IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

PredictiX: A Multi-Disease Prediction System Using MERN Stack and Machine Learning

Omprakash Pramod Singh[1], Shivam Kamtaprasad Pandey[2], Atharv Shrikant Powar[3] and Dr. Roopali Lolage[4]
Shree L.R. Tiwari College of Engineering, Mumbai[1-4]

Abstract: *: Early disease detection can make the difference between a manageable condition and a life-threatening one — yet for millions of people, getting to a specialist in time simply is not a realistic option. Hospitals are overburdened, diagnostic equipment is expensive, and the gap between a patient noticing something is wrong and receiving a meaningful clinical opinion can stretch into weeks or months. PredictiX was built with that gap in mind. It is a web-based multi-disease prediction platform that combines four purpose-trained machine learning models with a full-stack MERN architecture, making preliminary risk assessment for heart disease, diabetes, breast cancer, and lung cancer accessible to anyone with a browser. The system uses Logistic Regression for cardiac screening, a Support Vector Machine for diabetes risk evaluation, a Convolutional Neural Network for breast tissue analysis, and InceptionResNetV2 for pulmonary image classification. On the engineering side, it introduces a prescription upload and auto-fill feature powered by regex parsing, generates downloadable diagnostic PDF reports through pdf-lib, handles authentication via JWT tokens, and runs all Python-based ML inference through Node.js Child Processes — removing the typical dependency on a separate Flask server. Tested on standard benchmark datasets, the system achieves around 85% accuracy for heart disease, 82% for diabetes, and above 90% for both cancer detection modules. The platform is intended to serve as a smart, accessible first step in the diagnostic journey — not a clinical replacement, but a meaningful early signal.*

Keywords: Multi-Disease Prediction, MERN Stack, Machine Learning, Logistic Regression, Support Vector Machine, CNN, InceptionResNetV2, Deep Learning, Healthcare, React.js, Node.js, MongoDB, Transfer Learning, PDF Report Generation

I. INTRODUCTION

There is a well-documented problem in healthcare that technology has only partially solved: most people who develop a serious illness — whether it is a cardiac condition, Type-II diabetes, or cancer in its early stages — do not get a timely diagnosis. This is not because diagnostic tools do not exist. It is because those tools require trained professionals to operate, expensive laboratories to run, and an institutional infrastructure that remains out of reach for a large fraction of the global population. In a country like India, where the doctor-to-patient ratio still falls well below WHO recommendations, the problem is especially sharp.

Machine learning has, over the past decade, offered a genuinely different approach to this problem. Instead of relying on a specialist to interpret every test result, a trained model can process clinical parameters or radiological images and return a meaningful risk estimate in seconds. The research literature is now extensive — there are reliable models for heart disease, for diabetes, for various cancers — but almost all of them live as standalone scripts, trained on benchmark datasets and never connected to a real user interface, let alone deployed in a way that someone outside a research lab could actually use.

This paper describes PredictiX, an attempt to close that gap. PredictiX brings four machine learning models — covering heart disease, diabetes, breast cancer, and lung cancer — into a single, fully deployed web application. Users can log in, enter their clinical values or upload medical images, receive a prediction within seconds, and download a

Copyright to IJARSCT
www.ijarsct.co.in

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 20, April 2026 Impact Factor: 8.2

formatted diagnostic report. For heart disease and diabetes, they can also upload a prescription and have the form fields filled automatically through regex-based parsing. The backend is written entirely in Node.js and Express, with Python ML inference handled through Child Processes — no Flask server required.

The goal was not to build the most accurate model ever published for any of these diseases. The goal was to build something that actually works end-to-end: that a real person could open in a browser, use without confusion, and walk away from with a concrete, actionable result. That turned out to be a harder problem than tuning a classifier, and this paper documents how we approached it.

The contributions of this work can be summarized as follows:

- A single web platform that consolidates prediction for four major diseases, each backed by a dedicated and separately trained model.
- An architectural pattern that invokes Python ML inference directly from Node.js via Child Processes, keeping the deployment footprint to a single server.
- A prescription upload feature that parses structured values from unstructured medical text using regular expressions, significantly reducing data entry friction.
- Automated PDF report generation with patient details, prediction outcomes, and confidence scores — formatted for record-keeping rather than just on-screen display.
- A fully functional web deployment demonstrating that a multi-model healthcare application can be shipped with standard modern tooling.
-

II. LITERATURE SURVEY

The application draws on a reasonably well-established body of machine learning research in clinical prediction, though the integration of multiple models in a single production web application is itself somewhat novel. This section reviews the most relevant prior work for each of the four disease domains and for multi-disease platforms in general.

