SOIL: A Privacy-First Framework for Secure On-Device Inference of Leaf Diseases on Mobile

Kiran K. Kethineni*, Saraju P. Mohanty* and Elias Kougianos†
*Department of Computer Science and Engineering, University of North Texas, USA
†Department of Electrical Engineering, University of North Texas, USA
Email: kirankumar.kethineni@unt.edu, saraju.mohanty@unt.edu, elias.kougianos@unt.edu

***Abstract*—Mobile-based plant disease diagnosis applications typically require uploading an image of the diseased plant to a remote server that hosts the inference model and necessary computational resources. This raises significant concerns regarding data privacy and security. Alternatively, downloading and executing external models directly on mobile devices often necessitates additional library support, leading to cross-compatibility issues. To address these challenges, this paper presents SOIL: a privacy-first framework for Secure On-Device Inference of Leaf Diseases on mobile devices. The proposed framework consists of a web-based interface that dynamically fetches optimized CNN models corresponding to the crop selected by the user. These models are executed directly in the mobile browser using TensorFlow.js (TF.js), which leverages the native JavaScript engine to perform on-device inference. This ensures that all image data remains local to the device, thereby preserving user privacy. Since this approach does not require any additional software installations, it can be deployed across a wide range of mobile phones and tablets, regardless of platform. The SOIL framework was validated on multiple smartphones and demonstrated consistent inference capabilities across devices, highlighting its potential for scalable, privacy-aware agricultural diagnostics in real-world field conditions.**

***Keywords*—Smart Agriculture; Artificial Intelligence; Data Privacy; Data Security; Cross-Device Compatibility; TensorFlow.js (TF.js); TinyML.**

I. Introduction

Agriculture has remained the primary source of nutrition for humans, driving numerous technological advancements aimed at improving yield and sustainability. This evolution has led to the emergence of Smart Agriculture [1]. Subsequently, numerous Artificial Intelligence (AI) driven methods have been introduced to detect and identify disease infestations in crops by analyzing images of leaves or plants [2]. Running these models typically requires support for the specific libraries used during their development. As a result, they are often deployed on edge devices such as Raspberry Pi, Jetson Nano, or OpenMV. However, expecting farmers to invest in dedicated hardware solely to identify plant diseases is impractical, especially when many already possess smartphones.

As a result, mobile applications [3] have been developed that allow users to upload plant images to a centralized server, where the Machine Learning (ML) model resides along with the necessary computational resources to perform inference. This centralized approach raises concerns regarding privacy and security, as the data can be tampered with or eavesdropped on during transmission over the internet [4] as shown in Fig.1. In addition, users may be reluctant to install applications on their devices that are not used on a daily basis.

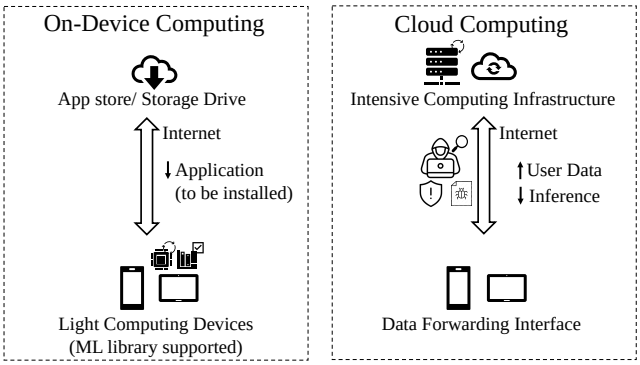

Figure 1: Traditional AI application architecture.

To address the challenges of data privacy and device cross-compatibility, this paper proposes SOIL: a framework for Secure On-Device Inference of Leaf Diseases. SOIL leverages libraries that are inherently available on most mobile phones and tablets to execute lightweight convolutional neural networks (CNNs) locally on the device, ensuring that the data never leaves the user's control.

The remainder of this paper is structured as follows. Section II outlines the proposed framework along with its key contributions. Section III reviews relevant literature. Section IV describes the methodology in detail, and Section V presents the experimental setup and results. Finally, Section VI concludes the paper and highlights possible directions for future work.

