German Medical Data Sciences 2025: GMDS Illuminates Health
R. Röhrig et al. (Eds.)
© 2025 The Authors.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/SHTI251408

Use of Client-Side Machine Learning Models for Privacy-Preserving Healthcare Predictions - A Deployment Case Study

Yacoub Abelard NJIPOUOMBE NSANGOU[a,b], Rajib KUMAR HALDER[c], Ashraf UDDIN[d], Laurenz ENGEL[a], Fruzsina KOTSIS[e], Ulla T. SCHULTHEISS[e], Johannes RAFFLER[f], Robin KOSCH[g,h], Michael ALTENBUCHINGER[a], Helena U. ZACHARIAS[g,h], Gabi KASTENMÜLLER[b] and Jürgen DÖNITZ[a,b,1]
[a]*Dept. of Medical Bioinformatics, University Medical Center Göttingen,Germany*
[b]*Institute of Computational Biology, Helmholtz Zentrum München, Munich, Germany*
[c]*Dept. of CS and Engineering, Jagannath University, Dhaka, Bangladesh*
[d]*School of Information Technology, Deakin University, Australia*
[e]*Institute of Genetic Epidemiology, Department of Data Driven Medicine, Faculty of Medicine and Medical Center, University of Freiburg, Freiburg, Germany*
[f]*Institute for Digital Medicine, University Hospital Augsburg, Germany*
[g]*Peter L. Reichertz Institute for Medical Informatics of TU Braunschweig, Germany*
hHannover Medical School, Hannover Medical School, Hanover, Germany
ORCiD: Jürgen Dönitz https://orcid.org/0000-0002-8401-8851

Abstract. Introduction Machine learning (ML) and deep learning (DL) models in healthcare traditionally rely on server-centric architectures, where sensitive patient data is transmitted to external servers for processing via frameworks like Flask, raising significant privacy concerns. This work demonstrates a privacy-preserving approach by executing healthcare prediction models entirely within the web browser. **Methods** Our approach leverages existing browser-based machine learning and deep learning technologies such as TensorFlow.js and ONNX Runtime Web, along with direct JavaScript implementations, to ensure all computations remain on the client side. We showcase three implementation strategies based on model complexity: direct JavaScript implementation for simple equation-based models, ONNX-based conversion and execution for medium-complexity models like Random Forest and finally TensorFlow.js deployment for complex deep learning models such as Optimized Convolutional Neural Networks. **Results** Our results indicate that client-side deployment is both feasible and effective for healthcare prediction models, preserving original performance metrics while offering substantial privacy benefits. **Conclusion** This approach guarantees patient data never leaves the user's device, eliminating risks associated with data transmission and making it particularly advantageous in healthcare settings where data confidentiality is critical, while also supporting offline functionality.

Keywords. Machine Learning, Deep Learning, Privacy, Web Browser, Confidentiality, Decision Support Systems, Clinical

[1] Corresponding Author, Jürgen Dönitz, University Medical Center Göttingen (UMG), Goldschmidtstr. 1, 37077 Göttingen, Germany; E-Mail: juergen.doenitz@bioinf.med.uni-goettingen de.

1. Introduction

Machine Learning (ML) and Deep Learning (DL) models have advanced healthcare decision-making by supporting diagnosis, prognosis and treatment planning based on patient data [1, 2]. Traditionally, these models are deployed in a server-centric architecture, where patient data is transmitted to servers on the internet. In this setup, backend frameworks like Flask [3] or Django [4] manage the requests, relay the data to the predictive model, and return the results to the client [5,6].

However, this approach introduces privacy considerations that remain relevant even with proper consent mechanisms. While the General Data Protection Regulation (GDPR) establishes data processing agreements and explicit patient consent as legal frameworks for server-based processing [7, 8], several scenarios still present challenges: cross-border data transfers between healthcare systems without established agreements [9], emergency medical situations where comprehensive consent procedures may be incomplete [10], research collaborations with strict institutional data governance policies [11] and international settings where varying regulatory frameworks create compliance complexities [9]. Additionally, regardless of consent status, minimizing data transmission provides an extra layer of protection against potential breaches, unauthorized access or unexpected regulatory changes. Even without demographic information, unusual combinations of laboratory values (such as extreme creatinine levels combined with distinctive blood cell counts) can create unique fingerprints that potentially allow re-identification [12].

This study addresses the research question: How can healthcare machine learning models be deployed to ensure complete patient data privacy through client-side execution while maintaining clinical accuracy and usability across different model complexities?

Several approaches to privacy-preserving ML deployment have been explored. Federated learning [13] allows models to be trained across multiple devices without sharing the underlying data, but requires coordination across multiple participants and may leak information through gradient inversion attacks on shared model updates. Secure multiparty computation (SMPC) enables collaborative model inference across multiple parties through cryptographic protocols without revealing individual inputs, though it involves complex coordination requirements and significant computational and communication overhead due to cryptographic operations [14]. Homomorphic encryption permits computations on encrypted data, but its high computational and memory costs often limit practical deployment [15]. Split learning partitions neural networks between client and server, with clients computing initial layers locally before sending intermediate activations to servers for deeper layer processing. While this approach reduces the computational burden on client devices, it incurs significant communication overhead from transmitting feature representations at each forward and backward pass, and remains predominantly suited to neural network-based models [16]. Trusted execution environments (TEEs) provide hardware-based secure enclaves for privacy-preserving model execution, though they require specialized hardware infrastructure and remain vulnerable to side-channel attacks [17]. Local differential privacy (LDP) guarantees privacy by injecting noise into individual data points before aggregation, though excessive noise can degrade model performance [18].

