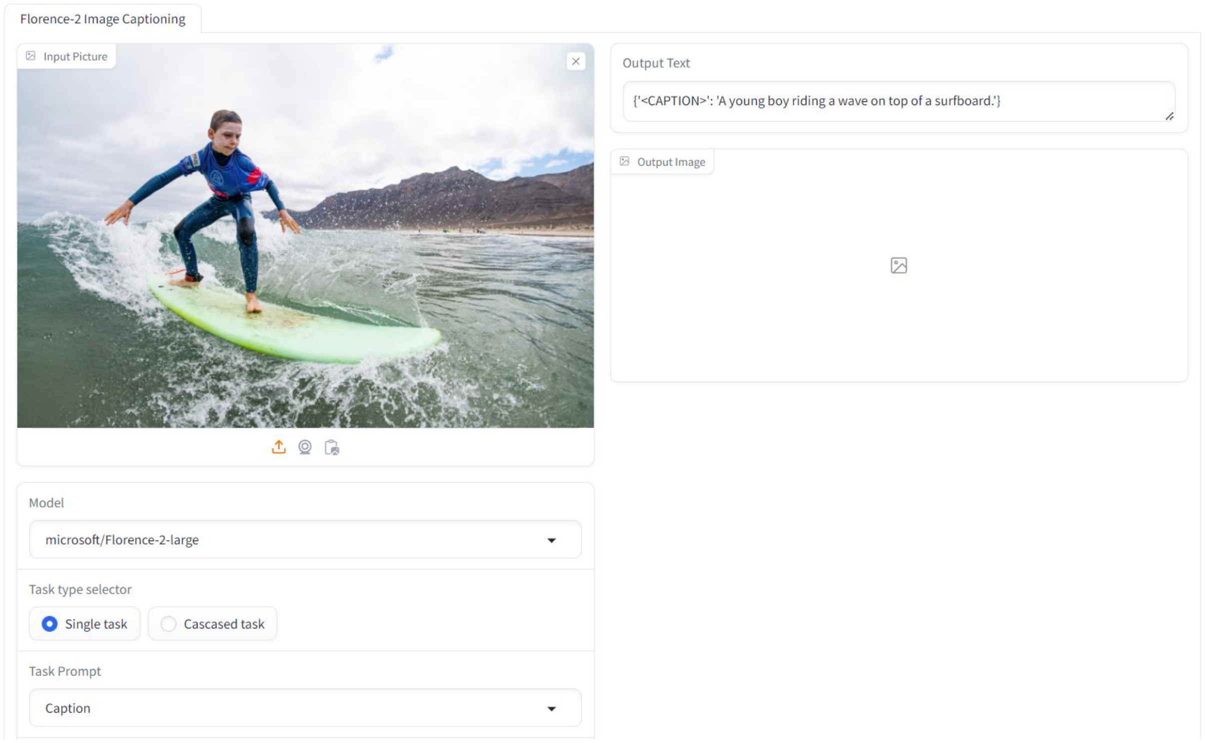
```
{'<OD>': {'bboxes': [[237.31199645996094, 174.59201049804688, 761.0880126953125,
719.3600463867188], [338.68798828125, 603.6480102539062, 1137.407958984375,
857.6000366210938]], 'labels': ['person', 'surfboard']}}
```

Model: microsoft/Florence-2-large

Task type selector: [✓ Single task] Cascased task

Task Prompt: Object Detection

Florence-2 Demo



Output Text

```
{'<CAPTION>': 'A young boy riding a wave on top of a surfboard.'}
```