A. Heart Disease Prediction

Logistic regression applied to structured cardiac data has a long track record in clinical informatics. The Cleveland Heart Disease dataset, introduced by Detrano et al. in 1989 [1], became the de facto benchmark for this problem and continues to appear in published comparisons today. Studies consistently place logistic regression in the 80–88% accuracy range on this dataset, which is competitive with more complex ensemble approaches when the feature set remains limited to the standard 13 clinical attributes. What makes logistic regression attractive here beyond raw performance is interpretability — the coefficient magnitudes directly indicate which features are driving the risk estimate, which matters when you need to explain a result to a non-technical user. For that reason, we chose to stay with logistic regression rather than moving to a black-box model, even though random forests and gradient boosting have been shown to squeeze out marginal gains on the same dataset.

B. Diabetes Prediction

The Pima Indians Diabetes Dataset [10] has served as the standard diabetes prediction benchmark since the late 1980s and has been used in hundreds of published studies. Among the classifiers tested on this data, Support Vector Machines with radial basis function kernels have shown consistently strong performance, particularly in handling the class imbalance that is inherent to the dataset — diabetic samples make up only about 35% of the records. Kavakiotis et al. [2] provide a thorough survey of ML methods applied to diabetes research and identify SVM as among the strongest performers, with accuracies in the 78–83% range depending on preprocessing choices. One practical consideration that influenced our implementation was the need for probability outputs rather than just binary predictions — for a patient-facing application, saying there is a 76% likelihood of diabetes is considerably more useful than a bare yes or no. We addressed this through Platt scaling applied post-training.

C. Breast Cancer Detection via Histopathology

Convolutional neural networks applied to histopathology images have produced results that frequently match or exceed trained pathologists on controlled benchmarks. The BreakHis dataset, introduced by Spanhol et al. [3], standardized

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

evaluation on breast tissue microscopy and has become a common reference point. Transfer learning from large pretrained architectures such as VGG16 and ResNet has been particularly effective when annotated histopathology data is scarce. For PredictiX, we trained a custom CNN from scratch rather than using a pretrained backbone, partly because the input resolution requirements and the available dataset size made fine-tuning less straightforward, and partly because a smaller custom model is faster to serve at inference time — which matters when you want real-time feedback on image upload.

D. Lung Cancer Detection via Radiology

Lung cancer carries one of the highest mortality rates of any cancer, in part because it is frequently diagnosed at a late stage when treatment options are limited. The central challenge in automated detection from chest radiographs is that early pulmonary nodules are small, low-contrast, and easily confused with normal anatomical structures. Deep architectures with strong representational capacity are therefore necessary, and InceptionResNetV2 — which combines the multi-scale feature extraction of Inception modules with the gradient flow advantages of residual connections [4] — has shown reliable results across multiple radiological imaging benchmarks. Wang et al. [5] demonstrated the scale at which such models can be trained when hospital-scale datasets are available; for PredictiX, we worked with a curated subset and relied on data augmentation and careful fine-tuning of the final classification layers to achieve performance that transfers meaningfully to new inputs.

E. Multi-Disease and Integrated Platforms

There is a noticeable gap in the published literature between research demonstrating ML performance on individual disease benchmarks and actual deployed systems that non-specialist users can interact with. Most publicly documented multi-disease prediction platforms are built with Flask or Streamlit, serve a single session at a time, have no authentication layer, and are never deployed beyond a local machine or a brief demo. A number of project repositories exist on GitHub following a broadly similar structure — train four or five models, expose them through a Streamlit UI — but none of them, to our knowledge, include prescription parsing, PDF report generation, MongoDB-backed user histories, or a production deployment that handles concurrent access. PredictiX was designed specifically to address that cluster of missing features.

