XAI Classification Platform:
A Unified Explainable AI System for Multi-Modal Deep Learning

Lucas BARREZ   Grégoire WORONIAK

January 2026

Abstract

This technical report presents the design, implementation, and integration decisions for a unified explainable AI platform that combines deepfake audio detection and medical image classification. The system integrates multiple pre-trained models with three XAI methods (LIME, SHAP, Grad-CAM) through a modern web interface. We describe our model selection and fine-tuning process, the architectural decisions for both frontend and backend, and the improvements made over the original source repositories. The platform enables direct comparison of XAI explanations and automatically adapts to different input modalities.

Contents

1 Introduction

This project was proposed and developed by Lucas Barrez and Grégoire Woroniak from TD group DIA6. The objective was to create a unified explainable AI platform by merging and refactoring two existing GitHub repositories focused on deepfake audio detection [1] and lung cancer detection [2]. The resulting platform provides a cohesive interface for both audio and image classification tasks with integrated explainability methods.

2 Project Objectives

The project requirements were structured around several key goals:

- **Unified Interface:** Merge and refactor two separate repositories into a single application with a cohesive interface, utilizing modern frameworks.
- **Multi-Modal Classification:** Support classification of both images and audio files (.wav format), with multiple pre-trained models for each modality. Support for additional formats constitutes a bonus.
- **XAI Integration:** Integrate compatible explainability methods, with LIME, Grad-CAM, and SHAP as mandatory techniques. Additional methods are considered a plus.
- **Comparison Dashboard:** Provide a comparison tab to analyze different XAI methods side-by-side on the same input data.
- **Adaptive Interface:** Automatically adapt available XAI methods based on the input data type (image or audio) to display only relevant options.
- **Clear Results Presentation:** Offer an intuitive interface presenting both classification results and their explanations in a user-friendly manner.

3 Classification Models

3.1 Fake or Real Dataset

3.1.1 Dataset Overview

The Fake or Real (FoR) dataset [3] is a comprehensive collection designed for deepfake audio detection. It contains 2-second normalized audio clips labeled as either real human speech or synthetic speech generated by various text-to-speech (TTS) systems. The dataset is specifically designed to test model generalization by including TTS engines in the test set that were not present in the training data.

3.1.2 Source Repository Analysis

The original deepfake audio detection repository [1] implemented a novel approach by converting audio signals into spectrograms, effectively transforming the audio classification problem into an image classification task. This enabled the use of well-established computer vision models. The repository provided comparative results for several model architectures, along with visualizations of three XAI methods and a demonstration video of the user interface.

3.1.3 Dataset Preparation

We adopted the spectrogram transformation approach from the source repository, as it allows leveraging powerful computer vision architectures. We used the normalized 2-second version of the FoR dataset, which is optimal for deep learning models as it provides consistent input dimensions and signal quality.

Critically, we preserved the original train-validation-test split provided with the dataset. This decision was deliberate: the test set is specifically designed to include TTS engines not seen during training, enabling proper evaluation of model generalization capabilities as described in Section III.D of [4]. This explains the lower accuracy on the test set compared to validation, which the original repository did not address by creating a custom split.

Moreover, we did not apply data augmentation. After initial experiments, augmented data yielded poor results. This outcome is logical: spectrograms represent time-frequency decompositions of audio signals, and typical image augmentations (rotation, zoom, flip) would distort these physical characteristics unnaturally, as real spectrograms never undergo such transformations.

Final dataset composition:

- Training: 13,957 audio samples
- Validation: 2,826 audio samples
- Test: 1,088 audio samples

3.1.4 Model Selection

For model selection, we retained the best-performing architectures from the source repository while updating to more recent variants. We excluded the custom CNN architectures that achieved the poorest results in the original work. We replaced ResNet with EfficientNetV2 [6], which represents a significant architectural improvement with better parameter efficiency and accuracy. We retained VGG16 [7] and MobileNetV2 [8] as they demonstrated strong performance in the baseline experiments.

3.1.5 Model Fine-Tuning and Results

Why Fine-Tuning? Transfer learning through fine-tuning leverages pre-trained weights from models trained on large-scale datasets (typically ImageNet) rather than training from scratch. This approach provides several advantages: faster convergence, better generalization with limited data, and incorporation of low-level feature detectors that transfer well across domains.

