MDPI

Article

iMediFood-Shield: Secure Edge AI for Food and Medication Interaction Screening

Sai Sri Harsha Chakravarthula [1,*], Indira Devi Siripurapu [1,*], Laavanya Rachakonda [2*], Saraju P. Mohanty [1], and Elias Kougianos [3]

[1] Department of Computer Science and Engineering, University of North Texas, Denton, TX 76205, USA; SaiSriHarshaChakravarthula@my.unt.edu; IndiraDeviSiripurapu@my.unt.edu; smohanty@ieee.org
[2] Department of Computer Science and Engineering, University of North Carolina Wilmington, Wilmington, NC, USA; rachakondal@uncw.edu
[3] Department of Electrical Engineering, University of North Texas, Denton, TX 76205, USA; elias.kougianos@unt.edu
* Correspondence: SaiSriHarshaChakravarthula@my.unt.edu (S.S.H.C.); IndiraDeviSiripurapu@my.unt.edu (I.D.S.); rachakondal@uncw.edu (L.R.)

Abstract

Food-medication interactions are an important daily health concern because medications are often taken alongside foods, drinks, herbs, or supplements that may change how a medication works. Missing an interaction can create a false sense of safety, especially when a screening system shows a risky pair as having no known interaction. The risk becomes more serious in edge-based screening because the evidence table, AI model, policy file, prescription input, or displayed recommendation can be modified or tampered with. These concerns motivate iMediFood-Shield, a secure edge AI framework for food and medication interaction screening. The framework combines evidence-based lookup, safety-gated AI prediction, prescription text extraction with user confirmation, input coverage checking, and tamper-evident verification of protected files and recommendation outputs. The signed screening workflow and tamper-evident verification layer were also implemented and validated on a Raspberry Pi edge device to evaluate practical edge deployment. In the DDID-based evaluation, the balanced AI baseline achieved 91.86% accuracy with a 5.29% false-safe rate. The strict screening mode of MedSafe-GATE v1 achieved 90.99% accuracy while reducing the false-safe rate to 0.62%, providing the strongest safety-focused performance among the evaluated settings. The prototype security layer detected all implemented tampering attacks. On the Raspberry Pi, provenance verification completed in 167.79 ms on average for verified cases and 159.71 ms on average for failed-verification cases over 30 runs. The results show that reliable food–medication screening needs both accurate AI and security-aware edge verification, and that the proposed framework can support practical tamper-evident screening on resource-constrained edge hardware.

Keywords: Trustworthy AI; Secure edge AI; AI security; Internet of Medical Things; Food–medication interaction screening; Tamper-evident verification

Received:
Revised:
Accepted:
Published:

1. Introduction

Medication use is part of everyday life, but medications are rarely taken in isolation. A person may take a tablet, drink coffee or juice, eat meals, and use herbs, supplements, or over-the-counter products without considering possible interaction effects. In the United States, 49.9% of people used at least one prescription drug in the past 30 days during

2017–March 2020, and 24.7% used three or more prescription drugs during the same period [1]. Since food, drinks, herbs, and supplements are common in daily routines, food-medication interaction awareness is important for patients, older adults, caregivers, pharmacists, clinicians, and general users who need quick screening support.

Food-medication interactions can change how a medication works in the body. A drug interaction may occur when a drug is taken with certain foods, supplements, other drugs, or medical conditions [2]. Food–drug interactions can affect medication effectiveness, side effects, and medication administration decisions [3,4]. Grapefruit juice is a common example because its interaction effects can be drug-specific and clinically important [5, 6]. Food and herb interactions may also affect drug release, absorption, metabolism, bioavailability, toxicity, or therapeutic effect [7–9].

Figure 1. User Uncertainty Before Food–Medication Screening.

Figure 1 shows a user with a prescription and several food or meal options. The scenario represents uncertainty about which items may be risky with the medication.

These decisions often happen quickly and outside a clinical visit. If a risky pair is shown as having no known interaction, the output can give the user a false sense of safety. Missing food–drug and herb–drug interactions can lead to serious outcomes, including reduced therapeutic effect, toxic drug effects, and other severe clinical consequences [7, 10,11]. Therefore, false-safe outputs are especially important in medication-aware food recommendation systems.

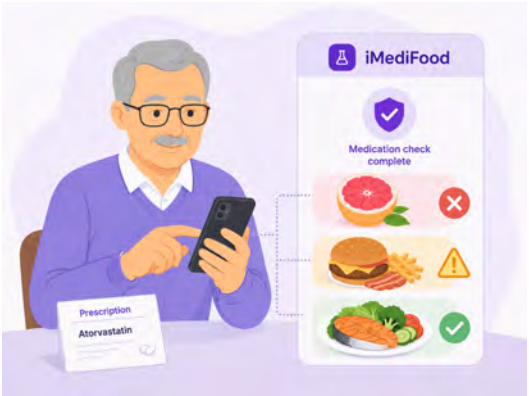

Figure 2. iMediFood-Shield-Assisted Food–Medication Screening.

Figure 2 shows how iMediFood-Shield screens selected food–medication pairs. The prototype displays avoid, caution, and no-known-interaction outcomes from the selected source.

AI-based screening can organize interaction evidence and support faster decision making, but accuracy alone is not enough. Machine learning has shown strong potential

in medicine, but clinical AI systems require careful validation, deployment planning, and safety-focused evaluation beyond technical accuracy [12,13]. The NIST Artificial Intelligence Risk Management Framework describes trustworthy AI using properties such as validity, reliability, safety, security, resilience, accountability, transparency, explainability, privacy enhancement, and fairness [14]. Security-focused AI studies in other domains, such as deepfake audio detection, also show the importance of reliable AI systems for trust-sensitive detection tasks [15]. In food–medication screening, the system must handle exact evidence, uncertain model predictions, missing coverage, prescription text errors, and conservative decision-making when false-safe outputs may be more harmful than over-cautious warnings.

Security is important because the final output depends on the evidence table, trained model, policy file, prescription input, logs, and displayed recommendation. If any of these assets are modified or rolled back, the system may still appear normal while showing an unsafe recommendation. In healthcare and Internet of Medical Things (IoMT) settings, such tampering can affect integrity, availability, privacy, and user safety. FDA guidance emphasizes secure design and cyber-resilient systems [16], while prior medical-device research shows the risk of safety-relevant attack surfaces when security is not built into the design [17]. Hash-based integrity checking and HMAC authentication provide established ways to detect unauthorized changes to protected files and signed records [18,19].

Edge deployment increases this risk because evidence resources, model files, policy files, logs, and recommendation outputs are stored locally. For a food–medication screener, changing one protected file or displayed recommendation may be enough to turn a cautionary result into a false-safe result. Therefore, edge-based screening needs security-aware verification along with AI prediction [14,16].

The Diet–Drug Interaction Database (DDID) provides a curated starting point for diet–drug interaction evidence, including drugs, foods, herbs, interaction effects, and interaction records [20,21]. However, a database alone is not enough. A practical system must decide when to use exact evidence, when to apply AI prediction, when to reject unsupported inputs, and how to verify that the evidence, model, policy, and displayed recommendation have not been tampered with.

These challenges motivate iMediFood-Shield, a secure edge AI framework for food and medication interaction screening. The framework combines evidence-aware screening, safety-gated AI prediction, prescription input confirmation, input coverage control, and tamper-evident verification. This is a research prototype and is not clinical decision support. Always confirm recommendations with a licensed healthcare professional.

2. Related Work

Research related to iMediFood-Shield spans food logging, dietary assessment, medication management, diet–drug interaction resources, edge-AI healthcare systems, and healthcare cybersecurity. Earlier IoMT-based healthcare systems have shown how edge devices and deep neural networks can support smart-health monitoring tasks such as stress-level detection [22]. Related IoMT-based food monitoring systems such as iLog focused on automatic food-intake monitoring and stress-related eating behavior [23]. More recent diet-management systems have also explored computer-vision-based dietary assessment for smart healthcare applications [24]. These works provide an important foundation for automated health and food monitoring, but their main goal is physiological or dietary tracking rather than medication-aware food safety screening.

Food and nutrition AI systems mainly focus on estimating calories, nutrients, food category, or food volume from user input or food images. For example, iLog 2.1 studied calorie estimation for pizza using Mask R-CNN and federated learning [25]. iLog 3.0

extended this direction by estimating food volume from 2D images using Mask R-CNN and monocular depth estimation [26]. Similarly, iLog 2.2 addressed volume and nutrition estimation for mixed foods [27]. These systems are useful for nutrition monitoring, but they do not directly check whether a selected food, herb, or drink may interact with a patient's medication.

Diet–drug and herb–drug interaction research provides the clinical foundation for this problem. Bushra et al. discussed general food–drug interaction mechanisms and their relevance to drug response [3]. Ased et al. further reviewed clinically significant food–drug interactions that may affect patient safety [4]. Amadi and Mgbahurike focused on selected food/herb–drug interactions and explained their mechanisms and clinical relevance [7]. These studies show that food and herbal products can change drug absorption, metabolism, or therapeutic effect, which makes medication-aware food screening important.

Several works and resources have also focused specifically on herbal medicine and structured diet–drug interaction knowledge. Izzo and Ernst reviewed interactions between herbal medicines and prescribed drugs [8], while Posadzki et al. summarized systematic reviews on herb–drug interactions [10]. More recently, DDID was introduced as a comprehensive resource for visualization and analysis of diet–drug interactions [21]. In this work, DDID is important because it provides structured diet–drug interaction evidence that can support food/herb/drink coverage control in the proposed screening pipeline.

Interaction evidence management has also been studied in the context of clinical decision support. Scheife et al. provided consensus recommendations for evaluating drug–drug interaction evidence for clinical decision support [28]. Grizzle et al. studied how drug interaction experts search for potential drug–drug interaction evidence [29]. Hochheiser et al. proposed a minimal information model for describing potential drug–drug interactions, showing the need for structured and shareable interaction evidence [30]. These works support the need for evidence-aware and traceable interaction screening rather than model-only prediction.

Medication-management applications address a different part of the problem. Tools such as Medisafe and MyTherapy mainly support medication reminders, dose schedules, adherence tracking, refill alerts, and medication-list management [31,32]. Nutrition applications such as MyFitnessPal, Cronometer, and Lose It! mainly focus on calorie counting, nutrient tracking, diet goals, and weight management [33–35]. These tools are useful in their own categories, but they generally separate nutrition tracking from medication-aware interaction screening.

Standalone interaction checkers provide another useful category of existing tools. Drugs.com and Medscape provide drug interaction lookup functions that can help users identify unsafe drug, food, alcohol, or supplement combinations [36,37]. Prior work has also evaluated consumer mobile applications for potential drug–drug interaction checking, showing that mobile interaction tools can support patient empowerment but still vary in quality, content, and functionality [38]. However, these tools are generally lookup-based systems rather than integrated edge-AI frameworks. They do not combine prescription confirmation, DDID-based food/herb/drink selection, AI-supported prediction, false-safe reduction, and tamper-evident recommendation verification in one pipeline.

Security-focused healthcare research shows why integrity and verification are important in medical and IoMT systems. Halperin et al. demonstrated that medical devices such as pacemakers and implantable cardiac defibrillators can face serious security and privacy risks [17]. Recent IoMT security surveys also show that healthcare edge and IoMT systems face physical, network, privacy, and integrity threats, which supports the need for lightweight verification mechanisms in medical-facing edge systems [39]. Broader guidance from NIST and FDA also emphasizes trustworthy AI, cybersecurity, and risk

management for healthcare and medical-device systems [14,16]. These works support the need to treat medication-aware screening not only as an AI task, but also as a safety and security problem.