Model: microsoft/Florence-2-large

Task type selector: [✓ Single task] Cascased task

Task Prompt: Caption

- **Model Fine-Tuning**: Fine-tuning Florence2 involves training the pre-existing model on the specific dataset collected in the earlier step. The **Amazon SageMaker Training Job** is used for this purpose. SageMaker is a fully managed service that simplifies the process of training and tuning machine learning models. The training job reads data from the S3 bucket, performs multiple iterations (epochs) to adjust the model's weights, and stores the final fine-tuned model in another S3 bucket.
- **Hyperparameter Tuning**: Hyperparameters such as learning rate, batch size, and regularization parameters are adjusted during fine-tuning to optimize the model's performance. SageMaker's **automatic model tuning** (also known as hyperparameter optimization) helps find the best combination of hyperparameters to ensure the model is both accurate and efficient.
- **Transfer Learning**: The Florence2 model utilizes **transfer learning**, which allows it to apply the knowledge learned from its original pre-training to the new dataset. By building on top of an already trained model, we reduce the amount of data required and accelerate the training process. This is particularly useful when the dataset for a specific use case is smaller than the datasets typically used for training models from scratch.
- **Model Validation**: Once trained, the model undergoes validation on a separate subset of images to evaluate its accuracy. Validation metrics such as precision, recall, and F1 score are used to assess the model's performance. These metrics help determine if further fine-tuning is required or if the model is ready for deployment.

3. AWS Infrastructure Setup

The system is built on AWS's cloud infrastructure, which provides scalability, flexibility, and security. Setting up this infrastructure is crucial for the system to handle large-scale data, compute-intensive model inference, and seamless deployment. Each component of AWS is configured to work together harmoniously to provide a seamless pipeline from image upload to tag generation.

- **Amazon API Gateway**: The first step is configuring **API Gateway** to expose RESTful APIs that allow users to interact with the system. API Gateway handles incoming HTTP(S) requests and routes them to the appropriate backend service (AWS Lambda). It provides authentication, rate-limiting, and scaling capabilities to handle a large number of concurrent users.
- **Amazon S3**: **S3 buckets** are set up to store both the incoming images and the training data. Access control policies are defined to ensure that only authorized users or services can read or write to the buckets. Additionally, versioning and lifecycle policies can be implemented to manage storage costs and automatically archive or delete old data.
- **AWS Lambda**: Lambda functions are developed using Python or another supported language to handle image pre-processing and to invoke the model for inference. Each Lambda function is triggered automatically when a new image is uploaded to S3 or an API call is made. These functions are lightweight, scalable, and cost-effective as they only run when needed.
- **Amazon SageMaker Endpoint**: The fine-tuned Florence2 model is deployed to a **SageMaker Endpoint**. The endpoint is a hosted environment that serves the model for inference in real-time. SageMaker endpoints are designed to handle production workloads, providing auto-scaling capabilities to meet demand. The endpoint exposes an API that Lambda calls to perform image classification and tagging.
- **Amazon CloudWatch**: **CloudWatch** is configured to monitor the entire system. Logs from API Gateway, Lambda, and SageMaker are collected in real-time, allowing developers to monitor the system's health, diagnose issues, and optimize performance. CloudWatch dashboards provide a centralized view of key performance metrics, such as request latency, memory usage, and error rates.

4. Application Logic and Workflow Implementation

With the infrastructure in place, the next step is implementing the system's core logic, which orchestrates how the different services interact.

- **Image Upload and Pre-processing**: When a user uploads an image via the API Gateway, the image is temporarily stored in S3. The S3 bucket triggers an AWS Lambda function, which fetches the image and performs pre-processing (e.g., resizing, format conversion) to prepare it for inference.
- **Model Inference**: The Lambda function sends the processed image to the **SageMaker Endpoint**, where the Florence2 model generates descriptive tags. The model's output, which includes a list of tags and their corresponding confidence scores, is returned to the Lambda function.
- **Tag Delivery**: The Lambda function formats the tags and sends the response back to the user via the API Gateway. The tags are either displayed on a web interface or sent to another system for further processing (e.g., updating a database, sending alerts, etc.).

5. Error Handling and Logging

Error handling is a critical part of system development. Mechanisms are built into the system to detect and handle errors gracefully. For instance, if an image is corrupt or an API request is malformed, the Lambda function is designed to return appropriate error messages to the user. **CloudWatch Alarms** can also be configured to send alerts in case of system failures, such as high latency in the SageMaker endpoint or API Gateway errors.