Beyond these cryptographic and distributed approaches, healthcare organizations commonly employ local deployment strategies. These include Python frameworks such as scikit-learn for statistical analysis and machine learning, R environments for statistical computing, containerized solutions using Docker and virtual machine deployments within institutional infrastructure, including hospital infrastructure [19]. While these local deployment approaches provide excellent computational performance and are well-suited for research environments with dedicated IT support, they present specific implementation requirements: Python and R frameworks require specialized programming expertise and environment setup, Docker containerization needs orchestration knowledge and infrastructure management, virtual machines demand significant hardware resources and face compatibility challenges across different clinical systems and traditional local deployments may lack the cross-platform compatibility and ease of distribution that web-based approaches provide.

Recent advancements in web technologies offer a complementary approach that addresses different deployment scenarios. ML frameworks for browsers such as TensorFlow.js [20], Brain.js [21], Open Neural Network Exchange (ONNX) Runtime Web [22] and WebDNN [23], now allow sophisticated models to run directly in web browsers. This approach particularly benefits individual clinicians who need accessible decision-support tools without requiring IT infrastructure setup or specialized technical expertise. Compared to Python scikit-learn or R-based local implementations, browser-based deployment offers distinct advantages: immediate cross-platform accessibility without environment setup, elimination of dependency management issues and simplified distribution to end-users lacking programming expertise. However, local Python and R frameworks maintain advantages in computational performance, access to specialized libraries and integration with existing research workflows. Goh et al. [24] comprehensively reviewed front-end deep learning technologies, highlighting the potential of browser-based machine learning in addressing privacy concerns while providing interactive, accessible solutions.

Unlike traditional privacy-preserving methods that primarily focus on secure data transmission or processing, our client-side approach eliminates data transmission entirely during the inference phase. This provides a complementary privacy solution that operates independently of consent frameworks, institutional agreements or regulatory compliance mechanisms.

In this work, we implement and evaluate a fully client-side approach to healthcare model deployment across different model complexities, focusing on kidney disease prediction models as a case study. We chose to focus on kidney disease prediction models due to the significant public health burden of chronic kidney disease (CKD), which affects approximately 10% of the global population [25] and is the 11th leading cause of death globally with 1.2 million annual deaths from kidney failure [26]. Moreover, as highlighted in recent surveys of nephrologist end-users, there is a growing need for reliable and accessible clinical decision support tools in this domain [27]. Additionally, several well-documented models of varying complexity exist in this domain, making it an ideal test case for evaluating different client-side implementation strategies.

2. Methods

2.1. Materials and Tools

We used for client-side deployment ONNX (version 1.17.0) [22] with skl2onnx for model conversion and TensorFlow.js (version 4.22.0) [20] with its conversion tool. For user interface development, we utilized standard web technologies (JS, HTML and CSS).

Data visualization was handled using D3.js (version 7.8.5) [28]. Deployment was managed using Nginx [29] as a static web server and GitLab pages for hosting.

2.2. Model Selection and Acquisition

Rationale for Model Selection: We selected three kidney disease prediction models to systematically evaluate client-side deployment feasibility across different model complexities. Our selection criteria prioritized: (1) computational diversity – from simple equations to medium-complexity random forests and compute-intensive CNNs – to systematically test browser performance limits, (2) implementation reproducibility – availability of executable formats (published equations, .pkl weights, trainable .ipynb) ensured fair comparison without reimplementation bias and (3) deployment methodology coverage – models requiring distinct browser technologies (native JavaScript, ONNX conversion, TensorFlow.js) to demonstrate the spectrum of available frameworks.

The three selected models, each representing a different level of complexity, are:

1. Equation-based Linear Models / Mathematical Equations (Zacharias et al. [30]): These minimal-complexity clinical calculators predict the probability of kidney failure requiring replacement therapy within four years. The Z6 equation (6 clinical variables) is specifically recommended in the 2024 KDIGO Clinical Practice Guideline for CKD evaluation and management as an externally validated risk equation for predicting kidney failure [26], making it an ideal candidate for demonstrating clinical-grade browser deployment. The Z14 equation uses these 6 variables plus 8 additional clinical parameters. We retrieved formulas along with all necessary coefficients and parameter values from the published literature.
2. Random Forest Model (RF) (Halder et al. [31]): This medium-complexity machine learning model was obtained as a ready-to-use .pkl file from the authors' published code repository. The model achieved 100% accuracy in predicting chronic kidney disease status based on laboratory and clinical parameters. We specifically selected this model because the authors provided both trained weights and a Flask-based server deployment, enabling direct comparison between traditional server-side approaches and our browser-based implementation.
3. Deep Learning Model (Mondol et al. [32]): This Optimized Convolutional Neural Network (OCNN) was specifically selected to demonstrate browser feasibility for complex deep learning architectures. The model was not available as a pre-trained file, so we re-executed the original Jupyter notebook (.ipynb file) provided through the authors' GitHub repository without modifying the parameters to generate a usable .h5 file. This model achieved 98.75% accuracy in binary CKD classification using 24 variables.