II. Novel Contributions of the Current Paper

A. Proposed Solution of the Current Paper

Most mobile phones, regardless of the operating system, include a web browser capable of executing client-side functionalities such as form validation and basic data checks (e.g., date verification). These tasks are typically implemented using JavaScript, enabling the code to run directly on the client device and offload computation from the server [5]. To enable machine learning execution within browsers, TensorFlow introduced TensorFlow.js (TF.js) [6] a framework that allows conversion of standard machine learning models into JavaScript format, enabling them to run on a wide range of

devices directly in the browser, thereby supporting the privacy-first design goals of the proposed SOIL framework. TF.js executes client-side using the device's GPU (WebGL) or CPU (WASM/JS) to run the converted model artifacts [7].

Although TF.js enables local execution of machine learning models, performance remains constrained by the limited resources such as RAM and processor capabilities available on mobile devices. To address this, the proposed SOIL framework incorporates Chroma-Sense [8], an optimization technique designed to reduce the computational overhead required for plant disease classification on edge devices. In addition, to further reduce model size and computational load, we propose allowing the user to select the type of crop. Based on this selection, a corresponding crop-specific model is dynamically loaded into the browser, and inference is performed on the uploaded or captured image.

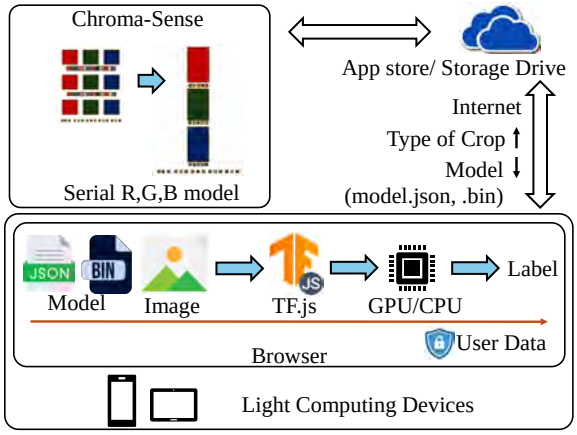

Figure 2: Overview of proposed SOIL.

B. Novelty and Significance of the Proposed Solution

The novel contributions of the proposed SOIL framework are as follows.

1) Privacy-First On-Device Execution: SOIL performs all inference locally, ensuring that image data never leaves the device and eliminating risks associated with data transmission to remote servers.
2) Cross-Platform Compatibility Without Installation: By leveraging native browser capabilities, SOIL operates seamlessly across diverse mobile operating systems without requiring additional software installations.
3) Dynamic Crop-Specific Model Loading: SOIL optimizes resource utilization by dynamically fetching lightweight CNN models tailored to the user's selected crop, minimizing memory footprint and improving inference efficiency.
4) Integration of Edge-Optimized Architectures: SOIL integrates pre-optimized CNN models developed using the Chroma-Sense [8] architecture into its framework, enabling efficient real-time inference even on low-specification mobile devices.

III. Related Prior Works

The article [9] presented an edge-optimized model for plant disease detection using MobileNetV3 in ONNX format, ensuring cross-platform compatibility. Authors of [10] proposed the use of parallel depthwise separable convolution and regression to develop lightweight models suitable for edge devices such as Jetson Nano. Similarly, models were developed using SqueezeNet and EfficientDet-Lite0, optimized for edge devices like Raspberry Pi 4, with the expectation that they can also perform efficiently on mobile phones given their resource requirements in [11]. The authors of [12] developed a mobile application that hosts the ML model on the cloud, performs inference remotely, and delivers results to the device via the internet. [13] presented an end-to-end framework for detecting and localizing diseased regions in plant images using explainable AI. However, this approach is designed for edge devices and aerial field surveillance rather than mobile devices. A fully local mobile application capable of performing inference directly on the device was proposed in [14], though it requires users to download and install the application. Similarly, an edge-optimized mobile app was developed by the authors of [15]. [16] evaluated lightweight CNN models enhanced with spatial attention on constrained IoT edge devices, demonstrating their efficiency and applicability for mobile platforms.