Tamper-evident verification is also relevant because medical recommendations should not be silently modified after generation. Secure hash standards and HMAC-based authentication provide established mechanisms for integrity checking and message authentication [18,19]. In iMediFood-Shield, these ideas motivate the edge-level verification layer, where the recommendation record and related artifacts can be checked for tampering before being trusted.

Table 1 summarizes representative research works related to food–drug interaction knowledge, food and nutrition AI, and healthcare security. These studies provide strong building blocks, including interaction reviews, curated diet–drug resources, food-intake monitoring, calorie estimation, food-volume estimation, diet management, and security analysis. Still, they do not directly combine medication-aware diet–drug screening with safety-gated AI decision-making and edge-level tamper verification.

Table 1. Comparison Of Research Works Related To Food–Medication Screening, Nutrition AI, And Healthcare Security.

Reference	Focus Area	Method/Resource	Missing Element
Bushra et al. [3]	Food–drug interaction	Interaction review	No AI screening or security layer.
Ased et al. [4]	Clinically significant food–drug interactions	Clinical review	No computational or edge-based screening framework.
Amadi and Mgbahurike [7]	Food/herb interaction	Clinical mechanism review	Not a computational screening system.
Izzo and Ernst [8]	Herb–drug interaction	Updated systematic review	No prescription-aware AI screening pipeline.
Posadzki et al. [10]	Herb–drug interaction	Review of systematic reviews	No integrated patient-facing decision system.
Hong et al. [21]	Diet–drug interaction data	DDID knowledge base	No integrated AI prediction, prescription confirmation, or tamper verification.
Rachakonda et al. [23]	Food monitoring	IoMT food logging	No medication interaction screening.
Siripurapu et al. [25]	Pizza nutrition	Mask R-CNN and FL	No medication-aware risk screening.
Siripurapu et al. [26]	Food volume	Mask R-CNN and depth	No interaction or security check.
Siripurapu et al. [27]	Mixed-food nutrition	Mask R-CNN and FL	No food–medication risk screening.
Veeramreddy et al. [24]	Diet management	Food recognition	No secure medication-aware screening.
Halperin et al. [17]	Medical security	Wireless attack analysis	No food–medication AI screening.

Table 2 summarizes representative commercial and consumer-facing tools. Overall, existing applications show a clear separation between nutrition tracking, medication tracking, and interaction lookup. The remaining gap is the lack of an integrated frame-

Table 2. Comparison Of Application-Level Tools Related To Food Logging, Medication Tracking, And Interaction Lookup.

Application/Tool	Primary Use	Typical Functionality	Missing Element
MyFitnessPal [33]	Food and fitness	Calories and macros	No food–medication interaction screening.
Cronometer [34]	Nutrition tracking	Macro and micronutrient logs	No medication-aware interaction check.
Lose It! [35]	Weight loss	Calorie and fasting logs	Focuses on diet goals, not medication safety.
Medisafe [31]	Medication tracking	Pill reminders and adherence	No food-based medication risk screening.
MyTherapy [32]	Medication reminders	Logs and refill reminders	No secure food–medication screening.
Drugs.com Checker [36]	Interaction lookup	Drug, food, and alcohol checks	No edge AI, OCR, or signed output.
Medscape Checker [37]	Clinical lookup	Drug, food, and supplement checks	No model or evidence integrity verification.

work that treats food–medication screening as both an AI safety problem and a security problem. iMediFood-Shield addresses this gap by combining evidence-aware food and medication screening, safety-gated AI prediction, prescription input confirmation, DDID-based food/herb/drink coverage control, false-safe reduction, and tamper-evident edge verification within a single screening pipeline.

3. Novelty of the Proposed Solution

Motivated by the gaps identified in the research and application comparisons, iMediFood-Shield introduces a security-centered edge AI framework for food–medication interaction screening. Existing nutrition systems primarily address food logging, calorie estimation, nutrient analysis, or diet management, while medication applications usually focus on reminders, adherence tracking, and medication schedules. Interaction checkers provide useful lookup support, but they are generally separated from AI-based screening, prescription input confirmation, coverage-aware decision control, edge deployment, and recommendation integrity verification. As a result, current solutions do not fully address the combined risk of false-safe AI outputs and tampered healthcare-facing recommendations.

The novelty of iMediFood-Shield is its integrated treatment of food–medication screening as an AI reliability, safety, and security problem. The framework is designed not only to predict interaction risk, but also to protect the evidence source, model artifact, policy logic, prescription-derived input, and displayed recommendation from integrity attacks. The prototype is further validated on Raspberry Pi hardware to show that tamper-evident verification can be executed on a practical resource-constrained edge device. This creates an end-to-end trustworthy screening pipeline where AI prediction, evidence grounding, input validation, edge execution, and tamper-evident verification operate together. The major contributions are summarized as follows:

- Security-aware food–medication screening framework: iMediFood-Shield advances beyond conventional food logging, medication tracking, and interaction lookup by combining screening intelligence with security verification. The framework treats recommendation integrity as a core design requirement rather than an optional post-processing step.

- DDID-based evidence grounding with safety-priority aggregation: iMediFood-Shield forms a pair-level screening dataset from the Diet–Drug Interaction Database (DDID). Original interaction effects are mapped into practical screening labels, and conflicting records are resolved using a safety-priority rule. This prevents lower-risk labels from overriding higher-risk interaction evidence during dataset formation.
- Evidence-first decision policy: The framework prioritizes exact DDID evidence before applying AI prediction. When a medication and food/herb pair exists in the evidence table, the evidence-derived label is returned directly. This improves trustworthiness by grounding recommendations in curated source evidence whenever exact evidence is available.
- False-safe-aware AI screening: MedSafe-GATE v1 is designed around the safety-critical false-safe problem, where a risky pair is incorrectly shown as having no known interaction. Instead of only maximizing overall accuracy, the safety gate reduces uncertain no-interaction outputs by shifting them toward a cautionary recommendation when confidence is insufficient.
- Prescription input protection through RxOCR-Guard: The framework supports prescription image and PDF input while reducing the risk of OCR-driven medication errors. RxOCR-Guard extracts medication candidates and requires user confirmation before screening, preventing unverified prescription text from directly controlling the recommendation pipeline.
- Coverage-aware rejection of unsupported inputs: iMediFood-Shield checks whether the medication and food/herb inputs are covered by the DDID-derived vocabulary. If either input is outside the supported evidence space, the framework returns an insufficient-evidence response instead of forcing an unsupported AI prediction.
- Multi-layer tamper-evident verification: The security layer protects the evidence table, trained model file, inference policy, logs, and recommendation outputs using hash-based verification and signed records. This enables detection of evidence modification, model replacement, policy rollback, log tampering, and displayed recommendation manipulation.
- Raspberry Pi edge validation: The tamper-evident verification layer is implemented and tested on a Raspberry Pi edge device. This validates that provenance verification can run locally on resource-constrained hardware and supports the practical edge-AI goal of the framework.
- Attack-aware evaluation of recommendation integrity: iMediFood-Shield evaluates security behavior using simulated tampering attacks against protected assets and signed outputs. This connects AI screening performance with cybersecurity validation and demonstrates how recommendation integrity can be assessed in an edge healthcare pipeline.
- Integrated AI, security, and edge-performance assessment: The framework is evaluated using accuracy, false-safe rate, evidence lookup behavior, tamper detection, signing overhead, verification overhead, secured inference latency, and Raspberry Pi validation timing. This evaluation positions iMediFood-Shield as an integrated trustworthy edge-AI framework rather than a standalone classifier or lookup tool.

4. Methodology

The iMediFood-Shield methodology is organized as a secure food–medication interaction screening pipeline. The framework begins with a curated diet–drug interaction evidence source, forms screening-oriented labels, processes prescription input through OCR and user confirmation, applies evidence-aware AI screening, rejects unsupported inputs, and protects each recommendation through layered integrity verification. The

methodology connects prediction reliability with security because a model error and a tampered output can both create the same user-facing risk: an incorrect food–medication recommendation. Figure 3 summarizes the methodology flow, including evidence formation, prescription input, coverage checking, AI screening, security protection, Raspberry Pi deployment, attack modeling, and evaluation.

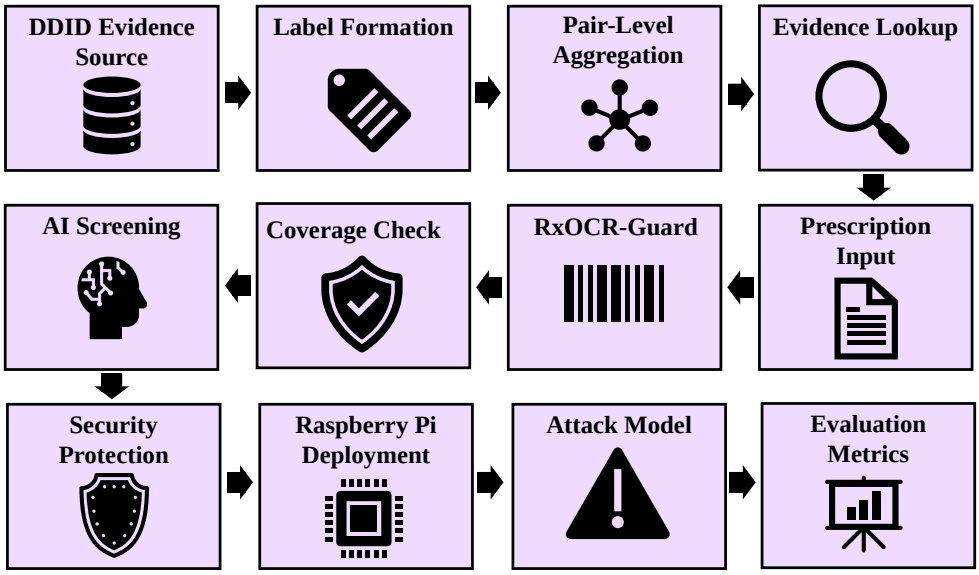

Figure 3. iMediFood-Shield Methodology Workflow From DDID Evidence Formation To Secure Edge Evaluation.

4.1. Dataset Source and Label Formation

The Diet–Drug Interaction Database (DDID) was used as the primary evidence source for food, herb, and medication interaction records [21]. DDID provides curated diet–drug interaction information, including drug names, food or herb names, interaction effects, and interaction records. The interaction information file was used as the main source for evidence table formation and AI model development.

The raw DDID records were cleaned by normalizing medication names, food/herb names, interaction types, and effect labels. Normalization was applied to reduce formatting differences between records and to support consistent lookup, vocabulary coverage checking, and model input formation. The original DDID effect labels were converted into three screening-oriented labels. Harmful interactions were mapped to "Avoid". Possible, positive, and negative interactions were mapped to "Use caution" because they indicate that an interaction effect exists or may exist. No-effect records were mapped to "No known interaction in selected source". This wording avoids claiming universal safety and only reflects the selected DDID source.