2.3. Framework Evaluation and Selection

We systematically evaluated available browser-based ML frameworks based on three primary criteria: hardware acceleration capabilities, active development status and robust model conversion tools.

Selection Criteria and Process: TensorFlow.js was chosen as it has emerged as the leading framework for browser-based deep learning [24], offering WebGL and WebAssembly hardware acceleration support along with comprehensive model conversion tools (tensorflowjs_converter). ONNX Runtime Web was selected for its framework-agnostic model representation, efficient WebAssembly runtime and robust conversion pipeline through skl2onnx for scikit-learn models.

Alternative Frameworks Considered: We evaluated but rejected several alternatives: Brain.js was excluded due to lack of hardware acceleration; Keras.js was deprecated in favor of TensorFlow.js; WebDNN was not selected due to different technical requirements for our use case. PyTorch models could follow similar deployment approaches through ONNX export or conversion to TensorFlow format, but were not required for our selected models.

2.4. Client-Side Processing Architecture

Our privacy-preserving architecture keeps all data processing local to the user's device. Unlike traditional approaches that rely on server-side frameworks such as Flask or Django, our applications consist entirely of static files (HTML, JavaScript, CSS and model files) served via a web server (e.g., Nginx) or static hosting service (e.g., GitLab Pages). When users interact with the application, their input remains on their device and is processed locally using JavaScript equations for the risk prediction models, ONNX Runtime for the random forest model, or TensorFlow.js for the OCNN model. Results are visualized directly in the browser using D3.js. The application uses localStorage to persist user input and sessionStorage to store temporary results, thereby enhancing usability while ensuring that sensitive healthcare data never leaves the user's device.

2.5. Implementation strategy

We developed three distinct implementation approaches tailored to the selected models, each optimized for different computational requirements while maintaining our core privacy-preserving architecture:

- Simple models (Equation-based linear models - Zacharias et al. [30]): For the two risk prediction equations (Z6 and Z14) from Zacharias et al. [30], we implemented a direct JS solution. These linear models use a deterministic Cox proportional hazards structure requiring basic mathematical operations available in JavaScript's native Math library. The implementation organizes essential parameters as arrays of JavaScript objects, with each object containing metadata including parameter labels, model coefficients, valid input ranges and unit conversion factors. This structured approach enables comprehensive unit conversion capabilities for different clinical measurement units.
- Medium-complexity models (RF - Halder et al. [31]): For the RF model from Halder et al. [31], we used the ONNX format as an intermediate representation. The pre-trained .pkl file, which is Python's native serialization format for storing trained ML models, was converted to ONNX using skl2onnx [33], a specialized converter that transforms scikit-learn models into the ONNX format. This conversion process preserves the model's structure and learned parameters while making it compatible with browser-based execution environments. The ONNX Runtime JS API was used for model inference in the browser and we

implemented custom error handling to manage potential issues during inference such as missing outputs or runtime errors.

- Complex Deep Learning model (OCNN - Mondol et al. [32]): For the OCNN developed by Mondol et al. [32], we employed TensorFlow.js. The pre-trained model was converted using the tensorflowjs_converter tool, producing a JSON file for the model architecture and a binary file for the weights and biases. Custom components were registered using TensorFlow.js's serialization API. Input data underwent pre-processing, including min-max scaling and standardization to match the original training conditions.

The implementation of these three different approaches follows a common workflow pattern illustrated in Figure 1. Our process consists of four key stages: (1) starting with different model types (Simple models, RF model and DL model), (2) converting each into a browser-compatible format (JavaScript Arrays, ONNX Format, and TensorFlow.js Format), (3) deploying them as static files (simple models via Nginx server and the RF/DL models via GitLab pages) and (4) executing them within the browser through a user interface that processes data locally and displays results. This unified workflow ensures that despite the varying complexity of models and deployment platforms, all implementations share the same privacy-preserving foundation where computations occur entirely within the client's browser.

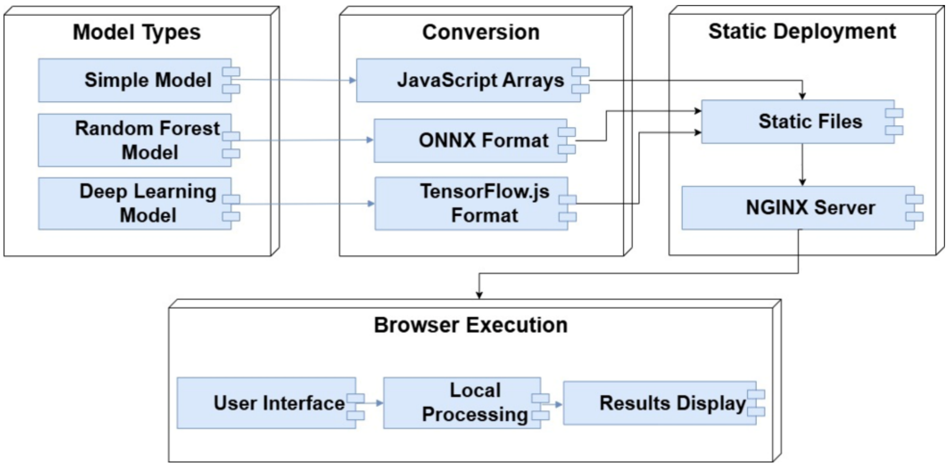

Figure 1. Client-side model deployment workflow. Models (equation-based, RF, Deep learning) are converted to browser-compatible formats (JavaScript Arrays, ONNX, TensorFlow.js) and deployed via Nginx server. All processing occurs locally in the browser, keeping patient data on-device.