While the works summarized in Table X present models suitable for edge or mobile deployment, this paper introduces SOIL, a framework for the distribution and execution of ML models on mobile devices, explicitly addressing data security and cross-compatibility challenges.

Table I: Relevant literature on plant disease detection.

Work	Method Used	CDC	Privacy
Khan et al. [9]	Edge-optimized Model	No	Yes
Ahamed et al. [10]	Edge-optimized Model	No	Yes
Akuthota et al. [11]	Edge-optimized Model	No	Yes
Chaitra et al. [12]	Cloud-based inference	Yes	No
Karim et al. [13]	Edge-optimized models	No	Yes
Iftikhar et al. [14]	On-device app	No	Yes
Foysal et al. [15]	On-device app	No	Yes
Zeeshan et al. [16]	IoT-optimized Model	No	Yes
SOIL	**Browser-based on-device**	**Yes**	**Yes**

Notes: "CDC" indicates Cross-Device Compatibility of the solution. "Privacy" indicates whether image data remains local to the device.

IV. Proposed Method

A. Chroma-Sense Based Model Development

The proposed work adopts the Chroma-Sense architecture [8] for the classification of plant leaf diseases. Chroma-Sense reduces the model size and memory footprint by exploiting the inherent simplicity of disease-related patterns such as rings, circles, dots, and patches that occur across individual color channels. Instead of jointly processing full RGB inputs, each channel is handled independently using the same feature extractor in a sequential manner, referred to as Serial

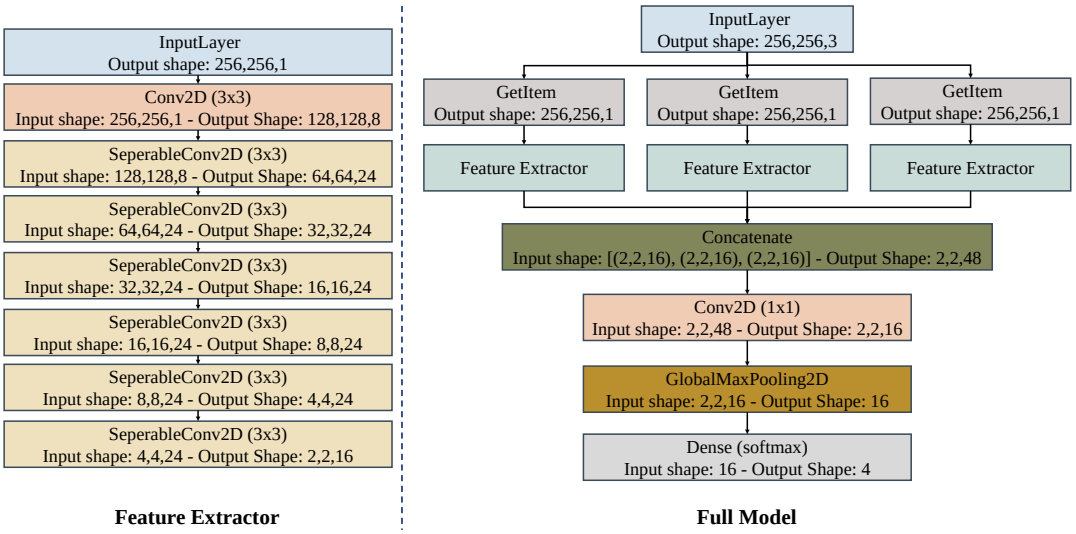

Figure 3: Overview of the proposed classification model.

Multi-Channel Processing. This reuse of the same network substantially reduces the number of learnable parameters and the overall memory footprint, while at the same time forcing the model to focus on learning texture- and shape-specific features that are highly characteristic of plant diseases.

Once the three channels are independently processed, the corresponding feature maps are concatenated to form a unified representation. To further enhance discriminative power, a point-wise convolution layer is applied over the concatenated maps, enabling the network to assign higher weights to the most informative features among three channels. This step not only restores the inter-channel correlation that would otherwise be lost but also embeds spatial and semantic relationships across the channels. As a result, the architecture achieves a favorable trade-off between efficiency and representational strength, making it particularly suited for low-resource devices where memory and compute capacity are constrained. The resulting model can generalize across different crops with relatively lightweight fine-tuning, while maintaining the ability to capture subtle disease semantics. The structure of a representative model trained for a crop with four disease classes is illustrated in Fig. 3, highlighting the compact yet effective design of the Chroma-Sense architecture.