A pair-level dataset was then created because the same medication and food/herb pair can appear in more than one DDID record. When multiple records for the same pair produced different screening labels, a safety-priority aggregation rule was applied in the following order:

$$\text{Avoid} > \text{Use caution} > \text{No known interaction in selected source.}$$

This rule preserves the strongest available caution level for a medication and food/herb pair. The cleaned row-level dataset contained 23,950 interaction records, and the final pair-level dataset contained 17,693 unique medication and food/herb interaction pairs.

4.2. Evidence Lookup Layer

The pair-level dataset was converted into an evidence lookup table. Each lookup key was formed using the normalized medication name and normalized food/herb name. The lookup value stores the evidence-derived screening label generated from the safety-priority aggregation rule. This evidence lookup layer is used before AI inference so that exact DDID-supported pairs are grounded in curated evidence rather than replaced by a learned prediction.

When a user-selected medication and food/herb pair exactly matches the lookup table, iMediFood-Shield returns the evidence-derived label directly. This layer reduces unnecessary model dependence and helps maintain traceability between the recommendation and the selected evidence source. If no exact pair is found, the workflow proceeds to the input coverage and AI screening layers.

4.3. Prescription Input Layer with RxOCR-Guard

Prescription input can be provided as a PDF or image. Selectable-text PDFs are processed through direct text extraction. Scanned PDFs and image files are processed through optical character recognition (OCR). RxOCR-Guard applies image preprocessing before OCR, including grayscale conversion, enlargement, contrast adjustment, sharpening, and thresholding. These steps improve the readability of medication names before text recognition. Tesseract OCR was used as the OCR engine, supported by its current open-source documentation and software resources [40].

The extracted prescription text is not used automatically as the final medication input. RxOCR-Guard compares extracted text with the DDID-derived medication vocabulary and generates candidate medication names through exact and approximate text matching. Candidate names are then presented for user confirmation before screening. This step is important because prescription images may contain dosage values, instructions, provider names, patient names, dates, and other text that should not control the recommendation pipeline. Only the confirmed medication name is passed to the screening workflow.

4.4. Input Coverage Layer

The input coverage layer checks whether the medication and food/herb inputs are represented in the DDID-derived vocabulary. If either input is outside the covered vocabulary, iMediFood-Shield returns "Insufficient evidence" instead of forcing an AI prediction. This layer prevents unsupported recommendations for random, misspelled, or out-of-source entities.

The coverage layer acts as a safety boundary between supported screening cases and unsupported inputs. A model prediction is allowed only when both entities are recognized by the covered vocabulary and no exact DDID pair is available. This design reduces the chance that the model produces a confident-looking result for an input that was never represented in the evidence source.

4.5. AI Screening Layer

MedSafe-GATE v1 is the AI screening layer used in iMediFood-Shield. The model input is a text representation formed from the normalized medication name and normalized food/herb name. The text representation is converted into term frequency–inverse document frequency (TF-IDF) features using unigram and bigram terms. TF-IDF was selected because it provides an interpretable sparse representation for text-based medication and food/herb pair classification.

Several model settings were evaluated during development, including a majority-class baseline, logistic regression, balanced logistic regression, and balanced LinearSVC.

Balanced learning was included because class imbalance can strongly affect classifier behavior and minority-class recognition [41]. The balanced LinearSVC configuration provided the strongest baseline behavior among the evaluated model settings. MedSafe-GATE v1 was then designed as a safety-gated extension focused on reducing false-safe outputs.

A false-safe output occurs when a truly risky pair, labeled as "Avoid" or "Use caution", is predicted as "No known interaction in selected source". This error type is more safety-sensitive than a conservative warning because it can create a false sense of safety. MedSafe-GATE v1 therefore applies a screening gate to no-interaction predictions. If the confidence for "No known interaction in selected source" does not meet the selected threshold, the output is shifted toward a cautionary label instead of being displayed as no known interaction.

4.6. Security Methodology

The iMediFood-Shield security methodology protects the digital assets that influence the final recommendation. The protected assets include the DDID-derived evidence lookup table, trained model file, inference policy file, label mapping records, dataset summaries, recommendation logs, and signed recommendation outputs. The goal is to detect integrity violations that could change the recommendation while allowing the system to appear normal to the user.

The first security layer protects evidence integrity. The evidence lookup table is hashed using SHA-256, which is specified in the NIST Secure Hash Standard [18]. If an attacker modifies an interaction label, removes a risky pair, or replaces the evidence table, the recomputed hash differs from the protected manifest.

The second security layer protects model and policy integrity. The trained model artifact and inference policy file are included in the protected manifest. This layer is important because model replacement or policy rollback can change whether a pair is handled through exact evidence, AI prediction, coverage rejection, or cautionary gating.

The third security layer protects recommendation-output integrity. Each recommendation output is serialized with key decision information, including the medication input, food/herb input, normalized names, recommendation label, decision source, and relevant asset hashes. The record is signed using HMAC-SHA256. NIST describes HMAC as a keyed-hash message authentication code that supports detection of unauthorized data modification [19]. If the displayed recommendation is changed after generation, output verification fails.

The fourth security layer supports provenance-style verification. The signed manifest stores the expected hashes of protected files. During verification, the manifest signature is checked and protected file hashes are recomputed. A successful verification indicates that the protected prototype files match the signed state. A mismatch produces a failed verification status. The current implementation uses a software demonstration key, so the security layer is described as tamper-evident prototype verification rather than a complete hardware-rooted security system.

4.7. Edge Deployment Methodology

The Raspberry Pi deployment evaluates whether the iMediFood-Shield tamper-evident verification layer can run locally on a resource-constrained edge device. The edge validation focuses on security verification rather than clinical deployment. The protected files are transferred to the Raspberry Pi along with the signed security manifest and the software HMAC-based verification script. The device then recomputes protected-file

hashes, checks the signed manifest, and returns either *VERIFIED_PROVENANCE* for an unchanged trusted state or *FAILED_VERIFICATION* when tampering is detected.

The Raspberry Pi methodology includes two validation states. First, the unchanged protected assets are verified to confirm that the trusted local deployment can reproduce the signed manifest state. Second, a tampered copy of a protected asset is checked against the same signed manifest to confirm that the edge device detects the integrity mismatch. Verification timing is measured across repeated runs to evaluate whether local tamper-evident checking is practical for edge-based screening. This deployment uses a software demonstration key and does not claim hardware-rooted trust. This is not clinical decision support. Always confirm recommendations with a licensed healthcare professional.

4.8. Attack Model

The attack model focuses on integrity attacks that can alter the food–medication screening result or the trustworthiness of the displayed output. The evaluated attacks include evidence table tampering, model file tampering, policy modification or rollback, output log tampering, and displayed recommendation tampering. These attacks were selected because each one can influence the recommendation pipeline without necessarily preventing the application from running.

Each attack is applied after the trusted manifest or signed output has been generated. Verification is then executed to determine whether the modified asset is detected. The expected secure behavior is a failed verification status when any protected file or signed output record is changed. This attack model aligns with the broader concern that healthcare-facing AI systems require security, resilience, and integrity protection across the lifecycle [14,16].

4.9. Evaluation Methodology

The evaluation methodology includes AI screening behavior, false-safe behavior, evidence lookup behavior, security attack detection, runtime overhead, and Raspberry Pi edge-validation timing. AI performance is measured using accuracy and macro F1-score. False-safe rate is used as a safety-focused metric because overall accuracy alone may not capture the risk of showing a risky pair as having no known interaction.

This is consistent with prior work showing that standard aggregate metrics can be less informative under class imbalance and that task-specific error analysis is important for imbalanced classification settings [42].

Security behavior is evaluated through tamper detection and overhead measurement. Tamper detection is measured by applying the implemented attack simulations and checking whether verification changes from verified provenance to failed verification. Runtime overhead is measured for file hashing, manifest verification, recommendation signing, output verification, and secured inference.

Raspberry Pi edge validation is measured using repeated verification timing for verified-provenance and failed-verification states. This evaluates whether local tamper-evident checking is practical on resource-constrained edge hardware.

This combined evaluation allows iMediFood-Shield to be assessed as both an AI screening framework and a security-aware edge recommendation pipeline.

5. Implementation

This section presents the implemented iMediFood-Shield pipeline using mathematical definitions, algorithms, and security verification logic. The implementation follows the same order as the screening workflow: DDID-based dataset construction, prescription text extraction, input coverage checking, MedSafe-GATE v1 inference, recommendation

signing, and tamper-evident verification. The equations define the computational logic of the framework, while the algorithms summarize how the steps are executed in the prototype.

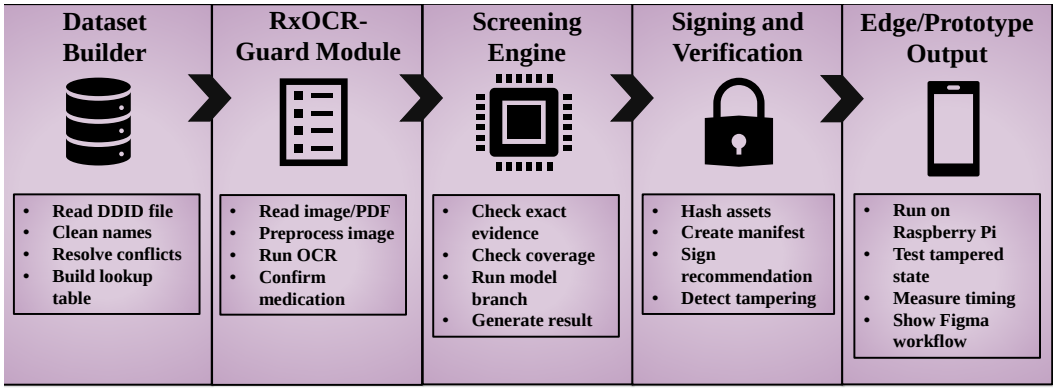

Figure 4. iMediFood-Shield Implementation Workflow Showing the Main Prototype Modules.

Figure 4 summarizes the implemented pipeline, including DDID dataset construction, RxOCR-Guard prescription processing, evidence-aware screening, recommendation signing, tamper detection, Raspberry Pi execution, and Figma prototype output.

5.1. Notation and Screening Classes

Each raw DDID interaction record is represented as a tuple, as shown in Eq. (1). This representation keeps the medication, food/herb entity, interaction type, effect label, and source evidence together before cleaning.

$$r_i = (d_i, f_i, t_i, e_i, s_i), \tag{1}$$

where r_i is the ith raw interaction record, d_i is the medication name, f_i is the food or herb name, t_i is the interaction type, e_i is the original DDID effect label, and s_i denotes supporting source information available in the interaction record.

The screening output space contains three labels, as defined in Eq. (2). These labels are used consistently in dataset formation, model training, inference, and final recommendation generation.

$$\mathcal{C} = \{c_A, c_C, c_N\}, \tag{2}$$

where c_A denotes "Avoid", c_C denotes "Use caution", and c_N denotes "No known interaction in selected source". The label c_N is intentionally source-bounded because the framework does not claim that an interaction is impossible outside the selected evidence source.

Text normalization is applied before lookup construction and model training. As shown in Eq. (3), the normalization function standardizes input strings so that the same medication or food/herb name is represented consistently across the pipeline.

$$\eta(x) = \text{clean}(\text{lowercase}(\text{strip}(x))), \tag{3}$$

where x is an input string, $\text{strip}(\cdot)$ removes leading and trailing spaces, $\text{lowercase}(\cdot)$ standardizes letter casing, and $\text{clean}(\cdot)$ removes formatting noise. This normalized representation is used for evidence lookup keys, vocabulary coverage checking, OCR candidate matching, and model input construction.