2.6. Performance Analysis and Network Behavior Verification

To validate both performance characteristics and privacy guarantees, we implemented comprehensive monitoring across all three model deployments. Our evaluation methodology focused on quantifying real-world performance metrics and confirming complete absence of external data transmission during model operation. Testing was conducted across multiple browsers (Chrome v134, Firefox v137 and Edge v135) on Windows 10 systems with 16GB RAM, with multiple execution cycles to account for environmental variability.

Test Data Specification: For performance evaluation, we used realistic clinical data profiles based on the original model development datasets. For the Zacharias models, we conducted performance testing using representative clinical parameter values derived from the German Chronic Kidney Disease (GCKD) study population characteristics as reported in the original publication [30], which included 4,915 patients with variables such as age, sex, eGFR, UACR, hemoglobin, and serum laboratory values. For both the Halder Random Forest model [31] and the Mondol OCNN model [32], we utilized clinical test profiles derived from the UCI machine learning repository's chronic kidney disease dataset - the same dataset both authors used for training their respective models. This dataset contains 400 instances with 24 clinical attributes including laboratory values (such as serum creatinine, blood urea, hemoglobin, blood pressure) and clinical indicators (such as diabetes, hypertension, coronary artery disease), among others. We verified that all test parameter values remained within the original training variable ranges and conducted multiple test iterations across different parameter combinations to evaluate computational consistency, execution timing, and browser performance characteristics under varying computational loads.

Our monitoring approach specifically focused on:

- Network activity verification: Custom middleware was implemented to intercept all outgoing network requests by overriding the browser's native fetch and XMLHttpRequest APIs. This approach enabled monitoring of network activity during model execution, with particular emphasis on distinguishing prediction requests from those related to model loading.
- Performance timing: We used high-precision timestamps via the browser's Performance API to measure four critical intervals: model loading time (initialization to ready state), pre-processing time (input transformation prior to inference), inference time (actual model execution), and total prediction time (end-to-end user experience).
- Resource utilization: We tracked memory consumption using framework-specific APIs (TensorFlow.memory() for OCNN) and browser performance monitoring (for ONNX Runtime and JS implementations). Model size was measured through content-length headers and file size analysis.

3. Results

3.1. Successful Model Deployment Outcomes

The implementation strategies described in Methods resulted in three fully functional, privacy-preserving web applications, each demonstrating successful client-side deployment across different model complexity levels.

Deployment Outcomes: The equation-based Kidney Failure Risk Calculator (Zacharias et al.) is accessible at https://ckdn.app/tools/eskdcalc/, providing immediate risk predictions through direct JavaScript implementation. The ONNX-based Random Forest Model (Halder et al.) is available at https://kidneypredict-rf-93614a.pages.gwdg.de/, offering CKD classification with complete source code at https://gitlab.gwdg.de/MedBioinf/metabolomics/ckdmodels/kidneypredict-rf. The TensorFlow.js implementation of the OCNN Model (Mondol et al.) operates at https://kidneypredict-ocnn-bc08ba.pages.gwdg.de/ with source code available at https://gitlab.gwdg.de/MedBioinf/metabolomics/ckdmodels/kidneypredict-ocnn.

Functional Verification: All deployed applications successfully maintained their original performance metrics through systematic cross-platform validation. We validated model accuracy by comparing outputs between the original model implementations and our browser-based deployments using identical clinical input datasets. For the Zacharias equations, browser calculations matched reference implementations using published coefficients. For the Random Forest model, we confirmed concordant results across three platforms: the original Python .pkl implementation, our browser ONNX deployment and the authors' Flask-based server application. For the OCNN model, our browser TensorFlow.js implementation produced identical classification results compared to the original Python model implementation. This validation process confirmed that client-side deployment preserved computational accuracy across all model complexity levels.

3.2. Privacy Protection Verification Results

Our comprehensive network monitoring confirmed complete privacy protection across all implementations. The monitoring framework successfully intercepted and analyzed all network requests during model operation phases.

Network Isolation Results: After initial model loading, network monitoring detected zero instances of outbound data transmission during prediction phases across all three implementations. The custom middleware confirmed that patient input data, intermediate calculations, and prediction results remained entirely within the browser environment.

Offline Functionality Validation: The applications are standard static web applications composed of HTML, CSS, JavaScript and model files. They can be deployed locally by downloading all necessary files to a protected machine and launching them directly from a local hard disk using a modern web browser. This setup enables full functionality (including data input, prediction and result visualization) without requiring any connection to external servers. As such, the applications are well suited for use in regulated healthcare environments where internet access is restricted or prohibited. Browser storage mechanisms (localStorage for input persistence and sessionStorage for temporary results) enhanced usability while maintaining complete local data processing.