Fine-Tuning Methodology: We followed the best practices outlined in the official TensorFlow documentation [10]. The process consists of several phases:

1. **Preparation:** Load a pre-trained model (e.g., EfficientNet) without its top classification layer, and freeze all base model layers to preserve learned features.
2. **Customization:** Add a new classification head adapted to our binary task (real vs. fake), including a Dropout layer to prevent overfitting.
3. **Warm-up:** Train only the new classification head with a moderate learning rate, allowing it to adapt to the frozen feature extractor's outputs.
4. **Fine-Tuning:** Unfreeze the final layers of the base model and retrain the entire network with a learning rate 10-100$\times$ lower than the warm-up phase. This low learning rate prevents catastrophic forgetting of pre-trained features while allowing specialization.

5. **Model Selection:** Monitor validation performance to identify the best epoch and save the corresponding model. The test set is reserved for final evaluation only.

Hyperparameter Tuning: We systematically explored key hyperparameters:

- **Dropout Rate:** High values prevent overfitting but may cause underfitting; low values risk memorization. We balanced between regularization and capacity.
- **Fine-tune Layer Depth (fine_tune_at):** Freezing more layers provides stability and faster training but limits specialization; unfreezing more layers allows greater adaptation but risks degrading pre-trained features.
- **Learning Rate:** High rates enable fast learning but cause instability and may destroy pre-trained weights; low rates provide precise, stable adjustments but require more epochs to converge.

Results for the Fake or Real dataset:

Table 1: Model Performance on Fake or Real Dataset

Model	**Parameters**	**Dropout**	**Fine-tune**	**LR**	**Accuracy**
MobileNet-V2	3.4M	0.3	35.1%	1e-6	0.8373
MobileNet-V2	3.4M	0.6	35.1%	1e-6	0.8263
MobileNet-V2	3.4M	0.6	55.5%	1e-6	**0.8447**
VGG-16	138M	0.3	21.1%	1e-6	0.7610
VGG-16	138M	0.6	21.1%	1e-6	0.8042
VGG-16	138M	0.6	10.5%	1e-6	0.8042
EfficientNet-V2	54.1M	0.3	22.0%	1e-6	0.7371
EfficientNet-V2	54.1M	0.6	22.0%	1e-6	0.8162
EfficientNet-V2	54.1M	0.6	2.5%	1e-6	0.7316

MobileNetV2 achieved the best performance with the smallest model size, demonstrating excellent efficiency for audio deepfake detection.

3.2 CheXpert Dataset

3.2.1 Dataset Overview

The CheXpert dataset is a large-scale chest radiograph dataset for thoracic disease classification. It contains frontal and lateral chest X-rays labeled for 14 different observations, including various pathologies such as cardiomegaly, edema, consolidation, and pleural effusion. The dataset is designed to support multi-label classification in medical imaging.

3.2.2 Source Repository Analysis

The lung cancer detection repository [2] provided limited documentation. It appeared to reduce the multi-label CheXpert dataset to a binary lung cancer classification task, though the exact methodology was not clearly explained. The repository listed model names and comparison metrics but lacked implementation details.

3.2.3 Dataset Preparation

We chose to retain the multi-label nature of the dataset rather than simplifying to binary classification. This decision is clinically motivated: thoracic pathologies often share visual characteristics, and training a model to discriminate between multiple related conditions should improve its ability to recognize specific pathologies, including lung cancer-related findings.

We used the CheXpert Small version [5] available on Kaggle and retained half of the images due to computational constraints. All images were resized to 224×224 pixels to accelerate training while maintaining sufficient resolution for pathology detection. We applied light data augmentation (rotation, translation, zoom) appropriate for medical images to improve model generalization.

Label Processing: The CheXpert dataset employs a unique labeling scheme for each pathology:

- **1:** Pathology explicitly mentioned as present
- **0:** Pathology explicitly mentioned as absent
- **Null:** No mention of the pathology in the radiology report
- **-1:** Uncertain presence of the pathology

We adopted a clinically conservative labeling strategy. Blank labels were converted to 0 (absent), reasoning that if a radiologist does not mention a pathology, it is likely absent. More importantly, uncertain labels (-1) were converted to 1 (present). This decision reflects medical practice priorities: in clinical settings, false positives (flagging a potentially absent pathology) are preferable to false negatives (missing a pathology that may require treatment). This approach biases the model toward sensitivity over specificity, which is appropriate for a screening tool.