B. Model Conversion

Following training, the TensorFlow (TF) models were further transformed into TensorFlow.js (TF.js) compatible artifacts for deployment within the browser. The TF.js converter generates two types of files: (1) a `model.json` file that encodes the model topology, input/output tensor specifications, and relevant metadata, and (2) a set of binary weight shard files in `.bin` format that store the model parameters in a compressed, partitioned manner. Sharding enables large weight matrices to be broken down into smaller chunks, reducing memory overhead during file transfer and enabling progressive loading in the browser.

These artifacts are lightweight compared to the original TensorFlow SavedModel representation, making them more suitable for constrained environments such as mobile browsers. Since the weight shards can be cached locally by the browser once downloaded, subsequent inferences can be performed without repeated network transfers, improving responsiveness. The generated files are hosted on a standard web server as represented in Fig. 4 for distribution to client devices.

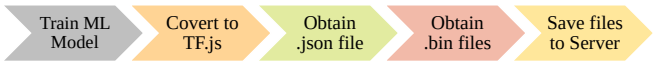

Figure 4: Training and conversion pipeline.

C. Browser-Based Execution with TensorFlow.js

The core novelty of the proposed SOIL framework is its ability to execute plant disease classification directly within the mobile browser using TensorFlow.js (TF.js). TF.js is an open-source library for client-side machine learning that supports multiple execution backends, enabling flexible adaptation to device capabilities. In this work, models are deployed in standard TF.js format and, by default, execute on the WebGL backend, which maps tensor operations to the mobile GPU for accelerated inference. Leveraging the parallelism of GPU shaders, WebGL enables near real-time inference for high-end and mid-range smartphones, even when processing input images of size 256×256.

When GPU access is limited, disabled, or unavailable, TF.js can seamlessly switch to the WebAssembly (WASM) backend, which compiles computational kernels into a low-level bytecode that executes efficiently across heterogeneous CPU architectures. WASM ensures deterministic performance

across devices that lack hardware acceleration, making it especially relevant for budget smartphones and tablets. If neither WebGL nor WASM support is present, TF.js automatically falls back to the pure JavaScript (CPU) backend. Although this configuration results in higher latency, the tiered backend selection guarantees that inference remains functional under all circumstances, thereby improving robustness and cross-device reliability.

An important advantage of this design is that it requires no additional installations, native applications, or device-specific libraries. All modern mobile browsers support JavaScript execution and WASM, enabling seamless deployment on both Android and iOS operating systems without the need for separate packaging or platform-specific maintenance. This cross-platform capability substantially reduces adoption barriers for end-users (farmers) who may operate a diverse set of devices. Moreover, browser-level caching mechanisms allow once-fetched models to be reused across sessions, minimizing bandwidth usage in rural areas with intermittent or low-speed internet. By embedding execution within the browser itself, SOIL combines portability, privacy preservation, and scalability, making it a practical solution for large-scale agricultural diagnostics.

D. Inference Workflow

When a user accesses the system, they connect to the server through their mobile browser and interact with a web-based interface that serves as the starting point inference workflow of the SOIL framework depicted in Fig. 5. After selecting the crop of interest from the user interface, the corresponding pre-trained model files (`model.json` and associated weight shards in `.bin` format) are dynamically fetched from the server. To improve efficiency, these files are cached locally on the device, ensuring that subsequent inferences can be performed without repeated downloads. Once the model has been loaded, TensorFlow.js (TF.js) is initialized, and the framework automatically selects the most appropriate backend available on the device (WebGL, WebAssembly, or pure JavaScript) for execution.

Following model initialization, the user can either upload an image from the device gallery or capture a new image of the diseased leaf directly within the browser. All preprocessing operations are performed entirely on the client side. Specifically, the captured image is resized to match the input resolution expected by the model (e.g., 256×256), normalized to the appropriate pixel value range, and then converted into a tensor representation using browser-native APIs such as `tf.browser.fromPixels`. This preprocessing pipeline ensures consistency with the conditions under which the models were trained, while maintaining low computational overhead.