3.3. Performance Analysis Results

Table 1 presents performance metrics across all implementations, with all metrics reported for Google Chrome v134 as the reference browser. Performance testing was conducted using clinical test profiles as detailed in Methods section 2.6. Server-side deployment metrics are included for comparison.

Performance Variability Analysis: Performance metrics demonstrated 10-15% variation across multiple test runs due to browser optimization states and system resource availability. ONNX memory usage showed fluctuation due to browser garbage collection timing, while TensorFlow.js measurements remained more consistent due to its integrated memory management API. Initial model executions consistently showed 15-20% slower performance due to JS engine warm-up effects. Despite these performance variations, privacy protection remained absolute - network monitoring confirmed no data transmission occurred during predictions after initial asset loading, ensuring patient data never left the device.

For the server-side comparison, we examined the Flask implementation by Halder et al. [31], which uses Python's time module to measure execution time between prediction start and end. While server-side implementations require the same total prediction time for every request due to network communication overhead, our client-side approach eliminates this network dependency: after an initial model loading phase, subsequent predictions execute significantly faster with complete privacy protection.

Cross-Browser Compatibility Results: Testing across Chrome, Firefox and Edge revealed execution speed variations up to 20% while maintaining consistent prediction accuracy. All browsers successfully executed all model types, confirming broad compatibility across major browser engines

Table 1. Performance Metrics for Client-Side Deployment

Metric	Risk prediction (Z6)	ONNX (RF)	TensorFlow.js (OCNN)	Server-Side (Flask)†
Inference Time (ms)	0.10	17.10	94.40	10‡
Total Prediction Time (ms)	0.20	557.70	148.00	250.00‡
Preprocessing Time (ms)	0.10	0.20	0.10	-
Model Loading Time (ms)	43.00	534.80	45.70	-
Model Size (KB)	22	101.93	7.62	-
Memory Usage (MB)	0.965	0.92	0.02	-

†For comparative purposes, we include metrics from the Flask-based server deployment of a Random Forest model for CKD prediction developed by Halder et al. [31]. ‡These values represent observations from direct testing of the server-side application. Inference time is based on the execution time measured in their code, while total prediction time includes network latency and round-trip communication. Missing values (-) indicate metrics that are not directly comparable between client and server architectures: model size does not impact client resources in server deployments; memory usage occurs on the server rather than user devices and model loading typically happens once at server start-up rather than per user request.

3.4. Comprehensive Deployment Approach Comparison

To directly address the innovation achieved through our browser-based approach compared to traditional Python, R and server-based frameworks, Table 2 presents a comprehensive comparison across key deployment factors from the perspective of clinical end-users.

Deployment Innovation: Traditional Python/R frameworks require environment setup, dependency management and technical expertise, creating significant barriers for clinical adoption. Our approach eliminates these requirements through simple web browser access while maintaining equivalent prediction accuracy and superior privacy protection through local processing. This represents a fundamental shift from execution speed optimization to accessibility-focused innovation in healthcare ML deployment.

Table 2. Comprehensive Comparison of ML Deployment Approaches in Healthcare

Factor	**Browser-based (Our Approach)**	**Local Statistical Environment (Python/R)**	**Server-based (Flask/Django)**
Setup Requirements	None (web browser only)	Language installation, required packages, model files	None (web browser only)
Programming Expertise Required	Basic web interaction	Basic scripting commands for model loading and prediction	Basic web interaction
Cross-platform Deployment	Immediate (any modern browser)	Platform-specific installations on clinical workstations	Immediate (any modern browser)
End-user Technical Barriers	Minimal (URL access)	Moderate to High (statistical environment setup, dependency management)	Minimal (URL access)
Data Privacy Model	Complete isolation (never leaves device)	Local processing (device-dependent security)	Data transmission required (compliance dependent)
Offline Capability	Yes (after initial loading)	Yes (post-installation)	No (required server connectivity)
Performance Characteristics	JavaScript engine dependent (0.10-94ms)	Native performance (1-50 ms depending on model complexity)	Network latency + server processing (200 ms+ typical)
Model Update Mechanism	Static file replacement	Manual file updates on each workstation	Centralized server deployment
Clinical Integration Complexity	Low (web-based workflow integration)	High (local IT infrastructure required on all workstations)	Low (web-based workflow integration)
scalability	Limited by client device resources	Limited by individual workstation resources	High (server resources can be scaled)
Regulatory Compliance	Simplified (no data transmission, reduced GDPR/HIPAA complexity)	Simplified (local processing, standard workstation compliance)	Complex (data transmission agreements, server security compliance)

4. Discussion

Our work demonstrates the feasibility of deploying ML and DL models entirely within the client browser, using technologies such as TensorFlow.js, ONNX Runtime Web and direct JS implementations. By shifting model execution from the server-side (e.g., using frameworks like Flask or Django) to the client device, we eliminate data transmission entirely during the inference phase while providing significant deployment and usability advantages.