Final dataset composition:

- Training: 76,410 images
- Validation: 9,551 images
- Test: 9,551 images

3.2.4 Model Selection

For the CheXpert dataset, we retained DenseNet121 [9] from the source repository, as it has demonstrated strong performance on medical imaging tasks. We excluded AlexNet, which is an outdated architecture with excessive parameters and inferior performance compared to modern alternatives. We added EfficientNetV2, known for excellent performance on medical imaging tasks, and MobileNetV2, which achieved the best results on our audio classification task.

3.2.5 Model Fine-Tuning and Results

We applied the same fine-tuning methodology described in Section 3.1.5, adapting the output layer for multi-label classification with sigmoid activation and binary cross-entropy loss.

Classification Threshold and Metrics: For pathology detection, we adopted a classification threshold of 0.3 rather than the conventional 0.5. This clinically motivated decision reflects standard medical AI practice: when a model predicts even a 30% probability of pathology presence, alerting the physician is preferable to missing a potential diagnosis. This conservative threshold is common in medical screening systems, where sensitivity is prioritized and physicians make final diagnostic decisions after reviewing flagged cases.

For evaluation, we report Area Under the ROC Curve (AUC) as the primary metric rather than accuracy. AUC is the appropriate metric for multi-label classification tasks as it evaluates the model's ability to rank positive instances higher than negative ones across all possible thresholds, independent of class imbalance. Unlike accuracy, which can be misleading when some pathologies are rare, AUC provides a threshold-independent assessment of discriminative performance for each pathology label.

Results for the CheXpert dataset:

Table 2: Model Performance on CheXpert Dataset (Multi-Label Classification)

Model	**Parameters**	**Dropout**	**Fine-tune**	**LR**	**Test AUC**
MobileNet-V2	3.4M	0.3	55.5%	1e-6	0.7537
EfficientNet-V2	54.1M	0.3	22.0%	1e-5	**0.7595**
DenseNet121	8.0M	0.3	26.7%	1e-6	0.7383

EfficientNetV2 achieved the highest performance, demonstrating its effectiveness for medical image classification tasks.

4 Explainability Methods

4.1 Grad-CAM

4.1.1 Principle

Gradient-weighted Class Activation Mapping (Grad-CAM) [11] is a technique that produces visual explanations for CNN-based models. It uses gradients flowing into the final convolutional layer to understand the importance of each spatial location for a prediction. Grad-CAM generates a coarse localization map highlighting the regions in the input image that are most important for the model's decision.

The method computes the gradient of the target class score with respect to feature maps of the last convolutional layer, then performs a weighted combination of forward activation maps followed by a ReLU operation to produce the heatmap.

4.1.2 Implementation

Grad-CAM presented our most significant implementation challenge. While the official Keras [12] documentation provide clear guidance, our fine-tuned models created nested model structures that prevented straightforward Grad-CAM model construction. Keras could not properly reconstruct the computational graph needed for gradient computation through the nested architecture.

After extensive research without finding solutions in online forums, we received assistance from Gemini 3, which provided a working solution. Instead of using the fine-tuned model's input layer, we reconstructed the Grad-CAM model starting from the base model's input layer and rebuilding the computational graph on top. This approach bypassed the nested model issue and enabled successful Grad-CAM computation with fine-tuned Keras models.

The implementation extracts the last convolutional layer's activations, computes gradients with respect to the predicted class, and generates a heatmap overlay on the original input:

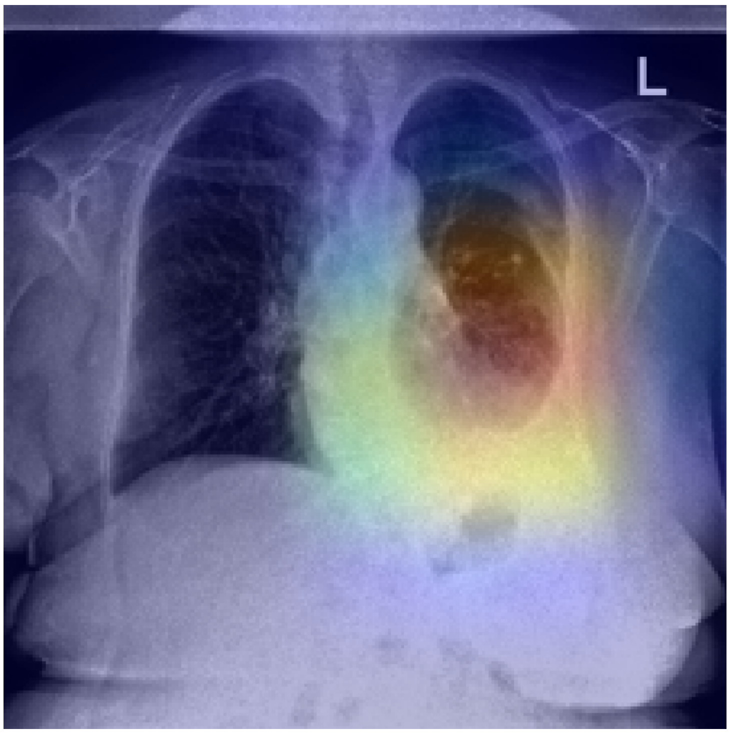


Figure 1: Grad-CAM visualization highlighting important regions for classification

4.2 SHAP

4.2.1 Principle

SHapley Additive exPlanations (SHAP) [13] is a unified approach to explain model predictions based on game theory. SHAP assigns each feature an importance value (Shapley value) for a particular prediction by considering all possible feature combinations. This ensures a fair distribution of the prediction value among features.

For image data, SHAP uses a partition-based approach where superpixels or patches are treated as features, and their contributions to the prediction are computed through perturbation analysis.

4.2.2 Implementation

SHAP implementation was more straightforward. The official SHAP documentation [14] provides comprehensive examples for image explanation. We followed the documented approach using the `ImageExplainer` class with appropriate background dataset sampling. The implementation requires minimal adaptation beyond setting the correct model prediction function and preprocessing pipeline.

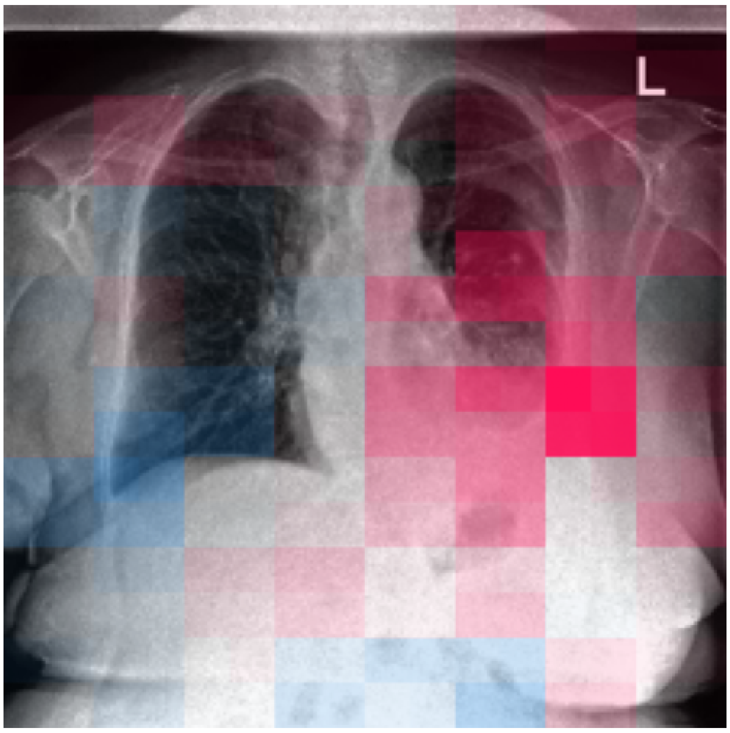


Figure 2: SHAP explanation showing pixel-level feature importance

4.3 LIME

4.3.1 Principle

Local Interpretable Model-agnostic Explanations (LIME) [15] explains individual predictions by learning an interpretable linear model locally around the prediction. For images, LIME segments the input into superpixels, perturbs the input by randomly turning superpixels on or off, obtains predictions for these perturbed inputs, and fits a linear model to approximate the classifier's behavior in that local region.