III. SYSTEM ARCHITECTURE

The architecture of PredictiX was shaped by a fairly practical constraint: we wanted to ship a working system without managing multiple servers or containerized deployments. Most student or prototype ML projects that combine a JavaScript frontend with Python models solve this by running a separate Flask or Fast API server alongside the Node backend. That works, but it complicates deployment significantly and increases the number of running services to maintain. The design decision we made instead was to invoke Python inference scripts directly from the Node.js backend using the built-in Child Process module. The tradeoff is that cold inference adds a process-spawn overhead of roughly 500–700ms, which is acceptable for a diagnostic tool where users expect to wait a moment for results.

A. Presentation Tier

The frontend is a React single-page application built with Vite and written in TypeScript. State is managed through React Context API rather than a heavier library like Redux, which kept the component tree manageable for a four-module application. Toast notifications (React-Toastify) provide immediate feedback on prediction completion, authentication events, and upload errors. The UI design was prototyped in Figma before implementation, which helped maintain visual consistency across the four very different predictor interfaces — some form-based, some image-upload-based. Report generation is handled on the client side through pdf-lib, which assembles a formatted PDF from the prediction result without any server round-trip.

B. Application Tier

The Express.js backend exposes a REST API with distinct route groups for authentication, prediction, file upload, and user account management. File uploads are handled by Multer, which saves incoming prescription documents and medical images to a local uploads/ directory before the relevant route handler processes them. For prediction requests,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

the route handler spawns a Python Child Process, passes the input data through stdin or as command-line arguments depending on whether the model takes structured data or a file path, reads the output from stdout, and returns the parsed result to the client. This keeps the Node layer clean — it knows nothing about scikit-learn or TensorFlow; it just runs scripts and relays results.

C. Data Tier

MongoDB stores user accounts, authentication tokens, and the history of prediction events. We chose MongoDB rather than a relational database partly because the schema for prediction records varies meaningfully across the four disease modules — a heart disease record has 13 clinical fields while a lung cancer record has essentially just an image path and an outcome — and a document store handles that variation more naturally than a normalized relational schema would. Mongoose is used for schema definition and validation on the Node side.

D. Technology Stack Summary

Table I: PredictiX Technology Stack

Layer	Technology / Library
Frontend	React.js (Vite), TypeScript, Context API, CSS
Backend	Node.js, Express.js, Child Process (ML runner)
Database	MongoDB (users, predictions, history)
ML / DL	scikit-learn, TensorFlow / Keras, Python 3.10
Reports	pdf-lib (client-side PDF generation)
File Upload	Multer (prescriptions & medical images)
Auth	JWT tokens, bcrypt password hashing

E. End-to-End Request Flow

A typical prediction request follows this path: the user authenticates and receives a JWT, selects a predictor, fills in values or uploads an image (optionally using the prescription auto-fill feature), and submits the form. The React frontend sends a POST request to the appropriate Express route with the JWT in the Authorization header. The backend validates the token, hands the input to the relevant Python script via Child Process, waits for the output, stores the result in MongoDB, and returns the prediction with a confidence score. The frontend displays the result in a results panel, triggers a toast notification, and makes the PDF download available. The entire round-trip, for structured data models, typically completes in under two seconds.

IV. METHODOLOGY

A. Dataset Selection and Description

Each of the four disease modules is backed by a different dataset, chosen based on public availability, citation history, and alignment with the clinical features we wanted to expose in the user interface.

The Cleveland Heart Disease dataset from the UCI Machine Learning Repository contains 303 patient records with 13 clinical attributes: age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting ECG results, maximum heart rate achieved, exercise-induced angina, ST depression, peak exercise ST slope, number of major vessels coloured by fluoroscopy, and thalassemia type. The binary target indicates the presence or absence of heart disease. This is one of the most widely cited datasets in medical ML and gives us a reliable training baseline.

The Pima Indians Diabetes Dataset contains 768 records, each with 8 features: number of pregnancies, plasma glucose concentration, diastolic blood pressure, triceps skin fold thickness, two-hour serum insulin, BMI, diabetes pedigree function, and age. The target is binary — diabetic or not. Around 35% of records are positive, which creates a mild class imbalance that we handle during preprocessing.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 20, April 2026 Impact Factor: 8.2

For breast cancer, we used a curated histopathology image dataset with binary labels (benign / malignant). Images were collected from microscopy slides and vary in staining intensity and tissue density, which introduces realistic variation that the model needs to handle. All images were resized to 224×224 for CNN input.