A critical consideration for client-side deployment is understanding where it provides the greatest value relative to existing alternatives. Healthcare ML applications commonly appear to operate within university networks and critical infrastructure hospital environments [19] using locally installed systems with Python scikit-learn, R or containerized solutions. However, client-side browser deployment offers complementary advantages even in these environments.

While local installations using Python or R frameworks provide excellent computational performance and are commonly used in research environments, they require significant IT infrastructure, ongoing maintenance and specialized technical expertise. Users must install Python or R software, manage library dependencies, handle version conflicts and maintain model file updates across individual machines. Container-based solutions using Docker offer improved isolation but necessitate container orchestration knowledge, dedicated hosting infrastructure and specialized IT staff to manage container runtime environments, networking configurations and security updates.

Browser-based deployment offers several unique advantages: (1) Zero installation requirements – users can access applications immediately through standard web browsers without software installation, IT approval processes or dependency management; (2) Cross-platform compatibility – applications function identically across Windows, macOS, Linux and mobile platforms without platform-specific modifications; (3) Integration with web-based health systems – seamless integration with existing web-based electronic health records and clinical workflows through standard web technologies; (4) Regulatory simplification – data never leaves the user's device, simplifying compliance with data protection regulations regardless of consent status or institutional agreements.

Browser-based deployment proves most valuable in scenarios requiring rapid deployment without IT infrastructure setup, integration with web-based clinical systems or additional privacy layers beyond traditional consent mechanisms. In contrast to university networks and critical infrastructure hospital environments [19] where local installation may be preferred for maximum computational performance, browser-based deployment excels in outpatient settings, patient self-monitoring applications, multi-institutional collaborations and emergency medical situations.

The practical success of healthcare ML deployment depends heavily on user acceptance and integration with existing clinical workflows. Browser-based deployment provides immediate accessibility and consistent behavior across different devices without requiring specialized software training or IT support. Loading times remained within acceptable ranges for clinical decision-making: equation-based models (e.g. 0.2ms total prediction time) provided instantaneous feedback comparable to traditional clinical calculators, medium-complexity models (557ms total time including model loading) remained acceptable for clinical decisions, while complex deep learning models (148ms inference time) exceeded expectations for real-time clinical use. These response times align with clinician expectations for decision support tools and suggest that our browser-based implementations would be well-received in practical clinical environments.

Modern browsers implement privacy protection through process isolation, same-origin policy enforcement and automatic memory management that provides security isolation comparable to Docker containers and virtual machines while requiring no specialized infrastructure setup. Our monitoring confirmed that after initial model loading, no network communication occurs during prediction phases, eliminating vulnerabilities such as man-in-the-middle attacks, DNS poisoning and server-side data breaches. This approach provides the security benefits of traditional sandboxing solutions while significantly reducing deployment complexity and maintenance overhead.

Testing across different device form factors using browser developer tools revealed excellent performance for equation-based models, with our random forest (ONNX) and OCNN models (TensorFlow.js) also demonstrating good performance on mobile devices due to their lightweight architectures.

Browser environments operate within computational resource limitations compared to dedicated servers or high-performance local installations. While modern desktop computers provide sufficient resources for our deployed model complexities, mobile devices may encounter performance bottlenecks for much larger or more complex architectures. Very large models such as transformer-based language models (e.g., BERT, GPT variants), deep convolutional neural networks with hundreds of layers or ensemble models requiring extensive memory allocation may be unsuitable for mobile browser deployment due to weaker CPUs, thermal throttling and restricted memory capacities.

Client-side deployment depends on consistent web standards implementation across browsers. While our testing across Chrome, Firefox and Edge demonstrated consistent functionality, older browsers or restrictive institutional configurations may lack required capabilities. Network bandwidth constraints restrict model size – while our largest model (101.93KB) loaded efficiently, significantly larger models (>10MB) may encounter loading delays, particularly on mobile devices or slow networks. Additionally, browsers cannot match the computational performance of specialized ML hardware such as dedicated Graphics Processing Units (GPU) or Tensor Processing Units (TPU), limiting deployment of very large language models or computationally intensive healthcare AI applications.

Advanced WebAssembly implementations promise substantial performance improvements while maintaining security isolation and cross-platform compatibility. Implementing adaptive model complexity selection based on real-time device capability detection could optimize performance across diverse hardware environments, automatically selecting appropriate model complexity levels based on detected device capabilities and user preferences.

This work focuses on implementing the discussed models rather than evaluating clinical utility. While the Zacharias et al. equation-based model (specifically Z6) [30] is an externally validated risk equation for predicting kidney failure recommended by KDIGO in their Clinical Practice Guideline for CKD Evaluation and Management [26], we are not evaluating the clinical utility of the Halder et al. random forest and Mondol et al. OCNN implementations in real-world settings. While our work demonstrates technical feasibility and performance characteristics for browser-based deployment, broader clinical validation of the RF and OCNN models remains necessary for widespread adoption. Future work should include prospective clinical studies comparing browser-based prediction accuracy and clinical workflow integration with traditional decision-making processes, assessing impact on clinical efficiency, user satisfaction and patient outcomes across different clinical specialties.

5. Conclusion

This study demonstrates the feasibility and practical advantages of deploying healthcare ML models entirely within client browsers, eliminating data transmission during inference while maintaining clinical accuracy across different model complexities.