LIME's model-agnostic nature makes it applicable to any classifier, and its local focus ensures the explanations are faithful to the model's behavior for the specific instance being explained.

4.3.2 Implementation

LIME implementation closely followed the official documentation [16]. The provided examples for image classification were directly applicable to our use case. Our implementation is nearly identical to the reference code, requiring only adjustment of hyperparameters (number of samples, number of superpixels, etc.) to optimize explanation quality.

The `LimeImageExplainer` generates perturbed images by masking superpixels and uses the model's predictions on these perturbations to identify which regions contribute most to the classification decision.

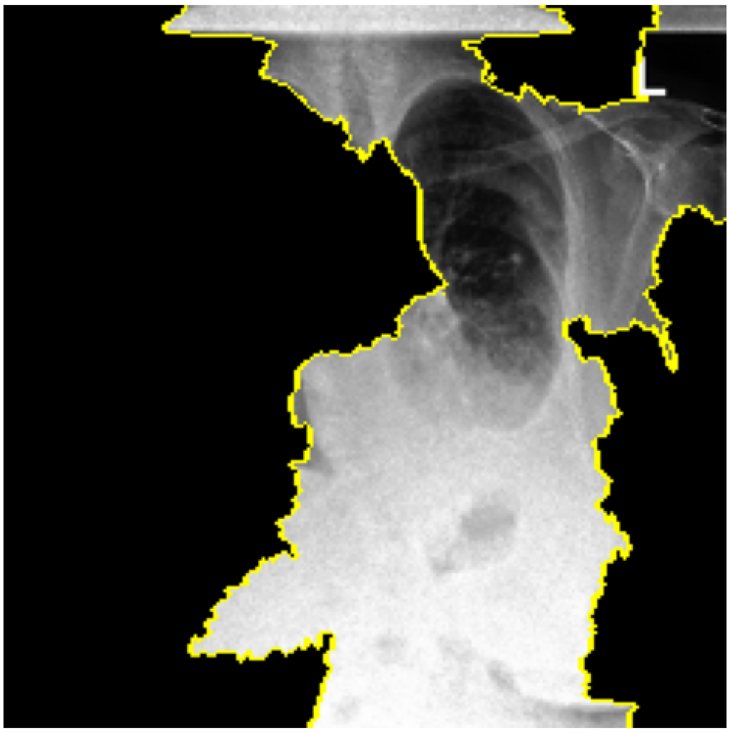


Figure 3: LIME explanation with superpixel-based importance

5 Backend Architecture

5.1 Why FastAPI?

We selected FastAPI [17] as our backend framework for several compelling reasons:

- **Performance:** FastAPI is built on Starlette and Pydantic, providing exceptional performance comparable to Node.js and Go frameworks. This is crucial for deep learning inference workloads.
- **Type Safety:** Python type hints and Pydantic models provide automatic request/response validation, reducing runtime errors and improving code maintainability.
- **Automatic Documentation:** FastAPI generates interactive API documentation (Swagger UI and ReDoc) automatically from code, facilitating frontend development and testing.
- **Async Support:** Native support for asynchronous request handling enables efficient I/O operations and concurrent request processing.
- **Modern Python:** FastAPI embraces modern Python features (type hints, async/await) while maintaining clean, readable code.

5.2 Architecture Overview

The backend follows a layered service-oriented architecture:

```
backend/
 api/          # API routes and endpoint definitions
    router.py
    v1/
        router.py
        endpoints/
            classification_models.py
            xai.py
 core/         # Configuration and constants
    constants.py
 models/       # Fine-tuned models with metadata
    audio/    # Audio classification models
    image/    # Image classification models
 schemas/      # Pydantic request/response schemas
    schemas.py
 services/     # Business logic and core functionality
     analysis_service.py
     audio_xai_service.py
     image_xai_service.py
     classification_service.py
     model_loader_service.py
     utils.py
```

This separation ensures clear responsibility boundaries: API layer handles HTTP concerns, services implement business logic, and schemas define data contracts.