For lung cancer, we assembled a dataset of chest X-ray images labelled as normal or cancerous. Images were resized to 299×299 to match the input dimensions expected by InceptionResNetV2. Both cancer image datasets were augmented to compensate for limited sample size.

B. Preprocessing Pipeline

For the structured datasets, preprocessing began with a check for missing or physiologically implausible values. Columns where zero was not a valid observation — glucose, blood pressure, insulin, BMI, skin thickness — had their zero values treated as missing and replaced with the column median computed on the training split. Categorical features such as chest pain type and ECG results were one-hot encoded. All continuous features were then normalized to zero mean and unit variance using scikit-learns StandardScaler, with the scaler fit on training data and applied identically to validation and test sets to prevent leakage. The heart disease data was split 70:15:15 across train, validation, and test; the diabetes data followed the same ratio.

Image preprocessing used a two-stage pipeline. In the first stage, images were resized and pixel values were scaled to the [0, 1] range. In the second stage, real-time augmentation was applied during training: random horizontal and vertical flips, rotations of up to 15 degrees, and random brightness and contrast shifts. Class weights were calculated and passed to the model's fit function to address the imbalance present in both cancer datasets. The train/validation/test split for images was 80:10:10 with stratification on the label.

C. Model Training and Architecture

Heart Disease — Logistic Regression: We trained a standard binary logistic regression classifier using scikit-learn, with L2 regularization and an inverse regularization strength of C=1.0, which was selected based on 5-fold cross-validation on the training set. The model outputs both a class prediction and a probability score. The probability is passed directly to the frontend as the confidence value shown in the results panel. On the held-out test set the model achieved 85% accuracy and an AUC-ROC of 0.91. Serum cholesterol, maximum heart rate, and ST depression were the three features with the highest absolute coefficient values.

Diabetes — SVM: An SVM with an RBF kernel was trained using scikit-learn's SVC class with probability=True, which enables Platt scaling for calibrated probability outputs. Hyperparameter search over C in {0.1, 1, 10} and gamma in {scale, auto} was conducted via 5-fold grid search on the training set, settling on C=1 and gamma=scale. On the held-out test set the model achieved 82% accuracy and an AUC-ROC of 0.87. As expected clinically, glucose concentration and BMI contributed most strongly to predictions.

Breast Cancer — CNN: The CNN consists of three convolutional blocks, each containing a Conv2D layer, Batch Normalization, ReLU activation, and MaxPooling. The output of the third block is flattened and passed through two fully connected layers with dropout (rate 0.4) before a sigmoid output unit. The model was trained using the Adam optimizer at a learning rate of 0.001, with binary cross-entropy loss and early stopping based on validation AUC with a patience of 8 epochs. Training ran for a maximum of 60 epochs. On the test set the model achieved above 90% accuracy.

Lung Cancer — InceptionResNetV2: We used the InceptionResNetV2 architecture with ImageNet-pretrained weights, freezing all layers up to the last 20 and replacing the original classification head with GlobalAveragePooling2D, a Dense layer of 512 units with ReLU, Dropout at 0.5, and a sigmoid output. The model was fine-tuned at a low learning rate of 1×10^{-5} using Adam, again with early stopping on validation AUC. The relatively low learning rate was important here to prevent the pretrained representations from being overwritten too quickly on the smaller medical imaging dataset. On the test set the model achieved above 92% accuracy.

D. Prescription Parsing and Auto-Fill

One feature that distinguishes PredictiX from comparable systems is the ability to upload a prescription document and have the relevant form fields populated automatically. When a user uploads a prescription on the heart disease or

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

diabetes predictor page, the backend reads the file contents as plain text and applies a set of regular expressions to extract labelled numerical values. Patterns are written to match common prescription phrasings — for example, expressions like 'Glucose: 126 mg/dL', 'BMI 28.4', 'BP: 130/85' and their reasonable variations. Matched values are mapped to the corresponding form fields and returned to the frontend as a JSON object, which the React component uses to pre-populate the input controls. The user can then review and adjust any values before submitting. This feature meaningfully reduces the effort required to use the predictor when a patient already has recent test results in hand.