Our key findings include:

1. **Cross-Platform Deployment Success:** Browser-based implementation successfully deployed three kidney disease prediction models of varying complexity (equation-based, Random Forest and deep learning) with identical performance to original implementations, requiring no software installation or IT infrastructure setup.
2. **Complete Privacy Protection:** Network monitoring confirmed zero external data transmission during prediction phases across all implementations, ensuring patient data never leaves the user's device.
3. **Clinical Performance Acceptability:** Performance testing revealed response times ranging from 0.2ms for equation-based models to 148ms for complex deep learning models for real-time clinical decision support scenarios.
4. **Deployment Accessibility Advantages:** Compared to traditional local deployments requiring programming expertise and IT infrastructure, browser-based deployment offers immediate accessibility through familiar web interfaces, eliminating technical barriers for clinical adoption.

Declarations

Ethical vote: Not applicable

Conflict of interest: The authors declare that they have no competing interests.

Authors contributions: YANN: Conceptualization, methodology, software implementation, writing – original draft preparation and visualization; JD: Conceptualization, methodology, review, editing and supervision; JR contributed to the deployment of linear models; HUZ, RKU and AU provided the models; all authors approved the manuscript; LE, RK, MA, UTS, FK, HUZ and GK critical review of manuscript.

Funding: This work was supported by the German Federal Ministry of Education and Research (BMBF) within the framework of the e:Med research and funding concept (grant numbers: 01ZX1912A, 01ZX1912B, 01ZX1912C, and 01ZX1912D). The work of UTS was also supported by the Collaborative Research Center (Sonderforschungsbereich, SFB) 1453 (Nephrogenetics; project ID: 431984000). Funding for open access charge: Open Access Publication Funds, Göttingen University.

References

[1] Ahsan M, Khan A, Khan KR, Sinha BB, Sharma A. Advancements in medical diagnosis and treatment through machine learning: A review. Expert systems. 2024 Mar;41(3):e13499, doi: 10.1111/exsy.13499

[2] Kabir R, Syed HZ, Vinnakota D, Sivasubramanian M, Hitch G, Okello SA, Pulikkottil AT, Mahmud I, Dehghani L, Parsa AD. Deep learning for clinical decision-making and improved healthcare outcome. InDeep Learning in Personalized Healthcare and Decision Support 2023 Jan 1 (pp. 187-201). Academic Press, doi: 10.1016/B978-0-443-19413-9.00004-7.

[3] Grinberg M. Flask web development. Sebastopol (CA): O'Reilly Media, Inc.; 2018.

[4] Forcier J, Bissex P, Chun WJ. Python web development with Django. Upper Saddle River (NJ): Addison-Wesley Professional; 2008.

[5] Ahmed N, Ahammed R, Islam MM, Uddin MA, Akhter A, Talukder MA, Paul BK. Machine learning based diabetes prediction and development of smart web application. International Journal of Cognitive Computing in Engineering. 2021 Jun 1;2:229-41, doi: 10.1016/j.ijcce.2021.12.001.

[6] Zhang P, Wang R, Shi N. IgA nephropathy prediction in children with machine learning algorithms. Future Internet. 2020 Dec 17;12(12):230, doi: 10.3390/fi12120230.

[7] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Article 28. Official Journal of the European Union. 2016 May 4;L 119:1-88.

[8] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Article 6. Official Journal of the European Union. 2016 May 4;L 119:1-88.

[9] Xia L, Cao Z, Zhao Y. Paradigm Transformation of Global Health Data Regulation: Challenges in Governance and Human Rights Protection of Cross-Border Data Flows. Risk Management and Healthcare Policy. 2024 Dec 31:3291-304, doi: 10.2147/RMHP.S450082.

[10] Dickert NW, Brown J, Cairns CB, Eaves-Leanos A, Goldkind SF, Kim SY, Nichol G, O'Conor KJ, Scott JD, Sinert R, Wendler D. Confronting ethical and regulatory challenges of emergency care research with conscious patients. Annals of emergency medicine. 2016 Apr 1;67(4):538-45, doi: 10.1016/j.annemergmed.2015.10.026

[11] Odebrecht C. Research Data Governance. The Need for a System of Cross-organisational Responsibility for the Researcher's Data Domain. Data Science Journal. 2025 Apr 11;24, doi: 10.5334/dsj-2025-012

[12] El Emam K, Jonker E, Arbuckle L, Malin B. A systematic review of re-identification attacks on health data. PloS one. 2011 Dec 2;6(12):e28071, doi: 10.1371/journal.pone.0028071.

[13] Li L, Fan Y, Tse M, Lin KY. A review of applications in federated learning. Computers & Industrial Engineering. 2020 Nov 1;149:106854, doi: 10.1016/j.cie.2020.106854.

[14] Mohassel P, Zhang Y. Secureml: A system for scalable privacy-preserving machine learning. In2017 IEEE symposium on security and privacy (SP) 2017 May 22 (pp. 19-38). IEEE, doi: 10.1109/SP.2017.12

[15] Pulido-Gaytan B, Tchernykh A, Cortés-Mendoza JM, Babenko M, Radchenko G, Avetisyan A, Drozdov AY. Privacy-preserving neural networks with homomorphic encryption: C hallenges and opportunities. Peer-to-Peer Networking and Applications. 2021 May;14(3):1666-91, doi:10.1007/s12083-021-01076-8.