5.2. DDID Label Mapping

The original DDID effect labels are converted into screening-oriented labels using the mapping rule in Eq. (4). This step translates database labels into user-facing screening categories.

$$\phi(e_i) = \begin{cases} c_A, & \text{if } e_i = \text{Harmful}, \\ c_C, & \text{if } e_i \in \{\text{Possible}, \text{possible}, \text{Positive}, \text{Negative}\}, \\ c_N, & \text{if } e_i = \text{No Effect}. \end{cases} \tag{4}$$

In Eq. (4), harmful records are mapped to the strongest warning class. Possible, positive, and negative records are mapped to "Use caution" because they indicate that an interaction effect exists or may exist. No-effect records are mapped to "No known interaction in selected source" to avoid overstating safety beyond the selected DDID evidence.

After label mapping, each cleaned row-level record is represented as shown in Eq. (5). This record becomes the basic unit for pair-level aggregation.

$$\tilde{r}_i = (\tilde{d}_i, \tilde{f}_i, \tilde{t}_i, y_i), \tag{5}$$

where $\tilde{d}_i = \eta(d_i)$, $\tilde{f}_i = \eta(f_i)$, $\tilde{t}_i = \eta(t_i)$, and $y_i = \phi(e_i)$. The full cleaned row-level dataset is defined in Eq. (6).

$$\mathcal{D}_{row} = \{\tilde{r}_i\}_{i=1}^{N}, \tag{6}$$

where $\mathcal{D}_{row}$ is the cleaned row-level dataset and $N = 23{,}950$ is the number of cleaned DDID interaction records used in the implemented pipeline.

5.3. Safety-Priority Pair-Level Aggregation

The same medication and food/herb pair may appear in multiple DDID rows. Therefore, each row-level record is first converted into a normalized pair key, as shown in Eq. (7).

$$k_i = (\tilde{d}_i, \tilde{f}_i), \tag{7}$$

where k_i is the normalized medication–food/herb pair key. For a given pair key k, all row-level labels associated with that pair are collected using Eq. (8).

$$\mathcal{Y}_k = \{y_i \mid (\tilde{d}_i, \tilde{f}_i) = k\}. \tag{8}$$

Here, $\mathcal{Y}_k$ is the set of mapped screening labels assigned to the same medication–food/herb pair. When conflicting labels exist, iMediFood-Shield uses the safety-priority score in Eq. (9).

$$\rho(y) = \begin{cases} 3, & \text{if } y = c_A, \\ 2, & \text{if } y = c_C, \\ 1, & \text{if } y = c_N. \end{cases} \tag{9}$$

In Eq. (9), larger values represent stronger safety warnings. The final pair-level label is selected using the maximum safety-priority value, as shown in Eq. (10).

$$Y_k = \arg\max_{y \in \mathcal{Y}_k} \rho(y). \tag{10}$$

Eq. (10) prevents a lower-risk record from overwriting a higher-risk record for the same pair. The resulting pair-level dataset is defined in Eq. (11).

$$\mathcal{D}_{pair} = \{(k_j, Y_{k_j})\}_{j=1}^{M}, \tag{11}$$

where $\mathcal{D}_{pair}$ is the pair-level dataset, k_j is the jth unique medication–food/herb pair, Y_{k_j} is its final aggregated screening label, and $M = 17{,}693$ is the number of unique pairs.

Algorithm 1 summarizes the implemented dataset formation process. The algorithm follows the flow described in Eq. (1)–Eq. (11): raw records are normalized, labels are mapped, pair keys are formed, and final labels are selected using safety-priority aggregation.

Algorithm 1 DDID-Based Pair-Level Dataset Formation

Require: Raw DDID interaction table $\mathcal{D}_{raw}$
Ensure: Pair-level screening dataset $\mathcal{D}_{pair}$
Initialize $\mathcal{D}_{row} \leftarrow \emptyset$
for each record $r_i = (d_i, f_i, t_i, e_i, s_i)$ in $\mathcal{D}_{raw}$ **do**
    Normalize fields using Eq. (3)
    Map effect label $y_i \leftarrow \phi(e_i)$ using Eq. (4)
    Create cleaned record $\tilde{r}_i$ using Eq. (5)
    Add $\tilde{r}_i$ to $\mathcal{D}_{row}$
end for
Initialize $\mathcal{D}_{pair} \leftarrow \emptyset$
for each unique pair key k in $\mathcal{D}_{row}$ **do**
    Collect label set $\mathcal{Y}_k$ using Eq. (8)
    Select aggregated label Y_k using Eq. (10)
    Add (k, Y_k) to $\mathcal{D}_{pair}$
end for
return $\mathcal{D}_{pair}$

5.4. Evidence Lookup and Coverage Guard

The pair-level dataset is converted into an evidence lookup table. The lookup rule is shown in Eq. (12) and allows exact DDID-supported evidence to be used before AI inference.

$$\mathcal{L}[(\tilde{d}, \tilde{f})] = Y_{(\tilde{d},\tilde{f})}. \tag{12}$$

In Eq. (12), $\mathcal{L}$ is the evidence lookup table, $(\tilde{d}, \tilde{f})$ is the normalized medication–food/herb key, and $Y_{(\tilde{d},\tilde{f})}$ is the final safety-priority label for that pair. This lookup layer is placed before the model so that exact curated evidence is not replaced by a learned prediction.

The supported medication vocabulary is defined in Eq. (13), and the supported food/herb vocabulary is defined in Eq. (14). These vocabularies form the input coverage boundary of the system.

$$\mathcal{V}_d = \{\tilde{d} \mid (\tilde{d}, \tilde{f}) \in \mathcal{D}_{pair}\}, \tag{13}$$

$$\mathcal{V}_f = \{\tilde{f} \mid (\tilde{d}, \tilde{f}) \in \mathcal{D}_{pair}\}. \tag{14}$$

In Eq. (13), $\mathcal{V}_d$ contains all medications covered by the pair-level dataset. In Eq. (14), $\mathcal{V}_f$ contains all food and herb entities covered by the same dataset. If either user input is outside these vocabularies, iMediFood-Shield returns an insufficient-evidence response instead of forcing a model prediction.

5.5. RxOCR-Guard Prescription Processing

RxOCR-Guard allows prescription images and PDFs to be used as medication input. The text extraction route is selected using Eq. (15).

$$T_P = \begin{cases} \text{TextExtract}(P), & \text{if } P \text{ contains selectable text,} \\ \text{OCR}(\psi(P)), & \text{if } P \text{ is scanned or image-based.} \end{cases} \tag{15}$$

In Eq. (15), P is the uploaded prescription file, T_P is the extracted prescription text, and $\psi(P)$ is the preprocessed image representation used for OCR. The preprocessing sequence is defined in Eq. (16).

$$\psi(P) = \text{Threshold}(\text{Sharpen}(\text{Contrast}(\text{Resize}(\text{Gray}(P))))). \tag{16}$$

Eq. (16) represents the image enhancement flow used before OCR. Grayscale conversion reduces color variation, resizing improves character visibility, contrast adjustment and sharpening improve text edges, and thresholding separates text-like foreground from background.

After OCR, medication names are detected by comparing extracted text against the DDID-derived medication vocabulary. Approximate string matching is used to handle spelling variation and OCR noise, following the broader class of edit-distance and error-tolerant string matching methods [43]. The matching score is defined in Eq. (17).

$$S(q_m, v) = \max\Big(S_{exact}(q_m, v), S_{fuzzy}(q_m, v)\Big), \tag{17}$$

where q_m is a candidate text span extracted from T_P, v is a medication name from $\mathcal{V}_d$, $S_{exact}(\cdot)$ is the exact substring score, and $S_{fuzzy}(\cdot)$ is the approximate string-matching score. Candidate medications are selected using Eq. (18).

$$\mathcal{M}_P = \{v \in \mathcal{V}_d \mid S(q_m, v) \geq \tau_{ocr}\}. \tag{18}$$

In Eq. (18), $\mathcal{M}_P$ is the prescription-derived medication candidate set and τ_{ocr} is the OCR matching threshold. The candidate set is not used automatically. RxOCR-Guard requires the user to confirm the medication before screening, which reduces the risk of using dosage text, provider names, or noisy OCR fragments as medication input.

Algorithm 2 summarizes the prescription-processing flow. It connects Eq. (15)–Eq. (18) with the user-confirmation step.

Algorithm 2 RxOCR-Guard Prescription Input Confirmation

Require: Prescription file P, medication vocabulary $\mathcal{V}_d$, OCR threshold τ_{ocr}
Ensure: Confirmed medication $\hat{d}$ or insufficient-confirmation status

if P contains selectable text **then**
    Extract text T_P using Eq. (15)
else
    Preprocess image using Eq. (16)
    Extract text T_P using Eq. (15)
end if
Initialize candidate set $\mathcal{M}_P \leftarrow \emptyset$
for each medication name $v \in \mathcal{V}_d$ **do**
    Compute $S(q_m, v)$ using Eq. (17)
    if $S(q_m, v) \geq \tau_{ocr}$ **then**
        Add v to $\mathcal{M}_P$
    end if
end for
Present $\mathcal{M}_P$ for user confirmation
if a candidate is confirmed **then**
    Set confirmed medication $\hat{d}$
    return $\hat{d}$
else
    return insufficient-confirmation status
end if

5.6. Model Input Representation

For the AI branch, each medication and food/herb pair is converted into a structured text input. The input string is defined in Eq. (19).

$$z = \tilde{d} \,||\, [SEP] \,||\, \tilde{f}, \tag{19}$$

where z is the model input string, $||$ denotes string concatenation, $[SEP]$ separates fields, $\tilde{d}$ is the normalized medication name, and $\tilde{f}$ is the normalized food/herb name. Eq. (19) preserves field boundaries while allowing the model to use a text-based representation.

The model input is converted into a TF-IDF vector [44]. Term frequency is computed using Eq. (20).

$$\mathrm{tf}(a, z_b) = \frac{n(a, z_b)}{\sum_{a' \in z_b} n(a', z_b)}, \tag{20}$$

where a is a unigram or bigram feature, z_b is the bth training input string, and $n(a, z_b)$ is the count of feature a in z_b. Eq. (20) measures how strongly a term appears within one input pair.

The inverse document frequency is computed using Eq. (21).

$$\mathrm{idf}(a) = \log\left(\frac{B+1}{\mathrm{df}(a)+1}\right) + 1, \tag{21}$$

where B is the number of training input strings and $\mathrm{df}(a)$ is the number of training inputs containing feature a. Eq. (21) gives lower weight to terms that appear in many training inputs and higher weight to more specific terms.

The final TF-IDF value is shown in Eq. (22).

$$x_{b,a} = \mathrm{tf}(a, z_b) \cdot \mathrm{idf}(a). \tag{22}$$

In Eq. (22), $x_{b,a}$ is the TF-IDF value of feature a for input z_b. The complete vector $\mathbf{x}_b = [x_{b,1}, x_{b,2}, \ldots, x_{b,p}]$ is used as the model input, where p is the number of selected unigram and bigram features.