E. PDF Report Generation

After a prediction is returned, the frontend uses pdf-lib to assemble a downloadable report entirely in the browser — no server round-trip is needed at this stage. The report includes the platform branding, the date and time of the prediction, the user's entered parameters formatted in a readable table, the binary prediction outcome, the model's confidence score as a percentage, and a clear disclaimer stating that the result is a preliminary ML-based assessment and not a substitute for professional clinical evaluation. Reports for image-based predictions also include the filename of the uploaded image. The generated PDF is served as a blob download from the browser.

V. RESULTS AND DISCUSSION

Testing was carried out on held-out sets that had no overlap with the training or validation splits used during model development. The results across the four modules are summarized in Table II.

Table II: Model Performance Summary

Disease	Algorithm	Input Type	Accuracy
Heart Disease	Logistic Regression	Clinical Data	~85%
Diabetes	SVM (RBF)	Metabolic Data	~82%
Breast Cancer	CNN	Histopathology Image	>90%
Lung Cancer	InceptionResNetV2	X-Ray / CT Scan	>92%

The heart disease model's 85% accuracy is consistent with the upper end of what logistic regression typically achieves on the Cleveland dataset, and the AUC-ROC of 0.91 suggests that the model discriminates well across confidence thresholds — not just at the default 0.5 cutoff. More complex ensemble methods have been reported slightly higher on this dataset in isolation, but the interpretability benefit of logistic regression was judged worth the small performance tradeoff for a patient-facing application.

The diabetes SVM's 82% accuracy on the Pima Indians test split is in line with the best published results for this dataset when basic preprocessing is applied. What matters more from a practical standpoint is the calibration of the probability output — an uncalibrated SVM decision value is not very meaningful as a patient-facing confidence score, and the Platt scaling applied post-training produces probability estimates that are meaningfully aligned with actual empirical risk.

The CNN for breast cancer detection exceeded 90% accuracy. Sensitivity was prioritized during training — false negatives in cancer screening carry a much higher cost than false positives — and this priority was reflected in the class weighting strategy. The model does occasionally produce false positives on images with unusual staining patterns, and we note this as a limitation rather than trying to obscure it.

InceptionResNetV2's performance on lung cancer detection exceeded 92%. Transfer learning proved critical here: training the full architecture from scratch on a dataset of this size would not have produced comparable results. The pretrained ImageNet representations capture low-level visual features — edges, textures, local contrast patterns — that transfer well to medical imaging even though the source domain is very different.

On the system side, structured data inference completes in under two seconds including the Child Process spawn time. Image inference is slower, typically 3–5 seconds, due to the additional model loading overhead — TensorFlow loads the SavedModel at process start each time. This is a known inefficiency in the current architecture; a persistent model

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

server would eliminate the loading time but would require a different deployment strategy. For the current use case it is acceptable.

Table III: PredictiX vs. Comparable Platforms

Feature	PredictiX	Flask Prototype	Streamlit App
Multi-Disease	Yes (4)	Partial	Partial
Auth / Login	JWT	None	None
PDF Reports	Yes	No	No
Rx Auto-fill	Yes	No	No
Image Input	Yes	Limited	Limited
Production Deploy	Yes	No	No

Table III situates PredictiX relative to the kinds of multi-disease prototypes that typically appear in academic project repositories. The key differentiators are not performance-related — they are architectural and user-experience-related: authentication, prescription parsing, PDF reports, and an actual production deployment.

VI. WEB APPLICATION

The PredictiX web application is organized into eight main areas, each corresponding to a distinct user task. The design principle throughout was to keep the interface as uncluttered as possible: the models handle complexity internally, and users should only be asked to provide what is strictly necessary.

A. Authentication

New users register with an email and password; returning users log in and receive a JWT that persists in local storage. All four predictor routes are protected — unauthenticated requests are redirected to the login page. Passwords are hashed with bcrypt before storage. There is no social login in the current version, though it is on the roadmap.

B. Predictors Dashboard

After logging in, users land on a dashboard that presents the four available predictors with a brief description and entry point for each. The dashboard also shows a summary of the user's recent predictions, pulled from their MongoDB history.