[16] Vepakomma P, Gupta O, Swedish T, Raskar R. Split learning for health: Distributed deep learning without sharing raw patient data. arXiv preprint arXiv:1812.00564. 2018 Dec 3, doi: 10.48550/arXiv.1812.00564

[17] Geppert T, Deml S, Sturzenegger D, Ebert N. Trusted execution environments: Applications and organizational challenges. Frontiers in Computer Science. 2022 Jul 7;4:930741, doi: 10.3389/fcomp.2022.930741

[18] Hernandez-Matamoros A, Kikuchi H. Comparative Analysis of Local Differential Privacy Schemes in Healthcare Datasets. Applied Sciences. 2024 Mar 28;14(7):2864, doi: 10.3390/app14072864.

[19] Bundesamt für Sicherheit in der Informationstechnik. KRITIS - Kritische Infrastrukturen [Internet]. Bonn: BSI; [cited 2025 Jun 18]. Available from: https://www.bsi.bund.de/dok/kritis

[20] Smilkov D, Thorat N, Assogba Y, Nicholson C, Kreeger N, Yu P, Cai S, Nielsen E, Soegel D, Bileschi S, Terry M. Tensorflow. js: Machine learning for the web and beyond. Proceedings of Machine Learning and Systems. 2019 Apr 15;1:309-21.

[21] Brain.js: GPU accelerated neural networks in JavaScript for browsers and Node.js [Internet]. 2025 [cited 2025 Mar 7]. Available from: https://brain.js.org/#/

[22] ONNX Runtime Web [Internet]. 2025 [cited 2025 Mar 10]. Available from: https://onnxruntime.ai/docs/tutorials/web/

[23] Hidaka M, Kikura Y, Ushiku Y, Harada T. Webdnn: Fastest dnn execution framework on web browser. InProceedings of the 25th ACM international conference on Multimedia 2017 Oct 19 (pp. 1213-1216), doi: 10.1145/3123266.3129394.

[24] Goh HA, Ho CK, Abas FS. Front-end deep learning web apps development and deployment: a review. Applied intelligence. 2023 Jun;53(12):15923-45, doi: 10.1007/s10489-022-04278-6.

[25] Francis A, Harhay MN, Ong AC, Tummalapalli SL, Ortiz A, Fogo AB, Fliser D, Roy-Chaudhury P, Fontana M, Nangaku M, Wanner C. Chronic kidney disease and the global public health agenda: an international consensus. Nature Reviews Nephrology. 2024 Jul;20(7):473-85, doi: 10.1038/s41581-024-00820-6

[26] Stevens PE, Ahmed SB, Carrero JJ, Foster B, Francis A, Hall RK, Herrington WG, Hill G, Inker LA, Kazancıoğlu R, Lamb E. KDIGO 2024 clinical practice guideline for the evaluation and management of chronic kidney disease. Kidney international. 2024 Apr 1;105(4):S117-314, doi: 10.1016/j.kint.2023.10.018

[27] Kotsis F, Bächle H, Altenbuchinger M, Dönitz J, Njipouombe Nsangou YA, Meiselbach H, Kosch R, Salloch S, Bratan T, Zacharias HU, Schultheiss UT. Expectation of clinical decision support systems: a

survey study among nephrologist end-users. BMC Medical Informatics and Decision Making. 2023 Oct 26;23(1):239, doi: 10.1186/s12911-023-02317-x

[28] Bostock M. D3.js – Data-Driven Documents [Internet]. 2012 [cited 2025 Mar 7]. Available from: https://d3js.org/

[29] Reese W. Nginx: the high-performance web server and reverse proxy. Linux Journal. 2008 Sep 1;2008(173):2.

[30] Zacharias HU, Altenbuchinger M, Schultheiss UT, Raffler J, Kotsis F, Ghasemi S, Ali I, Kollerits B, Metzger M, Steinbrenner I, Sekula P. A predictive model for progression of CKD to kidney failure based on routine laboratory tests. American Journal of Kidney Diseases. 2022 Feb 1;79(2):217-30, doi: 10.1053/j.ajkd.2021.05.018

[31] Halder RK, Uddin MN, Uddin MA, Aryal S, Saha S, Hossen R, Ahmed S, Rony MA, Akter MF. ML-CKDP: Machine learning-based chronic kidney disease prediction with smart web application. Journal of Pathology Informatics. 2024 Dec 1;15:100371, doi: 10.1016/j.jpi.2024.100371.

[32] Mondol C, Shamrat FJ, Hasan MR, Alam S, Ghosh P, Tasnim Z, Ahmed K, Bui FM, Ibrahim SM. Early prediction of chronic kidney disease: A comprehensive performance analysis of deep learning models. Algorithms. 2022 Aug 29;15(9):308, doi: 10.3390/a15090308.

[33] sklearn-onnx: Convert your scikit-learn model into ONNX [Internet]. 2025 [cited 2025 Mar 14]. Available from: https://onnx.ai/sklearn-onnx/