5.7. MedSafe-GATE v1 Model

MedSafe-GATE v1 is implemented as an interpretable safety-gated classifier built on sparse TF-IDF features and a balanced linear decision layer. The linear support-vector formulation is grounded in margin-based classification theory [45], and support-vector methods have been shown to be effective for high-dimensional text classification problems [46]. For each class, the model computes a linear score, as shown in Eq. (23).

$$g_c(\mathbf{x}) = \mathbf{w}_c^T \mathbf{x} + b_c. \tag{23}$$

In Eq. (23), $g_c(\mathbf{x})$ is the score for class c, $\mathbf{x}$ is the TF-IDF feature vector, $\mathbf{w}_c$ is the learned class-specific weight vector, and b_c is the class bias. This structure supports interpretability because each feature contributes $w_{c,a}x_a$ to the score of class c.

The baseline prediction is selected using Eq. (24).

$$\hat{y}_{base} = \arg\max_{c \in \mathcal{C}} g_c(\mathbf{x}). \tag{24}$$

Eq. (24) selects the class with the highest linear score before safety gating. To reduce the effect of class imbalance, class weights are computed using Eq. (25).

$$\alpha_c = \frac{B}{|\mathcal{C}| n_c}, \tag{25}$$

where α_c is the weight for class c, B is the total number of training examples, $|\mathcal{C}|$ is the number of classes, and n_c is the number of training examples in class c. Eq. (25) increases the training influence of underrepresented classes.

The class-weighted linear support vector objective can be written at a high level as Eq. (26).

$$\min_{\mathbf{W},\mathbf{b}} \frac{1}{2} \sum_{c \in \mathcal{C}} \|\mathbf{w}_c\|_2^2 + C \sum_{b=1}^{B} \alpha_{y_b} \max\left(0, 1 - g_{y_b}(\mathbf{x}_b) + \max_{c \neq y_b} g_c(\mathbf{x}_b)\right). \tag{26}$$

In Eq. (26), $\mathbf{W}$ contains all class weight vectors, $\mathbf{b}$ contains the bias terms, C is the regularization parameter, $\mathbf{x}_b$ is the bth training vector, y_b is its true label, and α_{y_b} is the class weight for that label. This objective penalizes misclassification while giving more importance to underrepresented labels.

Because the safety gate requires a confidence score for the no-interaction class, calibrated class confidence values are represented using Eq. (27).

$$\mathbf{p}(\mathbf{x}) = \kappa(\mathbf{g}(\mathbf{x})), \tag{27}$$

where $\mathbf{g}(\mathbf{x}) = [g_{c_A}(\mathbf{x}), g_{c_C}(\mathbf{x}), g_{c_N}(\mathbf{x})]$ is the score vector, $\kappa(\cdot)$ is the calibration function, and $\mathbf{p}(\mathbf{x}) = [p_A(\mathbf{x}), p_C(\mathbf{x}), p_N(\mathbf{x})]$ is the calibrated confidence vector. The value $p_N(\mathbf{x})$ is used by the safety gate.

Calibration is included because classifier scores do not always behave as reliable probability estimates, especially when confidence values are used for downstream decision rules [47].

The false-safe count is defined in Eq. (28). This metric counts cases where a risky pair is predicted as no known interaction.

$$FS = \sum_{q=1}^{Q} \mathbb{I}\big(y_q \in \{c_A, c_C\} \wedge \hat{y}_q = c_N\big), \tag{28}$$

where FS is the false-safe count, Q is the number of evaluated examples, y_q is the true label, $\hat{y}_q$ is the predicted label, and $\mathbb{I}(\cdot)$ is the indicator function. The false-safe rate is computed using Eq. (29).

$$FSR = \frac{\sum_{q=1}^{Q} \mathbb{I}\big(y_q \in \{c_A, c_C\} \wedge \hat{y}_q = c_N\big)}{\sum_{q=1}^{Q} \mathbb{I}\big(y_q \in \{c_A, c_C\}\big)}. \tag{29}$$

Eq. (29) measures the percentage of risky examples that are incorrectly displayed as no known interaction. This metric is central to MedSafe-GATE v1 because a false-safe output can create a misleading sense of safety.

The MedSafe-GATE v1 decision rule is shown in Eq. (30).

$$\hat{y}_{gate} = \begin{cases} c_C, & \text{if } \hat{y}_{base} = c_N \text{ and } p_N(\mathbf{x}) < \tau_N, \\ \hat{y}_{base}, & \text{otherwise.} \end{cases} \tag{30}$$

In Eq. (30), $\hat{y}_{gate}$ is the final safety-gated model output, τ_N is the no-interaction confidence threshold, and $p_N(\mathbf{x})$ is the calibrated confidence for c_N. A no-interaction result is allowed only when the model confidence reaches the selected threshold. Otherwise, the output is shifted to "Use caution" to reduce false-safe behavior.

Algorithm 3 summarizes the training flow for MedSafe-GATE v1. The algorithm connects the text representation in Eq. (19), the TF-IDF representation in Eq. (20)–Eq. (22), the balanced classifier in Eq. (26), and the safety threshold used in Eq. (30).

Algorithm 3 MedSafe-GATE v1 Training

Require: Pair-level dataset $\mathcal{D}_{pair}$ and class set $\mathcal{C}$
Ensure: TF-IDF vectorizer, trained classifier, calibration function $\kappa(\cdot)$, and threshold τ_N
1: Split $\mathcal{D}_{pair}$ into train, validation, and test sets without pair overlap
2: **for** each training pair **do**
3: Build input string z using Eq. (19)
4: **end for**
5: Compute TF-IDF features using Eq. (20)–Eq. (22)
6: Compute class weights using Eq. (25)
7: Train the class-weighted linear classifier using Eq. (26)
8: Calibrate decision scores using Eq. (27)
9: Select τ_N using validation false-safe behavior from Eq. (29)
10: **return** trained model components and τ_N

Algorithm 4 summarizes safety-gated inference. The algorithm first predicts the baseline class and then applies the false-safe-aware gate.

Algorithm 4 MedSafe-GATE v1 Inference

Require: Normalized medication $\tilde{d}$, normalized food/herb $\tilde{f}$, threshold τ_N
Ensure: Safety-gated prediction $\hat{y}_{gate}$
1: Build input string z using Eq. (19)
2: Transform z into TF-IDF vector $\mathbf{x}$ using Eq. (22)
3: Compute class scores using Eq. (23)
4: Select baseline prediction using Eq. (24)
5: Compute calibrated confidence using Eq. (27)
6: Apply the safety gate using Eq. (30)
7: **return** $\hat{y}_{gate}$

5.8. End-to-End Recommendation Logic

The end-to-end recommendation begins with either a manually entered medication or a medication confirmed through RxOCR-Guard. The normalized user inputs are defined in Eq. (31).

$$\tilde{d} = \eta(\hat{d}), \qquad \tilde{f} = \eta(\hat{f}), \tag{31}$$

where $\hat{d}$ is the confirmed or manually entered medication name, $\hat{f}$ is the selected food/herb input, and $\tilde{d}$ and $\tilde{f}$ are their normalized forms. The decision source is selected using Eq. (32).

$$\Omega = \begin{cases} \text{EvidenceLookup}, & \text{if } (\tilde{d}, \tilde{f}) \in \mathcal{L}, \\ \text{MedSafe-GATE}, & \text{if } \tilde{d} \in \mathcal{V}_d,\ \tilde{f} \in \mathcal{V}_f, \text{ and } (\tilde{d}, \tilde{f}) \notin \mathcal{L}, \\ \text{CoverageGuard}, & \text{otherwise.} \end{cases} \tag{32}$$

In Eq. (32), Ω records whether the recommendation came from exact evidence, AI screening, or coverage rejection. This decision-source field is included in the signed recommendation output for traceability.

The final recommendation label is assigned using Eq. (33).

$$R = \begin{cases} \mathcal{L}[(\tilde{d}, \tilde{f})], & \text{if } \Omega = \text{EvidenceLookup}, \\ \hat{y}_{gate}, & \text{if } \Omega = \text{MedSafe-GATE}, \\ \text{Insufficient evidence}, & \text{if } \Omega = \text{CoverageGuard}. \end{cases} \tag{33}$$

Eq. (33) gives priority to exact DDID evidence, uses MedSafe-GATE v1 only for covered non-exact pairs, and rejects unsupported inputs with an insufficient-evidence response.

Algorithm 5 summarizes the complete screening flow. It combines RxOCR-Guard, evidence lookup, input coverage checking, MedSafe-GATE v1 inference, and output signing.

Algorithm 5 End-to-End iMediFood-Shield Screening

Require: Medication input $\hat{d}$ or prescription file P, food/herb input $\hat{f}$, lookup table $\mathcal{L}$, vocabularies $\mathcal{V}_d$ and $\mathcal{V}_f$
Ensure: Signed recommendation record $\mathcal{S}_{output}$
1: **if** prescription file P is provided **then**
2: Obtain confirmed medication $\hat{d}$ using Algorithm 2
3: **end if**
4: Normalize inputs using Eq. (31)
5: Select decision source Ω using Eq. (32)
6: Generate recommendation R using Eq. (33)
7: Create recommendation record using Eq. (40)
8: Sign recommendation using Eq. (41)
9: **return** $\mathcal{S}_{output}$

5.9. Tamper-Evident Security Manifest

The security layer protects all digital assets that influence the final recommendation. The protected asset set is defined in Eq. (34).

$$\mathcal{A} = \{A_1, A_2, \ldots, A_G\}, \tag{34}$$

where $\mathcal{A}$ is the protected asset set, A_g is the gth protected file, and G is the number of protected files. In the prototype, protected assets include the DDID-derived evidence table, model artifact, policy file, label mapping file, dataset summaries, output logs, and signed recommendation records.

For each asset, a SHA-256 digest is computed as shown in Eq. (35).

$$h_g = H(A_g), \tag{35}$$

where $H(\cdot)$ is the SHA-256 hash function and h_g is the digest of asset A_g. Eq. (35) provides a compact integrity fingerprint for each protected file.

The manifest body is defined in Eq. (36).

$$\mathcal{B} = \{(A_g, h_g)\}_{g=1}^{G}. \tag{36}$$

In Eq. (36), $\mathcal{B}$ stores the expected hash value for each protected asset. To protect the manifest itself, HMAC-SHA256 is used. The HMAC computation is shown in Eq. (37).

$$\mathrm{HMAC}_K(m) = H\big((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, m)\big), \tag{37}$$

where K is the secret key, K' is the block-sized key derived from K, m is the message, $H(\cdot)$ is SHA-256 in this implementation, $\oplus$ denotes bitwise exclusive OR, $||$ denotes concatenation, and $opad$ and $ipad$ are the outer and inner padding constants. A canonical serialization format is used before signing so that the same manifest or recommendation record produces the same byte representation during signing and verification. Eq. (37) is used for both manifest signing and recommendation-output signing. The use of HMAC is also supported by prior cryptographic work on keyed hash functions for message authentication and formal analysis of HMAC security [48,49].

The manifest signature is computed using Eq. (38).

$$\sigma_{\mathcal{B}} = \mathrm{HMAC}_K(\mathrm{Serialize}(\mathcal{B})). \tag{38}$$

Here, $\sigma_{\mathcal{B}}$ is the authentication tag for the manifest body. The signed manifest is represented in Eq. (39).

$$\mathcal{S}_{manifest} = (\mathcal{B}, \sigma_{\mathcal{B}}). \tag{39}$$

Eq. (39) defines the trusted reference state used during verification. If a protected asset is modified after manifest creation, verification fails because the recomputed hash does not match the stored value.

Algorithm 6 summarizes the manifest-creation process.