C. Heart Disease and Diabetes Predictors

Both structured-data predictors present a form with clearly labelled input fields, range hints where relevant, and an optional prescription upload button at the top. When a prescription is uploaded, a loading state is shown while the backend parses the document; matched fields are then populated and highlighted so the user can verify them before submitting. The prediction result appears in a results panel below the form, showing the outcome and confidence score alongside a brief plain-language explanation of what the result means.

D. Cancer Detection Modules

The breast cancer and lung cancer modules follow a simpler interface: a drag-and-drop image upload area, a submit button, and a results panel. Preview of the uploaded image is shown before submission so the user can confirm they have selected the right file. Processing time is displayed as a loading indicator during inference. Given the higher clinical stakes of cancer detection, the results panel for these modules includes a more prominent disclaimer than the structured-data predictors.

E. Report Download

A download button appears in the results panel of all four predictors after a prediction has been returned. Clicking it triggers the pdf-lib report assembly and initiates a file download through the browser. No network request is made at this point — the PDF is assembled entirely from data already present in the React component state.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

VII. ADVANTAGES OF THE SYSTEM

Several design choices in PredictiX set it apart from comparable academic systems:

- Consolidating four disease predictors in a single platform means a user with multiple health concerns does not need to navigate four different tools — the prediction history in their account also accumulates across all modules, giving a more complete picture over time.
- The Node.js Child Process architecture keeps the deployment footprint to a single server. This is not just a cost consideration — it also reduces the number of moving parts that can fail in production and makes the system easier to maintain.
- Prescription auto-fill is the feature that most users in informal testing responded to positively. Entering 13 clinical values manually is tedious and error-prone; having them extracted from an existing prescription document and pre-populated removes a significant friction point.
- PDF report generation on the client side means reports are available even if the server goes down after the prediction has been returned. They are also formatted for printing and can be shared directly with a physician.
- JWT-based authentication with bcrypt-hashed storage means the system handles user data to a standard that would not embarrass a production application, rather than the no-auth approach common in prototype ML demos.
- Transfer learning for the cancer detection modules means that high-accuracy image classification is achievable without the kind of massive labeled dataset that only large hospital networks or research institutes can assemble.

VIII. LIMITATIONS

We think it is important to be specific about where PredictiX falls short, rather than listing limitations in the generic way that research papers sometimes do.

- The prescription parser works well for digitally generated prescription PDFs with consistent formatting. It struggles with handwritten prescriptions, unconventionally formatted documents, or cases where abbreviations differ from the patterns encoded in the regex set. OCR integration would help but is not yet implemented.
- Model inference restarts a Python process on every request. For image models in particular, the TensorFlow model load time adds 2–3 seconds to each response. A persistent inference service would eliminate this overhead but would require a hosting environment that supports long-lived background processes.
- The cancer detection models were trained and tested on curated benchmark datasets. Real clinical images often have different acquisition conditions, staining protocols, and equipment characteristics. The models have not been validated on prospectively collected hospital data, and their performance in a true clinical deployment would need to be evaluated separately.
- All models produce a single scalar confidence score alongside the binary prediction. There is currently no feature-level explanation — users cannot see which of their input values contributed most to the result. SHAP integration is planned for a future version.
- The platform has not yet been tested with a statistically meaningful user study. Informal feedback during development was positive, but formal usability and accuracy evaluation with a clinical population has not been conducted.

IX. FUTURE SCOPE

The current version of PredictiX is a working system, but there are several directions in which it can be meaningfully extended:

- OCR-based prescription parsing using Tesseract or Google Vision API would replace the regex approach and handle a much wider range of prescription formats, including handwritten documents.
- A persistent ML inference service — potentially a lightweight Flask or Fast API server running as a background process, or a containerized microservice — would eliminate the process-spawn latency and enable batch requests.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 20, April 2026 Impact Factor: 8.2

- Additional disease modules are feasible with the current architecture. Kidney disease, liver disorders, and skin lesion classification are the most natural next additions, given the availability of public datasets and the clinical prevalence of these conditions.
- Wearable device integration would allow continuous health metric ingestion from fitness trackers and smartwatches, enabling the structured-data predictors to be triggered automatically when readings fall outside expected ranges.
- SHAP value computation for the structured models and Grad-CAM visualization for the CNN and InceptionResNetV2 modules would give users and clinicians a meaningful window into what is driving each prediction — which matters both for trust and for triage.
- A React Native mobile application would extend the platform to iOS and Android, enabling the cancer detection modules to use the device camera directly for image capture rather than requiring a file upload.
- Regional language support would make the platform accessible to the large fraction of India's population that is more comfortable in Hindi, Tamil, Telugu, Bengali, or other regional languages than in English.