5.3 Service Layer

The service layer encapsulates all business logic:

- **AnalysisService:** Orchestrates the complete analysis pipeline, coordinating between classification and XAI services based on frontend requests.
- **AudioXAIService:** Generates XAI visualizations for audio spectrograms, returning base64-encoded images for all three methods (LIME, SHAP, Grad-CAM).
- **ImageXAIService:** Generates XAI visualizations for medical images with identical interface to audio service for consistency.
- **ClassificationService:** Performs model inference to obtain predicted labels and confidence scores.
- **ModelLoaderService:** Handles lazy loading of models and metadata, managing the Grad-CAM models, preprocessing modules, and class names.

This design enables independent testing of each service and facilitates future extensions such as additional XAI methods or data modalities.

5.4 API Endpoints

The API exposes two primary endpoints:

Get Available Models:

```
GET http://localhost:8000/v1/classification_models/model_list/
```

Returns metadata about all available models including their type (audio/image), architecture name, and supported XAI methods.

Perform Prediction with XAI:

```
POST http://localhost:8000/v1/classification_models/predict/
Content-Type: application/json

{
  "model": "efficientnet_audio",
  "xai_methods": ["lime", "shap", "gradcam"],
  "file_type": "audio",
  "file_b64": "<base64_encoded_file>"
}
```

Accepts a model identifier, list of XAI methods, file type, and base64-encoded file. Returns classification results with corresponding XAI visualizations.

6 Frontend Implementation

6.1 Why Next.js?

We chose Next.js [18] as our frontend framework for its modern features:

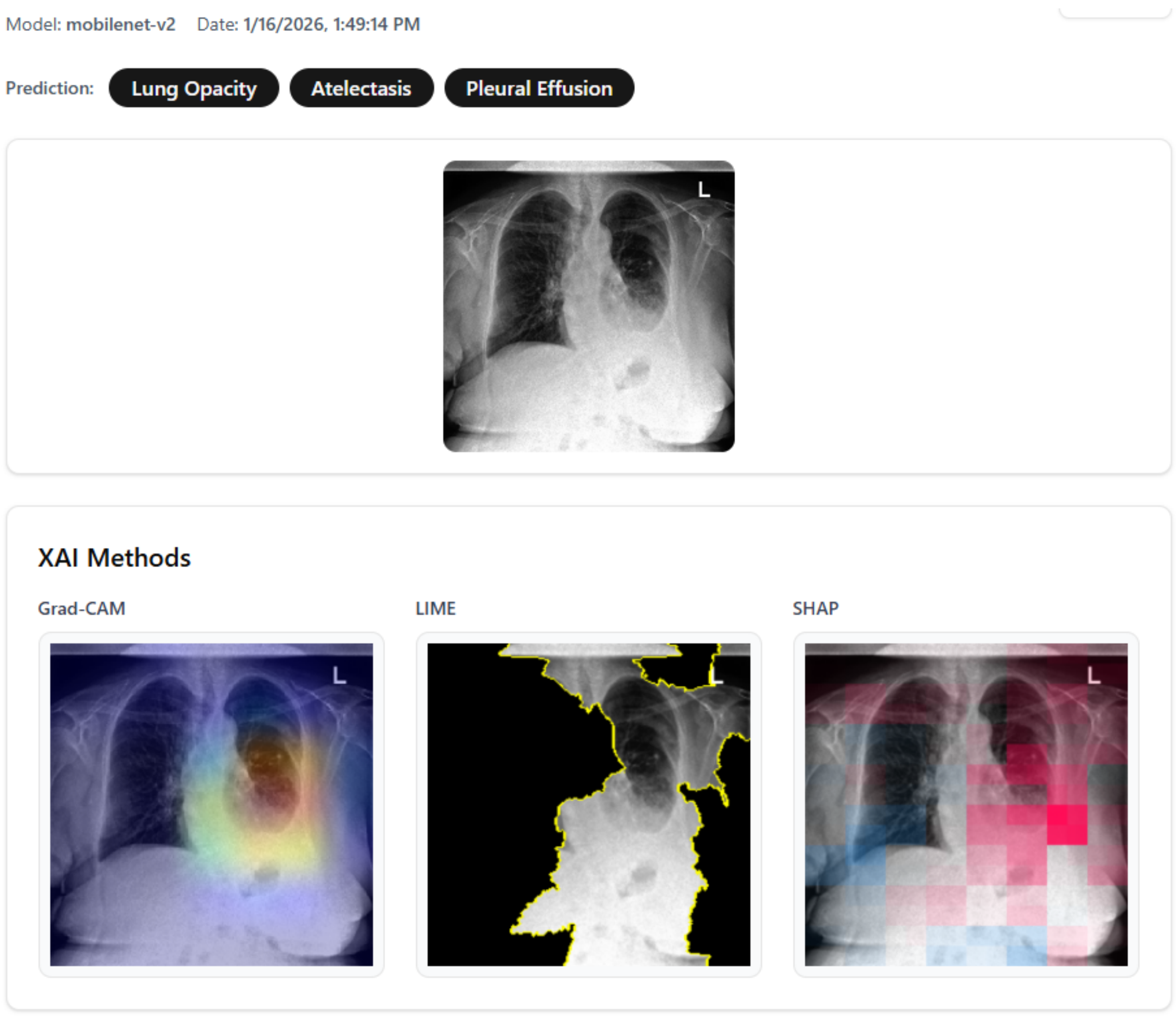
- **React Framework:** Built on React with additional capabilities like server-side rendering and static generation for improved performance.
- **Type Safety:** Native TypeScript support ensures type-safe development and reduces runtime errors.
- **Developer Experience:** Fast refresh, automatic routing, and excellent tooling accelerate development.
- **Modern UI Libraries:** Excellent compatibility with component libraries like shadcn/ui [19], which we use for consistent, accessible components.
- **Performance:** Automatic code splitting and optimization provide fast page loads and smooth user experience.