6. System Security

Security is paramount when developing cloud-based systems. Various security measures are implemented, including:

- **Authentication and Authorization**: **AWS Identity and Access Management (IAM)** roles and policies ensure that only authorized users or services can access specific resources (e.g., S3, Lambda, SageMaker).
- **Data Encryption**: S3 buckets are configured with server-side encryption to ensure that images and model artifacts are securely stored.

4. System Deployment

The deployment of the system can be divided into two parts: **AWS-based deployment** and **Streamlit-based deployment**.

AWS-based Deployment:

- **Amazon API Gateway** serves as the interface for users to upload images and request tags.
- **S3 (Image Storage)** is used to hold the image files before they are processed by the system.
- **AWS Lambda** fetches the images from S3 and invokes the SageMaker endpoint that runs the Florence2 model.
- The **SageMaker Endpoint** is where the Florence2 model resides, performing inference to tag the images.
- **CloudWatch** provides logging and monitoring, helping track the performance of the system and spot any bottlenecks.

This architecture is highly scalable, leveraging serverless components to reduce overhead and allow the system to handle fluctuations in traffic seamlessly.

Streamlit Deployment:

For a more user-friendly and interactive deployment of the automated image tagging system, it can be integrated into a **Streamlit Cloud app**. Streamlit is an open-source Python framework that allows developers to quickly build web applications for machine learning and data science projects. Its simplicity and ease of use make it an ideal choice for creating custom UIs that display real-time data, such as image classification results.

Streamlit facilitates the deployment of machine learning models without the need for deep knowledge of front-end development, as it allows users to create interactive web applications with just a few lines of Python code. The system leverages Streamlit's ability to integrate with back-end services (such as AWS) and display outputs in real time. Below is a breakdown of how the deployment process works in the context of this image tagging system:

Streamlit Overview

Streamlit provides a simple interface where developers can design web-based applications using a Python script. The script handles both the back-end and front-end logic, which means the developer doesn't need to worry about writing HTML, CSS, or JavaScript. The primary features that make Streamlit ideal for this system include:

- **Widgets**: Streamlit offers easy-to-use widgets such as file uploaders, sliders, and buttons, enabling user interaction with minimal code.
- **Live Updates**: Streamlit's real-time updates make it possible to dynamically display content, such as images and their corresponding tags, immediately after the user uploads an image.
- **Data Visualization**: It offers built-in support for displaying text, images, and other data types, making it an excellent platform for showcasing the results of the image classification model.

How the Streamlit App Works in This System

The Streamlit UI serves as the front end where users can interact with the system by uploading images. Behind the scenes, Streamlit communicates with the AWS-based back-end services to execute the image tagging process. Here's a detailed flow of how it works:

1. **User Interface (UI) for Image Upload**: The core of the Streamlit application starts with a widget that allows users to upload images from their local machines. This can be implemented using Streamlit's st.file_uploader() widget, which provides a user-friendly interface for selecting and uploading image files.
2. **Backend Processing and Communication with AWS Services**: Once the image is uploaded, the Streamlit app initiates back-end processing. The app uses AWS SDKs (e.g., **Boto3**) to communicate with various AWS services that form part of the system architecture.

   - **Image Storage in S3**: The uploaded image is first sent to an **Amazon S3** bucket, where it is stored temporarily for further processing.

    - **Triggering AWS Lambda**: After the image is successfully uploaded to S3, **AWS Lambda** is invoked to handle the pre-processing of the image (such as resizing and format conversion). The Lambda function then sends the image to the SageMaker endpoint for inference.
    - **Model Inference via SageMaker**: The Florence2 model, hosted on the **Amazon SageMaker Endpoint**, processes the image and generates classification tags. These tags, along with their confidence scores, are returned to the Streamlit app through the Lambda function.

3. **Real-Time Display of Image Tags**: The generated tags are displayed in the Streamlit app, allowing users to see the results almost instantly. The display can be implemented using Streamlit's st.write() or st.text() functions, which support dynamic updates. Additionally, the image itself can be rendered using st.image(), giving users both the visual and textual output simultaneously.