Once preprocessed, the tensor is passed to the TF.js runtime, where inference is executed locally on the device. The backend compiles and executes the required computational kernels, producing the probability distribution across disease classes. The top prediction, along with the associated confidence score, is immediately presented to the user in the browser interface. Because all computations occur entirely within the browser, the framework guarantees that no image data leaves the device, thereby upholding the privacy-first design philosophy of SOIL.

To ensure efficient resource utilization, all intermediate tensors are carefully managed using TF.js memory management APIs. Temporary tensors created during preprocessing and inference are explicitly disposed after each run, reducing memory footprint and preventing memory leaks that could degrade performance on devices with limited RAM. This workflow not only preserves privacy but also maintains responsiveness, providing farmers with timely and accurate disease predictions without the need for specialized hardware or sending image data over internet.

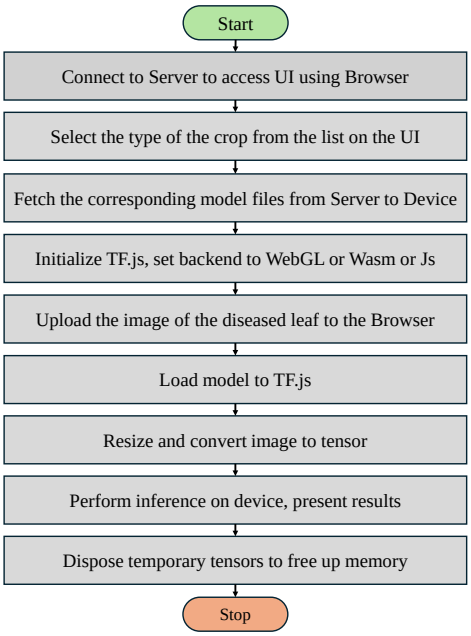

Figure 5: Overview of the proposed inference workflow.

V. Experimental Verification

The proposed classification models for serial RGB processing of plant leaf diseases were implemented using Python and TensorFlow. Training and validation were performed on a curated subset of the PlantVillage dataset [17]. The subset used in this study contained only Apple, Tomato, Grape, and Corn crops covering 18 distinct disease classes with approximately 18,000 images. Each image was resized to 256×256 pixels to match the input requirements of the Chroma-Sense based CNN models. The dataset was randomly split into an 80:20 ratio for training and validation, respectively. Care was taken to ensure that disease-specific semantics, such as rings, dots, or patches, were preserved across the splits, to encourage the model to learn robust discriminative features rather than dataset artifacts.The trained TensorFlow models were subsequently converted into TensorFlow.js (TF.js) compatible `model.json` and `.bin` files. For the experiments, these TF.js artifacts were

hosted on a local computer and distributed to clients via ngrok endpoints.

To evaluate interpretability, representative Grad-CAM [18] visualizations for models trained with 256×256 inputs are shown in Figs. 6–9. These visualizations confirm that the models attend to relevant features, such as lesions, necrotic spots, or rust patches, rather than background artifacts. In terms of predictive accuracy, the validation results demonstrated strong performance: 94% for Apple, 91% for Tomato, 96% for Grape, and 95% for Corn, as summarized in Table II. These results highlight the generalizability of the Chroma-Sense based models across multiple crop types, while maintaining computational efficiency suitable for real-world browser-based deployment.

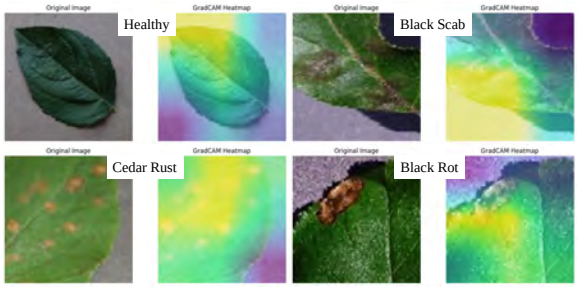

Figure 6: Grad-CAM results for the Apple dataset.