6.2 Key Features

The frontend implements several user-focused features:

- **File Input:** Drag-and-drop interface for audio (.wav) and image files with instant preview of uploaded content.
- **Sample Files:** Pre-loaded samples from test datasets enable quick testing without requiring users to source their own data.
- **Smart Model Selection:** Dynamic filtering displays only models compatible with the uploaded file type, preventing invalid combinations.

- **Flexible XAI Selection:** Users can select one or multiple XAI methods before analysis, and add additional methods post-analysis for comparison.
- **Analysis Workflow:** File submission triggers backend analysis with loading states and error handling, ensuring responsive user experience.
- **Side-by-Side Comparison:** Results panel displays all selected XAI explanations simultaneously for direct comparison.
- **Persistent Storage:** Users can save analysis results with custom names, stored in browser local storage for future reference and comparison.


Model: mobilenet-v2 Date: 1/16/2026, 1:49:14 PM

Prediction: Lung Opacity Atelectasis Pleural Effusion

Figure 4: Frontend interface showing saved analysis results

6.3 Implementation Approach

The frontend development leveraged GitHub Copilot with Claude Sonnet 4.5 to accelerate UI implementation. We maintained a simple, comprehensible architecture to retain full control over the codebase. AI assistance proved particularly valuable for quickly generating complete, polished UI components while we focused on application logic and user experience design.

The component structure follows React best practices with clear separation between:

- UI components (buttons, dialogs, layout elements)

- Feature components (file upload, result display, comparison panel)
- Server actions (API communication with backend)
- Hooks (state management, persistent storage)

7 Deployment

The entire project is containerized using Docker. A single command launches both frontend and backend services:

```
docker-compose up --build
```

The `docker-compose.yaml` configuration orchestrates both services:

- Frontend: Accessible at `http://localhost:3000`
- Backend: Accessible at `http://localhost:8000`

This containerized approach ensures consistent deployment across different environments and simplifies dependency management for both Python (backend) and Node.js (frontend) ecosystems.

8 Improvements Over Source Repositories

8.1 Unified Interface for Audio and Image

The source repositories were separate projects with independent codebases. The lung cancer detection repository lacked any user interface for testing models and XAI methods. Our platform provides a single, cohesive interface for both modalities, enabling rapid experimentation without switching between applications.

8.2 Pre-loaded Test Samples

Our platform includes curated samples from both datasets' test sets, which the models have never seen during training. This enables immediate testing and demonstration without requiring users to possess or source appropriate audio and medical image files. The samples showcase the models' generalization capabilities on truly held-out data.

8.3 XAI Comparison and Persistence

The platform enables side-by-side comparison of multiple XAI methods on the same input, facilitating qualitative assessment of explanation quality and consistency. Users can save analyses with descriptive names for later review or comparison against different inputs, maintaining a persistent analysis history in the browser.