X. CONCLUSION

PredictiX set out to answer a specific question: can a small team build a multi-disease ML prediction system that is genuinely usable, properly deployed, and meaningfully different from the prototype scripts that typically represent the end state of student ML projects? We believe the answer is yes, and this paper has documented how.

The four models — logistic regression for heart disease, SVM for diabetes, CNN for breast cancer, InceptionResNetV2 for lung cancer — were chosen to match the nature of each prediction task rather than to achieve the highest possible benchmark numbers. The engineering decisions — Node.js Child Processes for inference, prescription regex parsing, client-side PDF generation, MongoDB for user history — were made to keep the system simple to deploy and maintain while delivering features that actually matter to users.

The platform is live. That matters. Most research in this space produces a model and a paper; PredictiX produces a model, a paper, and a working application that a person with a health concern can open in a browser and use today. Whether it helps them is ultimately a clinical question that a randomized study would need to answer. But as a demonstration that the infrastructure problem — connecting ML models to users in a reliable, usable way — is solvable with standard web technologies, we think it succeeds.

Future work will focus on addressing the limitations described above, starting with OCR-based prescription parsing and persistent model serving, and moving toward explainability and clinical validation. The codebase is maintained in a version-controlled repository and is available for review.

ACKNOWLEDGMENT

The authors are grateful to the Department of Information Technology at Shree L.R. Tiwari College of Engineering, Thane, for providing access to the computational resources and the academic environment that made this project possible. Special thanks are owed to Dr. Roopali Lolage, whose guidance throughout the development and documentation process was invaluable. We also acknowledge the open-source communities behind React.js, Node.js, TensorFlow, and scikit-learn, and the Kaggle contributors who made the datasets used in this project publicly available.

REFERENCES

[1] R. Detrano, A. Janosi, W. Steinbrunn et al., "International Application of a New Probability Algorithm for the Diagnosis of Coronary Artery Disease," American Journal of Cardiology, vol. 64, no. 5, pp. 304–310, 1989.

[2] I. Kavakiotis, O. Tsave, A. Salifoglou, N. Maglaveras, I. Vlahavas, and I. Chouvarda, "Machine Learning and Data Mining Methods in Diabetes Research," Computational and Structural Biotechnology Journal, vol. 15, pp. 104–116, 2017.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 20, April 2026
Impact Factor: 8.2

[3] F. Spanhol, L. S. Oliveira, C. Petitjean, and L. Heutte, "A Dataset for Breast Cancer Histological Image Classification," IEEE Transactions on Biomedical Engineering, vol. 63, no. 7, pp. 1455–1462, 2016.
[4] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in Proc. AAAI Conference on Artificial Intelligence, 2017.
[5] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "ChestX-Ray8: Hospital-Scale Chest X-Ray Database and Benchmarks," in Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
[6] A. Esteva, B. Kuprel, R. A. Novoa et al., "Dermatologist-level classification of skin cancer with deep neural networks," Nature, vol. 542, pp. 115–118, 2017.
[7] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, "Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus," in Proc. Annual Symposium on Computer Application in Medical Care, 1988.
[8] S. Lundberg and S. Lee, "A Unified Approach to Interpreting Model Predictions," in Advances in Neural Information Processing Systems (NeurIPS), 2017. (SHAP framework)
[9] nevonProjects.com, "Multiple Disease Prediction System using Machine Learning," n.d. [Online]. Available: https://nevonprojects.com
[10] hallowshaw, "PredictiX: Multi-Disease Prediction System," GitHub, 2024. [Online]. Available: https://github.com/hallowshaw/PredictiX
[11] Kaggle Inc., "Heart Disease Dataset" (johnsmith88); "Diabetes Prediction Dataset" (iammustafatz), n.d. [Online]. Available: https://www.kaggle.com