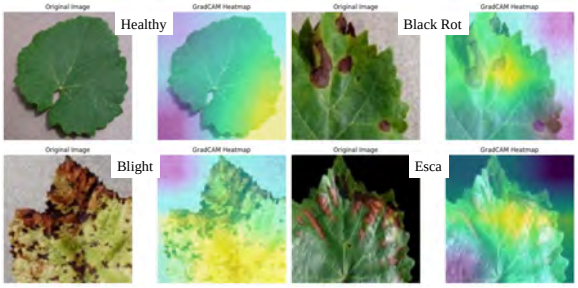

Figure 7: Grad-CAM results for the Grape dataset.

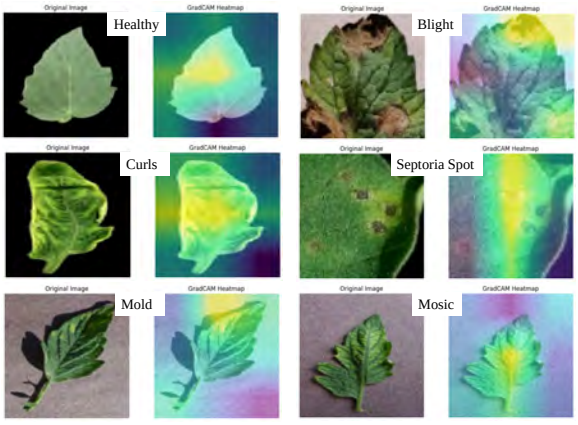

Figure 8: Grad-CAM results for the Tomato dataset.

To further assess the deployment feasibility of the proposed SOIL framework, inference latency was measured across a range of devices with heterogeneous hardware configurations.

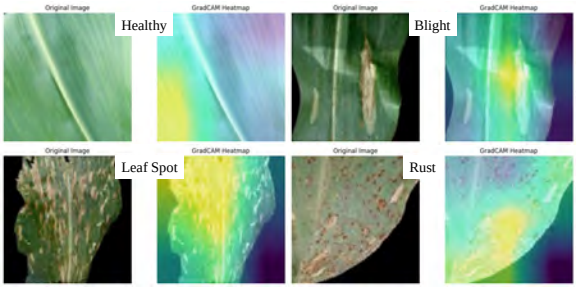

Figure 9: Grad-CAM results for the Corn dataset.

Table II: F1 Scores for different crop types.

Apple	F1 Score	Tomato	F1 Score
Healthy	0.95	Healthy	0.92
Black Rot	0.95	Mold	0.90
Black Scab	0.94	Curls	0.91
Cedar Rust	0.92	Blight	0.90
		Mosaic	0.93
		Septoria Spot	0.91
Grape	**F1 Score**	**Corn**	**F1 Score**
Healthy	0.98	Healthy	0.96
Black Rot	0.96	Rust	0.94
Esca	0.96	Blight	0.94
Blight	0.95	Leaf Spots	0.96

On a Windows laptop equipped with an NVIDIA GPU, the average inference time per image was 13.5 ms, reflecting the benefit of GPU acceleration through the TF.js WebGL backend. On an Apple iPhone 14 Pro Max, inference required approximately 16 ms, demonstrating that modern flagship mobile devices can deliver near real-time diagnostic performance directly in the browser. For mid-range and entry-level devices, the observed latencies were higher: 94 ms on a Samsung A16 and 400 ms on an LG Stylo 4. These results highlight the scalability of the framework across different hardware tiers: while high-end and mid-tier smartphones comfortably support interactive and timely diagnosis, even resource-constrained devices remain fully functional, but with longer response times. Across all platforms the inference was executed entirely on-device within the browser session, without reliance on external servers.