8.4 Modern Model Architectures

We updated the model selection with more recent and efficient architectures:

- Replaced ResNet with EfficientNetV2, achieving better accuracy with improved parameter efficiency
- Removed AlexNet due to its outdated architecture and poor parameter efficiency
- Added models that demonstrated strong performance across both domains

8.5 Proper Dataset Splits

We preserved the carefully designed train-validation-test splits provided with the FoR dataset, particularly the test set containing unseen TTS engines. This enables proper evaluation of generalization capabilities, which the original repository's custom split did not address.

8.6 Multi-Label Classification

For CheXpert, we retained the multi-label formulation rather than reducing to binary classification. This clinically motivated decision enables the model to learn discriminative features across multiple thoracic pathologies, potentially improving performance on any specific condition of interest.

9 Conclusion

9.1 Objectives Achieved

We successfully addressed all project objectives:

- **Unified Interface:** Created a single Next.js application integrating both audio and image classification with seamless user experience.
- **Multi-Modal Support:** Implemented support for both .wav audio and medical image formats with appropriate preprocessing pipelines.
- **XAI Integration:** Successfully integrated LIME, SHAP, and Grad-CAM, overcoming significant technical challenges with nested model architectures.
- **Comparison Features:** Enabled side-by-side XAI comparison with persistent storage for longitudinal analysis.
- **Adaptive Interface:** Implemented smart filtering that displays only applicable models and methods based on input type.
- **Clear Presentation:** Designed an intuitive interface with immediate visual feedback and comprehensive result displays.

9.2 Future Improvements

Several enhancements could extend the platform's capabilities:

- **Additional XAI Methods:** Integrate Integrated Gradients, Attention Rollout, or TCAV for richer explanation diversity.
- **More Datasets:** Extend to other domains such as natural images, text classification, or time-series data.
- **Comparison Analytics:** Implement quantitative metrics for XAI method comparison and correlation analysis.
- **Model Performance:** Explore ensemble methods, larger architectures, or domain-specific pre-training for improved accuracy.
- **Input Format Support:** Add support for additional formats such as CSV, Excel for tabular data, or video inputs.

- **Web Deployment:** Deploy the platform to cloud infrastructure for public access without local installation requirements.
- **Batch Processing:** Enable analysis of multiple files simultaneously for efficiency in research workflows.
- **Explanation Export:** Provide functionality to export explanations and results in various formats (PDF reports, CSV metrics).

References

[1] Deepfake Audio Detection with XAI Repository. https://github.com/Guri10/Deepfake-Audio-Detection-with-XAI

[2] Lung Cancer Detection Repository. https://github.com/schaudhuri16/LungCancerDetection

[3] Fake or Real Dataset. https://www.kaggle.com/datasets/mohammedabdeldayem/the-fake-or-real-dataset

[4] FoR Dataset: A Dataset for Synthetic Speech Detection, Ricardo Reimao and Vassilios Tzerpos, 2020. https://bil.eecs.yorku.ca/wp-content/uploads/2020/01/FoR-Dataset_RR_VT_final.pdf

[5] CheXpert: A Large Chest Radiograph Dataset. https://www.kaggle.com/datasets/ashery/chexpert

[6] Tan, M., & Le, Q. V. (2021). EfficientNetV2: Smaller Models and Faster Training. *International Conference on Machine Learning.*

[7] Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR.*

[8] Sandler, M., et al. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *CVPR.*

[9] Huang, G., et al. (2017). Densely Connected Convolutional Networks. *CVPR.*

[10] TensorFlow Transfer Learning and Fine-tuning Guide. https://www.tensorflow.org/tutorials/images/transfer_learning

[11] Selvaraju, R. R., et al. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *ICCV.*

[12] Keras Grad-CAM Implementation Guide. https://keras.io/examples/vision/grad_cam/

[13] Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *NeurIPS.*

[14] SHAP Documentation. https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/image.html

[15] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *KDD.*

[16] LIME Documentation. https://www.kaggle.com/code/yohanb/explaining-keras-model-with-lime

[17] FastAPI Framework. https://fastapi.tiangolo.com/

[18] Next.js Framework. https://nextjs.org/

[19] shadcn/ui Component Library. https://ui.shadcn.com/