Algorithm 6 Security Manifest Creation

Require: Protected asset set $\mathcal{A}$ and HMAC key K
Ensure: Signed manifest $\mathcal{S}_{manifest}$
1: Initialize manifest body $\mathcal{B} \leftarrow \emptyset$
2: **for** each asset $A_g \in \mathcal{A}$ **do**
3: Compute asset digest h_g using Eq. (35)
4: Add (A_g, h_g) to $\mathcal{B}$
5: **end for**
6: Compute manifest signature using Eq. (38)
7: Form signed manifest using Eq. (39)
8: **return** $\mathcal{S}_{manifest}$

5.10. Recommendation Output Signing

Each recommendation output is signed so that post-inference modification can be detected. The recommendation record is defined in Eq. (40).

$$\mathcal{O} = (\hat{d}, \hat{f}, \tilde{d}, \tilde{f}, R, \Omega, h_{data}, h_{model}, h_{policy}, \lambda), \tag{40}$$

where $\mathcal{O}$ is the unsigned recommendation record, $\hat{d}$ and $\hat{f}$ are the user-facing medication and food/herb inputs, $\tilde{d}$ and $\tilde{f}$ are the normalized inputs, R is the recommendation label, Ω is the decision source, h_{data} is the evidence-table hash, h_{model} is the model-file hash, h_{policy} is the policy-file hash, and λ represents metadata such as timestamp and record identifier.

The output signature is computed using Eq. (41).

$$\sigma_{\mathcal{O}} = \mathrm{HMAC}_K(\mathrm{Serialize}(\mathcal{O})). \tag{41}$$

In Eq. (41), $\sigma_{\mathcal{O}}$ authenticates the recommendation record. The final signed output is represented in Eq. (42).

$$\mathcal{S}_{output} = (\mathcal{O}, \sigma_{\mathcal{O}}). \tag{42}$$

Eq. (42) links the recommendation to the input names, decision source, and protected asset hashes. Any change to the displayed recommendation or its metadata causes signature verification to fail. This design follows the same general motivation as secure audit-log mechanisms, where generated records should not be modified silently after creation [50].

5.11. Verification Logic

Verification checks both the signed manifest and the signed recommendation output. The manifest signature is recomputed using Eq. (43).

$$\sigma^*_{\mathcal{B}} = \mathrm{HMAC}_K(\mathrm{Serialize}(\mathcal{B})). \tag{43}$$

In Eq. (43), $\sigma^*_{\mathcal{B}}$ is the recomputed manifest signature. It is compared with the stored signature using Eq. (44).

$$V_{sig} = \mathbb{I}(\sigma^*_{\mathcal{B}} = \sigma_{\mathcal{B}}). \tag{44}$$

Here, $V_{sig} = 1$ means the manifest signature is valid, while $V_{sig} = 0$ means the manifest signature check fails. Each protected file is then rehashed using Eq. (45).

$$h^*_g = H(A^{current}_g). \tag{45}$$

In Eq. (45), h^*_g is the current digest of protected asset $A^{current}_g$. File integrity is checked using Eq. (46).

$$V_{files} = \prod_{g=1}^{G} \mathbb{I}\left(h^*_g = h_g\right). \tag{46}$$

Eq. (46) returns 1 only when every protected file matches its stored digest. The complete manifest verification result is computed using Eq. (47).

$$V_{manifest} = V_{sig} \cdot V_{files}. \tag{47}$$

In Eq. (47), $V_{manifest} = 1$ indicates that the manifest signature and all protected file hashes are valid. Output verification is performed by recomputing the recommendation signature, as shown in Eq. (48).

$$\sigma^*_{\mathcal{O}} = \mathrm{HMAC}_K(\mathrm{Serialize}(\mathcal{O})). \tag{48}$$

The signed recommendation is accepted only when Eq. (49) returns 1.

$$V_{output} = \mathbb{I}(\sigma^*_{\mathcal{O}} = \sigma_{\mathcal{O}}). \tag{49}$$

Eq. (49) detects unauthorized changes to the recommendation label, decision source, input names, or protected-asset hashes stored in the signed output.

Algorithm 7 summarizes the complete verification process. The algorithm first validates the manifest, then validates the recommendation output, and finally returns the provenance status.

Algorithm 7 Tamper-Evident Verification

Require: Signed manifest $\mathcal{S}_{manifest}$, protected assets $\mathcal{A}$, signed output $\mathcal{S}_{output}$, HMAC key K
Ensure: Verification status
Recompute manifest signature using Eq. (43)
Check manifest signature using Eq. (44)
for each protected asset $A_g \in \mathcal{A}$ **do**
    Recompute current hash using Eq. (45)
end for
Check protected file integrity using Eq. (46)
Compute manifest verification result using Eq. (47)
Recompute output signature using Eq. (48)
Check output integrity using Eq. (49)
if $V_{manifest} = 1$ and $V_{output} = 1$ **then**
    return VERIFIED_PROVENANCE
else
    return FAILED_VERIFICATION
end if

5.12. Edge Implementation

The Raspberry Pi edge implementation executes the tamper-evident verification logic locally on a resource-constrained device. This implementation does not retrain MedSafe-GATE v1 on the Raspberry Pi. Instead, the protected files, signed manifest, software HMAC key, and verification script are deployed to the edge device to test whether the local system can reproduce the trusted manifest state and reject a tampered state. The implementation follows the verification logic defined in Algorithm 7 and extends it with repeated timing measurements for edge validation.

For each Raspberry Pi validation run, the manifest-level status is computed from the manifest signature check and protected-file hash checks, as shown in Eq. (50).

$$S_r = \begin{cases} \mathit{VERIFIED\_PROVENANCE}, & \text{if } V_{\text{manifest},r} = 1, \\ \mathit{FAILED\_VERIFICATION}, & \text{if } V_{\text{manifest},r} = 0. \end{cases} \tag{50}$$

In Eq. (50), S_r is the verification status for Raspberry Pi run r, and $V_{\text{manifest},r}$ is the manifest verification result for that run. A verified status means that the signed manifest and protected-file hashes match the trusted state, while a failed status means that at least one verification check fails.

The runtime for each Raspberry Pi verification run is measured using the elapsed time between the start and end of the verification procedure, as shown in Eq. (51).

$$\Delta t_r = t_r^{\text{end}} - t_r^{\text{start}}. \tag{51}$$

In Eq. (51), Δt_r is the elapsed verification time for run r, t_r^{start} is the timestamp before verification begins, and t_r^{end} is the timestamp after verification completes.

The average Raspberry Pi verification time across repeated runs is computed using Eq. (52).

$$\mu_{\mathrm{RPI}} = \frac{1}{R}\sum_{r=1}^{R}\Delta t_r. \tag{52}$$

In Eq. (52), μ_{RPI} is the mean Raspberry Pi verification time, R is the total number of repeated validation runs, and Δt_r is the elapsed time for run r.

The runtime variation across repeated Raspberry Pi verification runs is computed using Eq. (53).

$$\sigma_{\mathrm{RPI}} = \sqrt{\frac{1}{R-1}\sum_{r=1}^{R}(\Delta t_r - \mu_{\mathrm{RPI}})^2}. \tag{53}$$

In Eq. (53), σ_{RPI} is the standard deviation of Raspberry Pi verification time. This value shows how stable the local verification runtime is across repeated edge-device runs.

Algorithm 8 summarizes the Raspberry Pi edge validation procedure. The algorithm verifies the trusted protected-asset state, verifies a tampered state, records the resulting status values, and stores timing values for later reporting in the Results and Discussion section.

Algorithm 8 Raspberry Pi Edge Verification and Timing

Require: Signed manifest S_{manifest}, protected asset set A, HMAC key K, number of runs R
Ensure: Verified-state timing list T_{verified}, failed-state timing list T_{failed}, verification statuses

Initialize $T_{\mathrm{verified}} \leftarrow \emptyset$
Initialize $T_{\mathrm{failed}} \leftarrow \emptyset$
for $r = 1$ to R **do**
    Record t_r^{start}
    Verify unchanged protected assets using Algorithm 7
    Record t_r^{end}
    Compute Δt_r using Eq. (51)
    Store Δt_r in T_{verified}
    Store verified-state status S_r using Eq. (50)
end for
for $r = 1$ to R **do**
    Apply a tampered protected-asset state
    Record t_r^{start}
    Verify tampered protected assets using Algorithm 7
    Record t_r^{end}
    Compute Δt_r using Eq. (51)
    Store Δt_r in T_{failed}
    Store failed-state status S_r using Eq. (50)
end for
Compute mean timing using Eq. (52)
Compute timing variation using Eq. (53)
return T_{verified}, T_{failed}, and verification statuses

The Raspberry Pi implementation therefore validates the same tamper-evident security layer on local edge hardware. The implementation is limited to prototype security validation with a software HMAC-based key and does not claim hardware-rooted trust. This is not clinical decision support. Always confirm recommendations with a licensed healthcare professional.

5.13. Attack Simulation

The security evaluation simulates integrity attacks against files and outputs that influence the final recommendation. The implemented attack set is defined in Eq. (54).

$$\mathcal{T} = \{T_{data}, T_{model}, T_{policy}, T_{log}, T_{display}\}, \tag{54}$$

where T_{data} denotes evidence-table tampering, T_{model} denotes model-file tampering, T_{policy} denotes policy modification or rollback, T_{log} denotes output-log tampering, and $T_{display}$ denotes displayed recommendation tampering.

Each attack modifies a protected asset or signed output after manifest or output signing. This attack operation is represented in Eq. (55).

$$A_g^{attack} = T_j(A_g), \tag{55}$$

where T_j is the jth attack and A_g^{attack} is the modified version of protected asset A_g. A successful security response means that verification fails after tampering, as expressed in Eq. (56).

$$V_{attack}(T_j) = 0. \tag{56}$$

In Eq. (56), $V_{attack}(T_j) = 0$ indicates that the tampered state is rejected by the verification layer. The attack-detection rate is computed using Eq. (57).

$$ADR = \frac{\sum_{j=1}^{|\mathcal{T}|} \mathbb{I}\big(V_{attack}(T_j) = 0\big)}{|\mathcal{T}|}. \tag{57}$$

Eq. (57) measures the fraction of implemented tampering attacks detected by the prototype. This metric is used only for the implemented attack simulations and does not imply complete security against all possible attacks.

5.14. Implementation Summary

The implementation combines the dataset logic in Algorithm 1, the prescription confirmation flow in Algorithm 2, the MedSafe-GATE v1 training and inference procedures in Algorithm 3 and Algorithm 4, the full screening workflow in Algorithm 5, and the verification process in Algorithm 7. Together, these components define how iMediFood-Shield forms evidence-grounded labels, screens medication and food/herb pairs, reduces false-safe outputs, signs recommendation records, and detects tampering of protected files or displayed recommendations.

6. Results and Discussion

This section presents the experimental results for iMediFood-Shield. The evaluation includes DDID dataset formation, AI screening performance, false-safe reduction, evidence lookup behavior, tamper-evident security validation, runtime overhead, Raspberry Pi edge validation, and mobile prototype demonstration. The results are interpreted as research-prototype findings and do not represent clinical validation.

6.1. Dataset Formation Results

The DDID interaction file was processed into row-level and pair-level datasets for food, herb, and medication interaction screening. Table 3 summarizes the main dataset formation outputs. The raw interaction file contained 23,950 records and 26 columns. After cleaning and safety-priority pair aggregation, the final pair-level dataset contained 17,693 unique medication–food/herb pairs.