3. Benefits of Streamlit Integration

Integrating Streamlit into the automated image tagging system offers several advantages:

- **User-Friendly Interface**: The system becomes accessible even to non-technical users, as it requires no knowledge of machine learning or AWS. Users can simply upload an image and receive a set of tags through a clean, intuitive interface.
- **Interactive Experience**: Users can interact with the system in real time, adjusting inputs (i.e., uploading new images) and receiving immediate feedback in the form of image tags. This makes the system suitable for demos, presentations, and small-scale use cases where user interaction is essential.
- **Rapid Prototyping**: Streamlit allows rapid iteration and experimentation. Developers can tweak the underlying model or the image pre-processing steps, and users can test the system immediately without needing to redeploy the app.

- **Low-Code Deployment**: The deployment process in Streamlit is highly streamlined. By running a simple Python script, the developer can push the app to **Streamlit Cloud**, where it becomes accessible via a unique URL, allowing users to interact with the system directly from their web browsers.
- **Cost-Effective**: For small-scale applications or initial testing, Streamlit Cloud provides free-tier services, reducing the deployment cost significantly while maintaining scalability for later expansion.

4. Step-by-Step Setup of Streamlit for the Image Tagging System

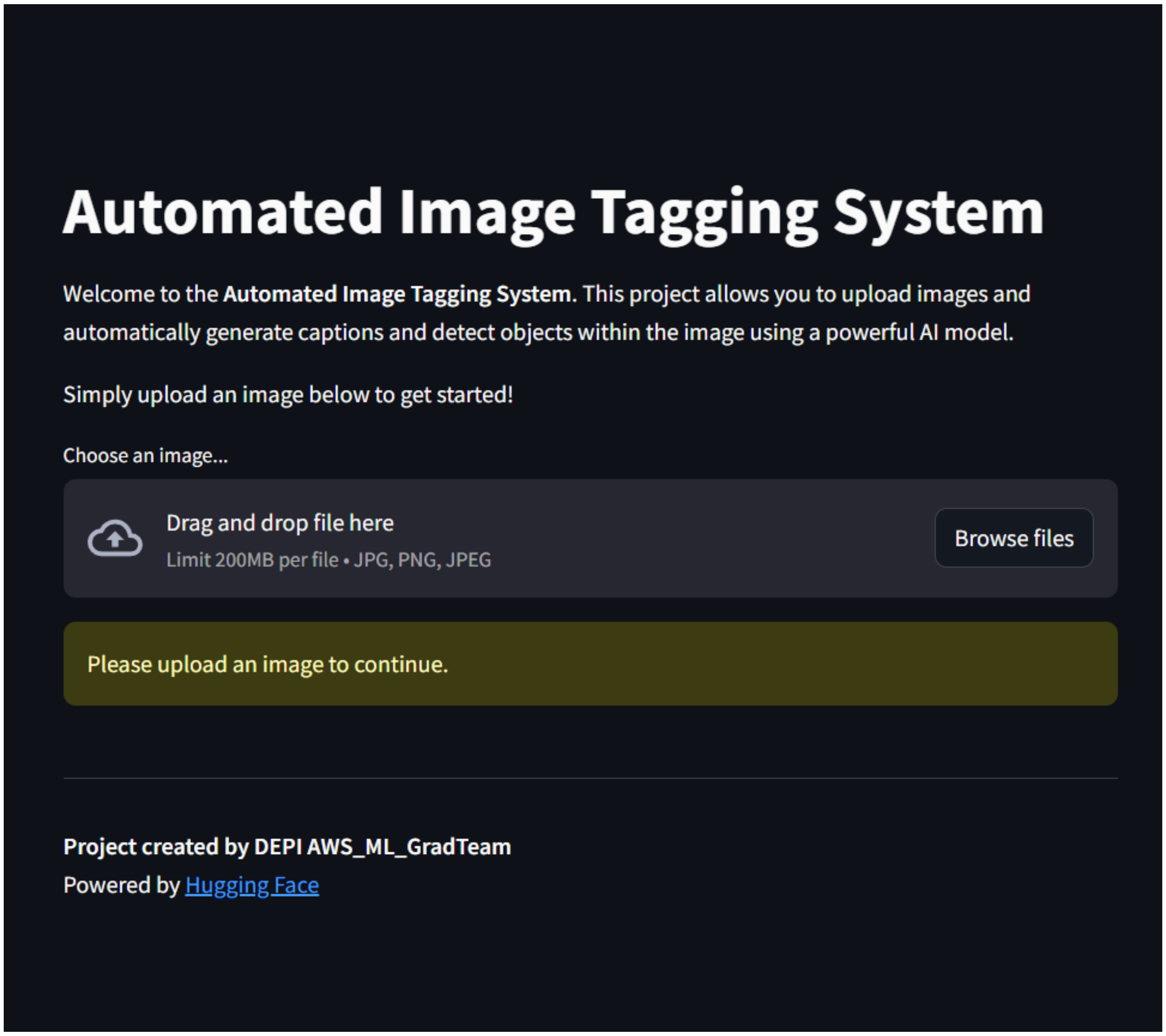
Below is a high-level guide to setting up the Streamlit deployment for the automated image tagging system:

1. **Install Streamlit**: Install Streamlit via pip with the command pip install streamlit. This ensures that the necessary dependencies are set up in your local environment.
2. **Create the Streamlit App Script**: Write a Python script that includes the following:
   - A file uploader widget (st.file_uploader()) for users to upload images.
   - Code to handle file processing and communication with AWS S3 and Lambda via the Boto3 library.
   - Code to display the image and the model-generated tags in real time.
3. **Connect to AWS**: Using the **Boto3 SDK**, set up connections to the AWS services (S3, Lambda, SageMaker). Ensure the app can send uploaded images to S3, trigger Lambda functions, and retrieve the tags from SageMaker.
4. **Deploy to Streamlit Cloud**: Once the app is functional, deploy it to **Streamlit Cloud**. This can be done directly from the command line using streamlit deploy or by pushing the code to a GitHub repository and linking it to Streamlit Cloud. The app will be hosted on Streamlit's servers and assigned a unique URL that users can access.

5. Key Considerations for Streamlit Deployment

- **Scalability**: While Streamlit works well for small-scale applications, it is not designed for high-traffic production environments. If the system needs to scale to handle a large number of concurrent users, additional considerations, such as deploying the app on more robust cloud infrastructure (e.g., AWS EC2 or Kubernetes), might be necessary.
- **Authentication**: Streamlit Cloud provides basic authentication, but for more advanced use cases, integrating with AWS Cognito or another authentication provider may be required to restrict access to the system.

Pictures of the streamlit app interface with an example



Automated Image Tagging System

Welcome to the **Automated Image Tagging System**. This project allows you to upload images and automatically generate captions and detect objects within the image using a powerful AI model.