VI. Conclusion

This work presented SOIL, a privacy-first framework for on-device plant disease diagnosis using mobile browsers. By combining crop-specific lightweight CNNs optimized through Chroma-Sense with TensorFlow.js and WebAssembly, SOIL enables efficient inference directly in the browser without requiring app installation, specialized hardware, or internet-based inference. The framework ensures that image data never leaves the device, thereby preserving privacy while remaining cross-platform compatible. Validation across multiple crops demonstrated competitive accuracy with reduced resource demands, highlighting SOIL's potential as a secure and scalable solution for real-world agricultural diagnostics. Since SOIL makes disease prediction lightweight and accessible, it also

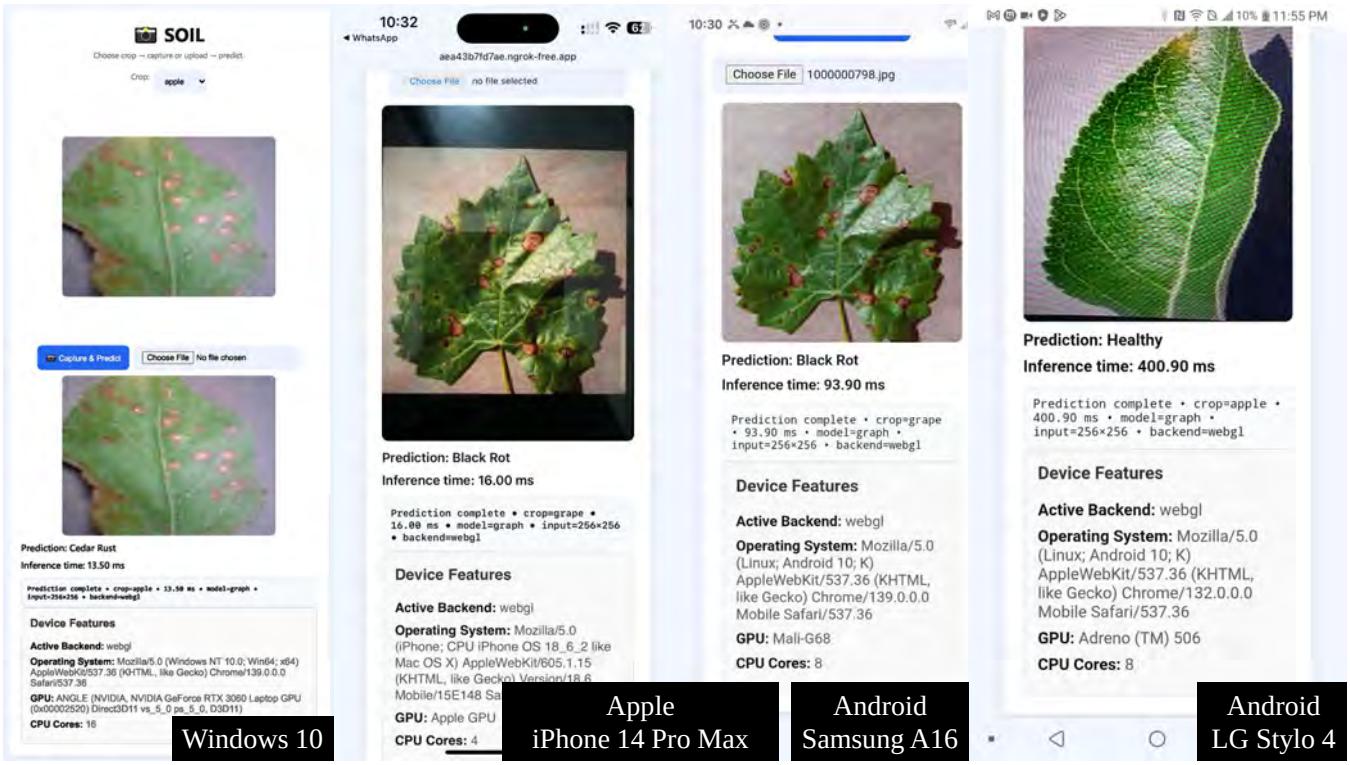

Figure 10: Inference latency of proposed SOIL framework across devices.

enables timely selection of the appropriate pesticide to be used in spraying systems [19] for better disease management.

REFERENCES

[1] A. Mitra, S. L. T. Vangipuram, A. K. Bapatla, V. K. V. V. Bathalapalli, S. P. Mohanty, E. Kougianos, and C. Ray, "Everything You wanted to Know about Smart Agriculture," 2022. [Online]. Available: 10.48550/ARXIV.2201.04754

[2] A. Bhargava, A. Shukla, O. P. Goswami, M. H. Alsharif, P. Uthansakul, and M. Uthansakul, "Plant leaf disease detection, classification, and diagnosis using computer vision and artificial intelligence: A review," *IEEE Access*, vol. 12, pp. 37 443–37 469, 2024.