Table 3. DDID Dataset Formation Summary for iMediFood-Shield.

Dataset property	Value
Raw DDID interaction records	23,950
Raw DDID columns	26
Clean row-level records	23,950
Unique medication–food/herb pairs	17,693
Pair-level herb records	13,964
Pair-level food records	3,729
Pairs with effect conflicts	495
Pairs with paper-label conflicts	285

Table 3 shows that the dataset contains a larger number of herb-related records than food-related records. The conflict counts show why pair-level safety-priority aggregation was needed, since some medication–food/herb pairs appeared in multiple DDID records with different effect labels or screening labels.

The screening-label distribution after DDID effect mapping is shown in Table 4. The row-level dataset is dominated by "Use caution" records, and the same trend remains after pair-level aggregation. The "No known interaction in selected source" label is source-bounded and does not claim clinical safety outside the selected DDID evidence source.

Table 4. Screening-Label Distribution Before and After Pair-Level Aggregation.

Screening label	Row-level count	Pair-level count
Use caution	21,183	15,766
No known interaction in selected source	2,374	1,566
Avoid	393	361

Table 4 shows a strong class imbalance, with "Use caution" representing most records. This imbalance motivated the use of balanced classifiers and false-safe-aware evaluation rather than relying only on overall accuracy.

Table 5 reports the train, validation, and test split sizes. The split was performed at the unique pair level, and no medication–food/herb pair appeared in more than one split.

Table 5. Train, Validation, and Test Split Summary.

Split property	Value
Full pair-level dataset	17,693
Training set	12,385
Validation set	2,654
Test set	2,654
Train–validation pair overlap	0
Train–test pair overlap	0
Validation–test pair overlap	0

Table 5 confirms that the evaluation avoids exact pair leakage across train, validation, and test sets. This is important because repeated medication–food/herb pairs across splits could overestimate model performance.

6.2. AI Screening Performance

Table 6 compares the main AI screening settings on the DDID-based test set. The balanced AI baseline is the TF-IDF + LinearSVC balanced model. MedSafe-GATE v1 adds a safety gate to reduce uncertain no-interaction outputs.

Table 6. AI Screening Performance and False-Safe Behavior on the Test Set.

Model setting	Threshold	Accuracy	Macro F1	False-safe count	False-safe rate	Operating role
Balanced AI baseline	–	91.86%	74.01%	128	5.29%	Strongest non-gated baseline.
MedSafe-GATE v1 balanced-safety mode	0.60	91.67%	68.60%	47	1.94%	Balances accuracy and false-safe reduction.
MedSafe-GATE v1 strict-screening mode	0.70	90.99%	61.57%	15	0.62%	Strongest safety-focused operating point.

Table 6 shows that the balanced AI baseline achieved the highest macro F1 among the listed settings, while MedSafe-GATE v1 strict-screening mode achieved the lowest false-safe rate. The strict mode slightly reduces accuracy and macro F1 because it intentionally shifts uncertain no-interaction outputs toward caution.

Figure 5 shows the row-normalized confusion matrix for MedSafe-GATE v1 strict-screening mode. The matrix gives class-level detail for the strict operating point reported in Table 6.

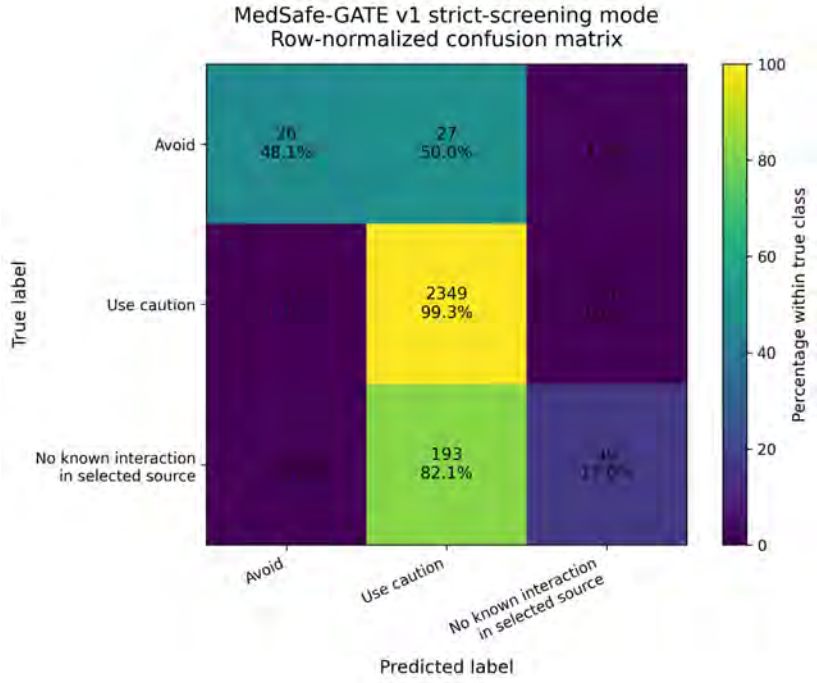

Figure 5. Row-Normalized Confusion Matrix for MedSafe-GATE v1 Strict-Screening Mode.

Figure 5 shows that strict-screening mode keeps false-safe predictions low. Only 1 true "Avoid" case and 14 true "Use caution" cases are predicted as "No known interaction in selected source". The matrix also shows conservative behavior: many true "No known interaction in selected source" cases are shifted to "Use caution". This behavior is expected because strict-screening mode prioritizes reducing false-safe outputs over maximizing no-interaction recall.

6.3. False-Safe Reduction Analysis

False-safe outputs are the most safety-sensitive error type in iMediFood-Shield because they occur when a truly risky pair is shown as "No known interaction in selected source". Table 7 summarizes the false-safe reduction across model settings.

A calibrated balanced baseline is also included as an intermediate comparison to show how calibration alone affects false-safe reduction before applying MedSafe-GATE v1.

Table 7. False-Safe Reduction Across Evaluated Model Settings.

Model setting	Count	Rate (%)	Reduced	Rel. red. (%)
Balanced AI baseline	128	5.29	–	–
Calibrated baseline	96	3.97	32	25.00
MedSafe-GATE v1, $\theta = 0.60$	47	1.94	81	63.28
MedSafe-GATE v1, $\theta = 0.70$	15	0.62	113	88.28

Table 7 shows that the strict-screening threshold reduced false-safe cases from 128 to 15. This corresponds to an 88.28% relative reduction compared with the balanced AI baseline.

Figure 6 visualizes the same reduction trend. The figure is useful because it directly shows the main safety improvement of MedSafe-GATE v1.

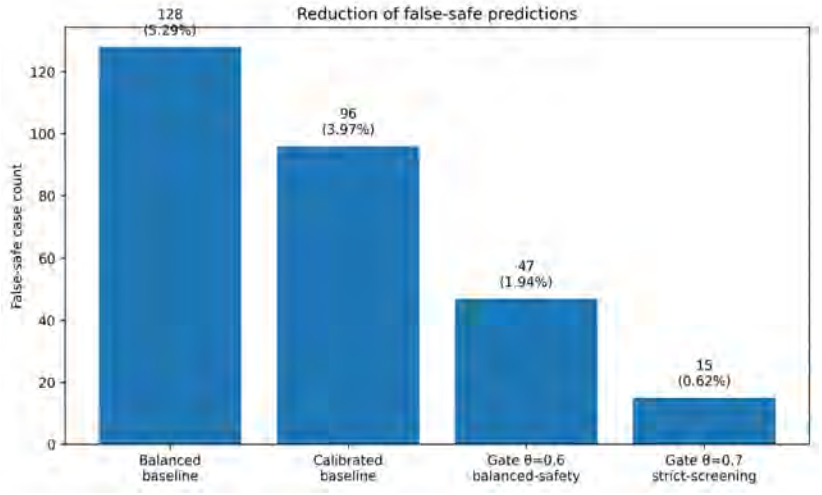

Figure 6. Reduction of False-Safe Predictions Across Baseline, Calibrated, and Safety-Gated Settings.

Figure 6 shows that increasing the safety threshold reduces false-safe outputs. The strict-screening setting gives the strongest safety-focused result, but this comes with more conservative caution predictions, as also shown in Figure 5.

6.4. Evidence Lookup Guard Analysis

The final inference policy applies DDID exact-pair evidence lookup before AI prediction. This design is important because exact evidence should not be replaced by a model prediction when the pair exists in the selected source. Table 8 summarizes the evidence lookup guard behavior for the remaining strict-screening false-safe cases.

Table 8. Evidence Lookup Guard Behavior for Strict-Screening False-Safe Cases.

Evidence lookup guard property	Value
Strict-screening false-safe cases before evidence guard	15
False-safe cases with exact DDID pair-level evidence	15
Exact-evidence false-safe cases blocked by lookup guard	15
Remaining exact-evidence false-safe cases after lookup guard	0

Table 8 presents an ablation-style analysis of the deployed evidence-first policy. The 15 remaining strict-screening false-safe cases had exact DDID pair-level evidence in the selected source. Therefore, in the deployed pipeline, these model-only false-safe outputs would be blocked because the evidence lookup layer returns the DDID-derived label before

model output is used. This result supports the use of evidence-first screening rather than a model-only recommendation pipeline when exact curated evidence is available.

6.5. Security Attack Detection

The tamper-evident security layer was evaluated using implemented integrity attacks against protected files and signed recommendation outputs. Table 9 summarizes the evaluated attack cases and verification outcomes.

Table 9. Tamper-Evident Security Attack Detection Results.

Attack case	Tampered asset or output	Verification status	Detection result
DDID lookup table tampering	Evidence lookup table modified after signing	*FAILED_VERIFICATION*	Detected
Model file tampering	Trained model artifact modified after signing	*FAILED_VERIFICATION*	Detected
Policy modification or rollback	Inference policy changed after signing	*FAILED_VERIFICATION*	Detected
Output log tampering	Recommendation output log modified after signing	*FAILED_VERIFICATION*	Detected
Displayed recommendation tampering	Signed recommendation label changed after generation	*FAILED_OUTPUT_VERIFICATION*	Detected

Table 9 shows that all five implemented tampering attacks were detected in the prototype evaluation. This result means that the evaluated modifications caused verification failure. It does not imply complete security against all possible attacks or deployment conditions.

6.6. Runtime and Security Overhead

Table 10 reports the measured overhead of hashing, manifest verification, recommendation signing, and signed-output verification in the software prototype. These values measure the cost of the tamper-evident security layer.

Table 10. Security Overhead for Protected-File Hashing, Manifest Verification, and Output Signing.

Operation	Target	Target size	Average time
SHA-256 file hashing	DDID lookup table	12.80 MB	15.485 ms
SHA-256 file hashing	Raw interaction file	18.39 MB	21.556 ms
SHA-256 file hashing	Model file	0.49 MB	0.577 ms
Full manifest verification	9 protected files	–	48.539 ms
Sign one recommendation	One recommendation record	–	0.030 ms
Verify signed outputs	5 signed records	–	0.143 ms
Verify tampered signed outputs	5 records with one tampered record	–	0.143 ms

Table 10 shows that recommendation signing and signed-output verification add very small overhead in the software prototype. Full manifest verification is larger because it recomputes hashes for multiple protected files.

Table 11 reports secured inference latency for representative cases. The measured time includes recommendation generation and output signing.