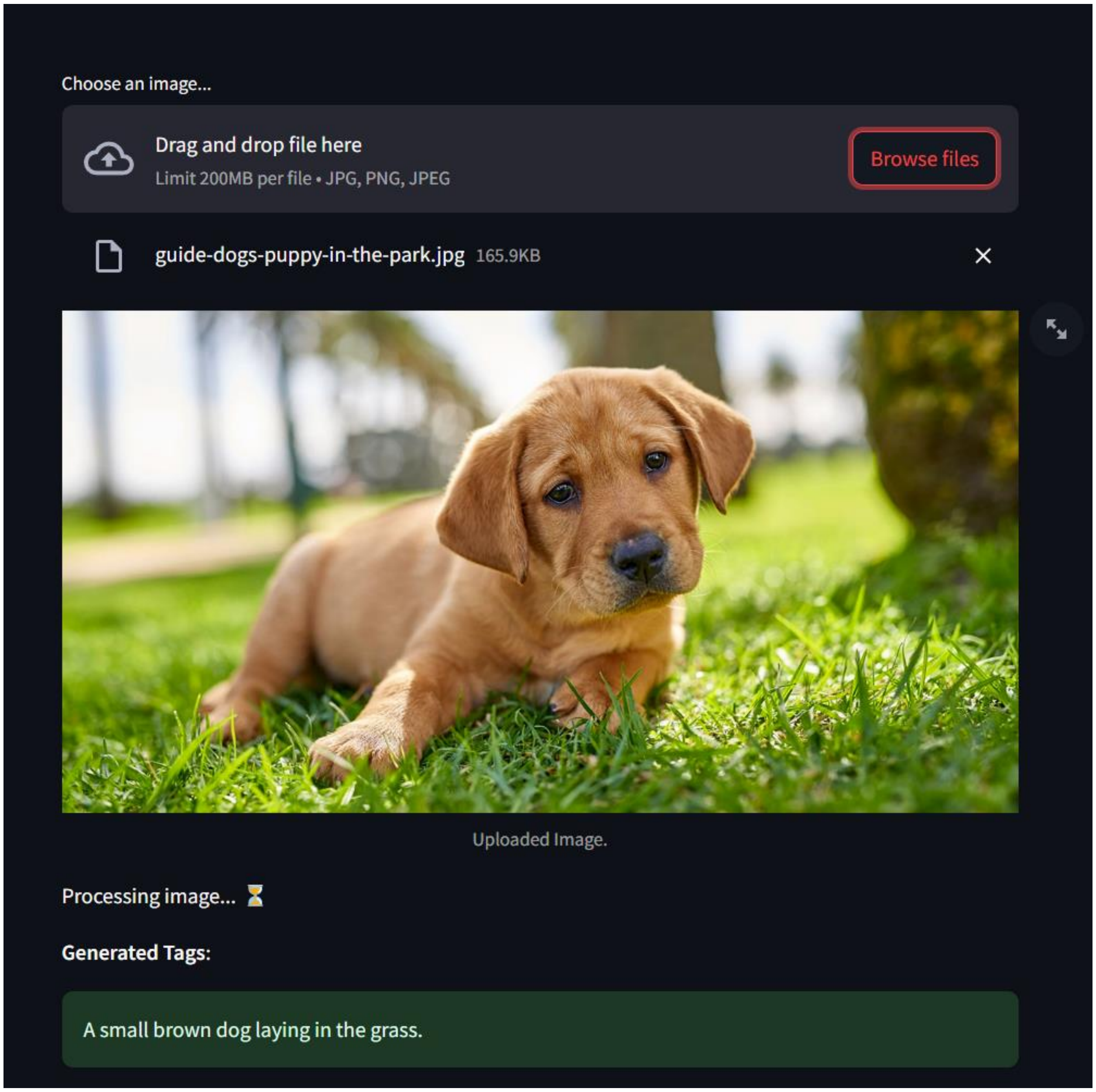
Simply upload an image below to get started!

Choose an image...

Please upload an image to continue.

Project created by DEPI AWS_ML_GradTeam
Powered by Hugging Face

performing tagging


Choose an image...