[3] A. Siddiqua, M. A. Kabir, T. Ferdous, I. B. Ali, and L. A. Weston, "Evaluating plant disease detection mobile applications: Quality and limitations," *Agronomy*, vol. 12, no. 8, 2022.

[4] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Proc. International Conference on Computer Science and Electronics Engineering*, vol. 1, 2012, pp. 647–651.

[5] M. Jazayeri, "Some trends in web application development," in *Proc. Future of Software Engineering (FOSE '07)*, 2007, pp. 199–213.

[6] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel, S. Bileschi, M. Terry, C. Nicholson, S. N. Gupta, S. Sirajuddin, D. Sculley, R. Monga, G. Corrado, F. B. Viégas, and M. Wattenberg, "Tensorflow.js: Machine learning for the web and beyond," 2019. [Online]. Available: https://arxiv.org/abs/1901.05350

[7] S. Bileschi, E. Nielsen, and S. Cai, *Deep learning with JavaScript: neural networks in TensorFlow. js*. Simon and Schuster, 2020.

[8] K. K. Kethineni, S. Y. Wu, S. P. Mohanty, and E. Kougianos, "Chromasense: a memory-efficient plant leaf disease classification model for edge devices," in *Proc. IEEE Conference on Artificial Intelligence (CAI)*, 2025, pp. 1–6.

[9] A. T. Khan, S. M. Jensen, A. R. Khan, and S. Li, "Plant disease detection model for edge computing devices," *Frontiers in Plant Science*, vol. Volume 14 - 2023, 2023.

[10] M. F. Ahamed, A. Salam, M. Nahiduzzaman, M. Abdullah-Al-Wadud, and S. R. Islam, "Streamlining plant disease diagnosis with convolutional neural networks and edge devices," *Neural Computing and Applications*, vol. 36, no. 29, pp. 18 445–18 477, 2024.

[11] U. C. Akuthota, Abhishek, and L. Bhargava, "Plant disease detection on edge devices," in *Proc. Data Science and Applications*, 2024, pp. 337–349.

[12] C. S, S. Ghana, S. Singh, and P. Poddar, "Deep learning model for image-based plant diseases detection on edge devices," in *Proc. 6th International Conference for Convergence in Technology (I2CT)*, 2021, pp. 1–5.

[13] M. J. Karim, M. O. F. Goni, M. Nahiduzzaman, M. Ahsan, J. Haider, and M. Kowalski, "Enhancing agriculture through real-time grape leaf disease classification via an edge device with a lightweight cnn architecture and grad-cam," *Scientific Reports*, vol. 14, no. 1, p. 16022, 2024.

[14] M. Iftikhar, I. A. Kandhro, N. Kausar, A. Kehar, M. Uddin, and A. Dandoush, "Plant disease management: a fine-tuned enhanced cnn approach with mobile app integration for early detection and classification," *Artificial Intelligence Review*, vol. 57, no. 7, p. 167, 2024.

[15] M. A. H. Foysal, F. Ahmed, and M. Z. Haque, "Multi-class plant leaf disease detection: A cnnbased approach with mobile app integration," *International Journal of Computer Applications*, vol. 186, no. 41, p. 62–68, Sep. 2024.

[16] M. Zeeshan, M. S. Baghini, and A. Pandey, "Edgeplantnet: Lightweight edge-aware cyber–physical system for plant disease detection using enhanced attention cnns," *Pervasive and Mobile Computing*, vol. 110, p. 102059, 2025.

[17] A. Ali, "PlantVillage Dataset," 2019, last accessed April 25 2025. [Online]. Available: https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset

[18] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626.

[19] K. K. Kethineni, S. P. Mohanty, E. Kougianos, S. Bhowmick, and L. Rachakonda, "SprayCraft: Graph-Based Route Optimization for Variable Rate Precision Spraying," 2024. [Online]. Available: https://arxiv.org/abs/2412.12176