Drag and drop file here

Limit 200MB per file • JPG, PNG, JPEG

guide-dogs-puppy-in-the-park.jpg 165.9KB

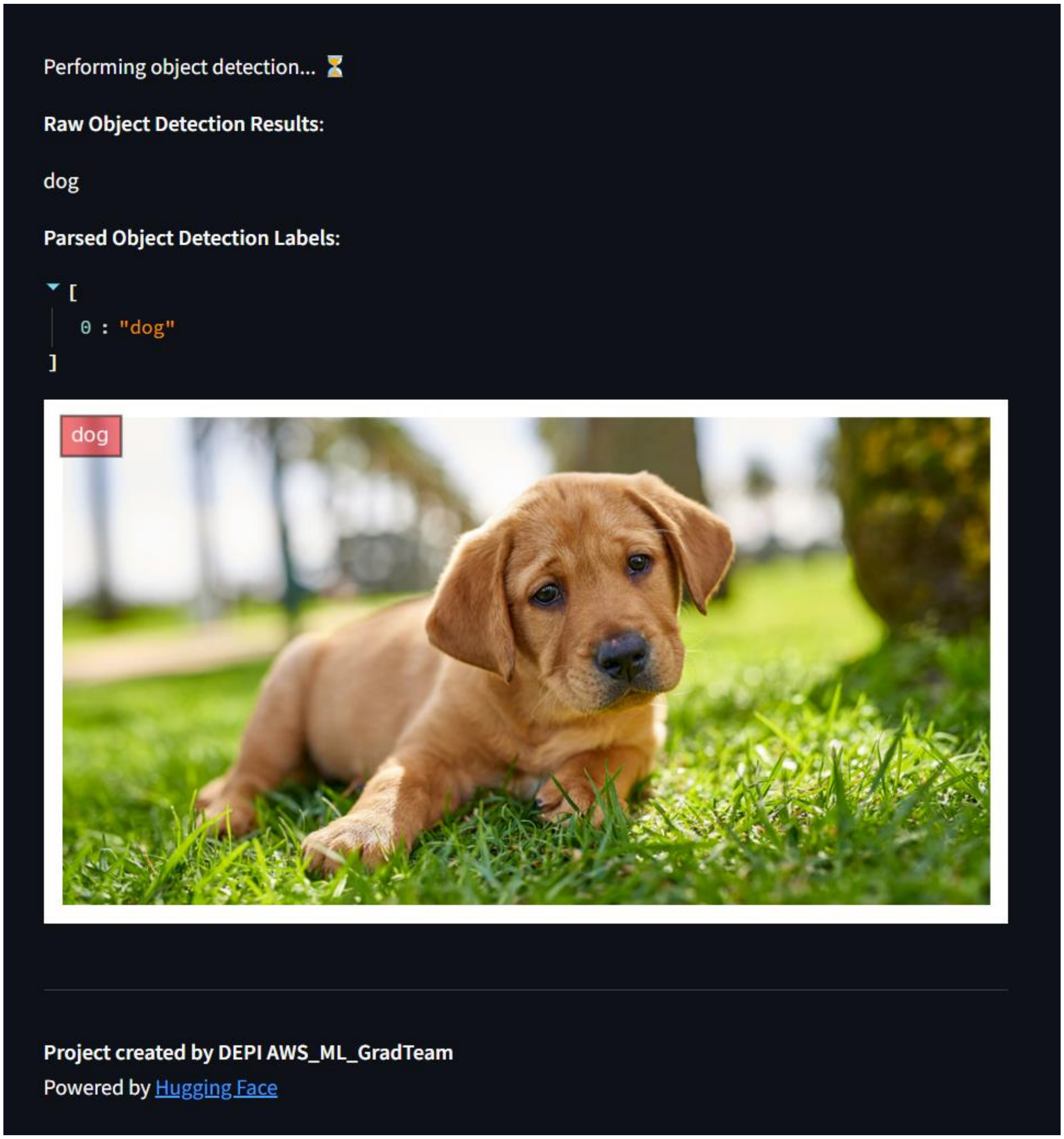
Uploaded Image.

Processing image... ⌛

Generated Tags:

A small brown dog laying in the grass.

performing object detection



Performing object detection... ⏳

Raw Object Detection Results:

dog

Parsed Object Detection Labels:

```
[
  0 : "dog"
]
```

Project created by DEPI AWS_ML_GradTeam

Powered by Hugging Face

5. Conclusion & Future work

The automated image tagging system using the **Microsoft Florence2 model** combined with **AWS cloud infrastructure** provides a scalable and efficient solution for various image-heavy industries. By automating the tagging process, the system eliminates the need for manual labeling, improves accuracy, and allows organizations to focus on leveraging insights from their visual data. The deployment strategies, both in AWS and using Streamlit, ensure that the system can handle a wide range of use cases, from large-scale enterprise solutions to easy-to-use interfaces for smaller projects. This versatility makes it an invaluable tool for anyone working with large image datasets.

The future work for the automated image tagging system aims to expand its capabilities to **mineral identification** by gathering a comprehensive dataset of mineral images. This dataset would include diverse, high-quality images of various minerals, labeled with key identifying information. The Florence2 model would be fine-tuned through transfer learning to specialize in classifying minerals based on visual characteristics. The system would benefit industries like mining, education, and research, allowing users to quickly identify minerals in real-time via a **Streamlit** interface. Challenges such as data diversity, model accuracy, and scalability would need to be addressed for successful implementation.