Table 11. Fast Secured Inference Latency for Representative Screening Cases.

Medication	Food/herb item	Recommendation	Decision source	Average time
Ibuprofen	Ginkgo Biloba	Avoid	DDID exact-pair evidence lookup + signing	0.057 ms
Atorvastatin	High-Fat Meal	Use caution	DDID exact-pair evidence lookup + signing	0.056 ms
RandomDrugXYZ	RandomFoodXYZ	Insufficient evidence	Coverage guard + signing	0.061 ms
Known DDID entities without exact pair	Known food/herb item	Model-derived label	Known-entity model branch + signing	3.614 ms

Table 11 shows that exact lookup and coverage-guard branches are faster than the model branch. The known-entity model branch is slower because it requires TF-IDF transformation, model inference, safety-gate processing, and output signing.

6.7. Edge Validation

The Raspberry Pi validation evaluates whether the tamper-evident verification layer can run locally on resource-constrained edge hardware. Table 12 summarizes the device configuration used for the Raspberry Pi tests.

Table 12. Raspberry Pi Edge-Device Configuration Used for iMediFood-Shield Validation.

Parameter	Value
Operating system	Linux 5.10.103-v7l+
Python version	3.9.2
Machine architecture	armv7l
CPU cores	4
CPU model	Cortex-A72
CPU maximum frequency	1500 MHz
RAM	3.75 GB

Table 12 defines the Raspberry Pi hardware and software context for the edge validation. The CPU, memory, and architecture values show that the validation was performed on a resource-constrained local edge device rather than only on a desktop-class environment.

Figure 7 shows the Raspberry Pi execution of signed food–medication screening examples.

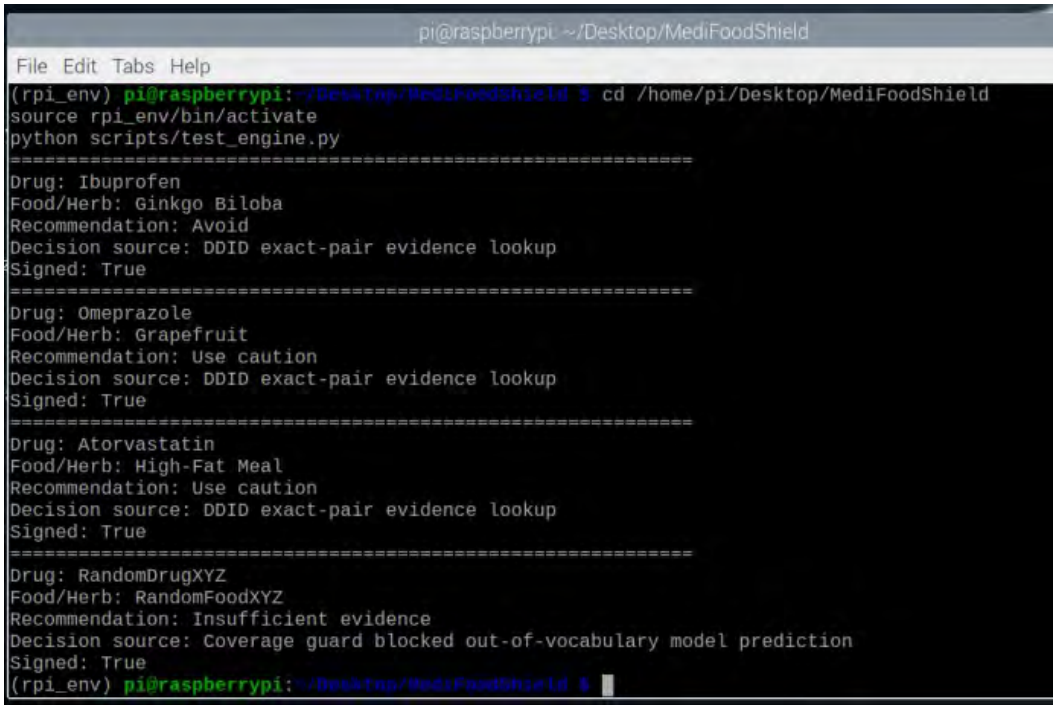

Figure 7. Raspberry Pi Execution of Signed Food–Medication Screening Examples.

Figure 7 shows that the Raspberry Pi prototype executed evidence lookup, coverage guard behavior, and recommendation signing locally. Exact DDID-supported pairs returned evidence-derived labels, while an out-of-vocabulary medication–food pair returned an insufficient-evidence response.

The same Raspberry Pi screening outputs are summarized in Table 13. The table defines the medication input, food/herb input, recommendation label, decision source, and signed-output status for each local edge example.

Table 13. Raspberry Pi Signed Recommendation Examples.

Medication	Food/herb item	Recommendation	Decision source	Signed
Ibuprofen	Ginkgo Biloba	Avoid	DDID exact-pair evidence lookup	True
Omeprazole	Grapefruit	Use caution	DDID exact-pair evidence lookup	True
Atorvastatin	High-Fat Meal	Use caution	DDID exact-pair evidence lookup	True
RandomDrugXYZ	RandomFoodXYZ	Insufficient evidence	Coverage guard blocked out-of-vocabulary model prediction	True

Table 13 shows that signed outputs were generated for both supported and unsupported input cases. The unsupported case is important because iMediFood-Shield does not force a model prediction when the medication or food/herb item is outside the DDID-derived vocabulary.

Figure 8 shows the verified and tampered verification states on the Raspberry Pi.

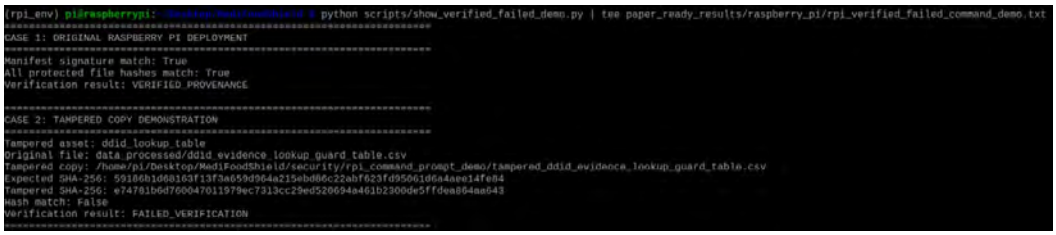

Figure 8. Raspberry Pi Verification of Original and Tampered Protected Assets.

Figure 8 shows that the unchanged Raspberry Pi deployment returned *VERIFIED_PROVENANCE*. The tampered DDID lookup-table copy produced a hash mismatch and returned *FAILED_VERIFICATION*. This confirms that the local edge device detects protected-asset modification.

Figure 9 shows the Raspberry Pi security refresh and attack-detection summary.

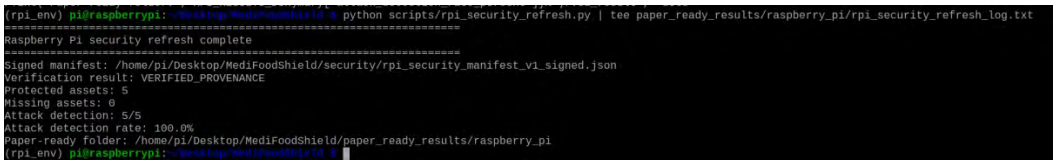

Figure 9. Raspberry Pi Security Refresh and Attack-Detection Summary.

Figure 9 shows that the refreshed signed manifest verified the protected assets, reported no missing assets, and detected all five implemented tampering attacks. The same security-refresh output is summarized in Table 14.

Table 14. Raspberry Pi Security Refresh Summary.

Security refresh property	Value
Verification result	*VERIFIED_PROVENANCE*
Protected assets	5
Missing assets	0
Implemented attacks detected	5
Attack detection rate	100.0%

Table 14 shows that the Raspberry Pi deployment successfully verified the protected local asset set and detected all implemented attack cases in the prototype test. The 100.0%

value refers only to the implemented attack simulations and should not be interpreted as a complete security guarantee.

Figure 10 shows the repeated Raspberry Pi verification timing output.

Figure 10. Raspberry Pi Repeated Verification Timing Output.

Figure 10 shows that 30 Raspberry Pi verification runs were executed for both verified and failed-verification states. The timing values from this output are summarized in Table 15.

Table 15. Raspberry Pi Verification Timing Over 30 Runs.

Verification state	Runs	Mean time	Std. dev.	Min. time	Max. time
VERIFIED_PROVENANCE	30	167.790 ms	0.137 ms	167.504 ms	168.050 ms
FAILED_VERIFICATION	30	159.709 ms	0.436 ms	159.093 ms	161.527 ms

Table 15 shows that local Raspberry Pi verification completed in less than 0.2 seconds on average for both verified and tampered states. The low standard deviation values indicate stable repeated verification timing in the prototype edge validation.

6.8. Mobile Prototype Demonstration

The mobile prototype interface demonstrates how iMediFood-Shield can present prescription input, OCR confirmation, food/herb selection, and interaction results to a user. The interface screens were drawn in Figma [51] as a mobile app prototype interface for demonstrating the user workflow. The Figma prototype is not a fully deployed clinical application. The real OCR, recommendation, security verification, and Raspberry Pi validation components were implemented separately in the backend prototype.

Figure 11 shows the prescription-side workflow, including the home screen, prescription upload page, OCR processing view, and detected medication selection.

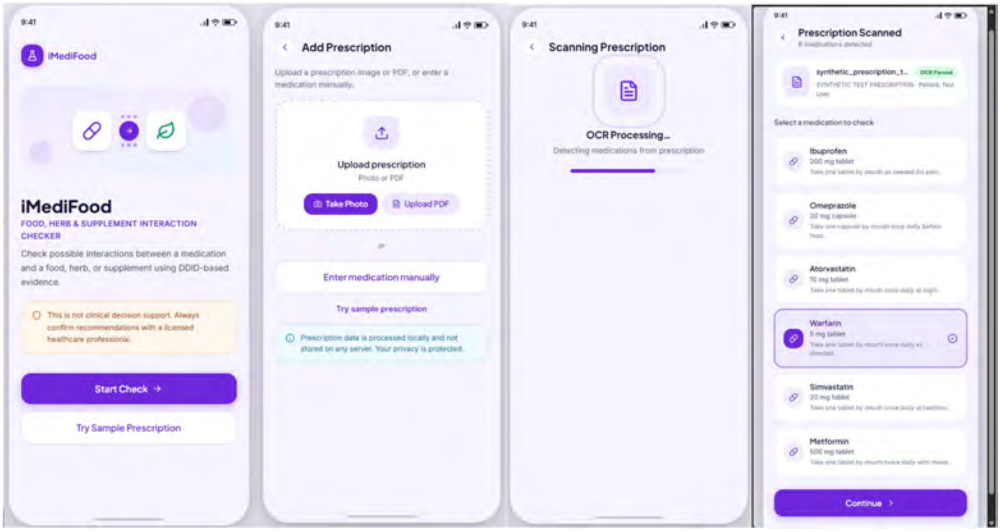

Figure 11. iMediFood-Shield Mobile Prototype Workflow Drawn in Figma for Prescription Upload, OCR Processing, and Medication Selection.

Figure 11 shows how the Figma prototype guides the user from starting a check to selecting a detected medication. The OCR flow is demonstrated using a synthetic prescription, and the medication is confirmed before interaction screening.

Figure 12 shows the food/herb selection and result-display screens.

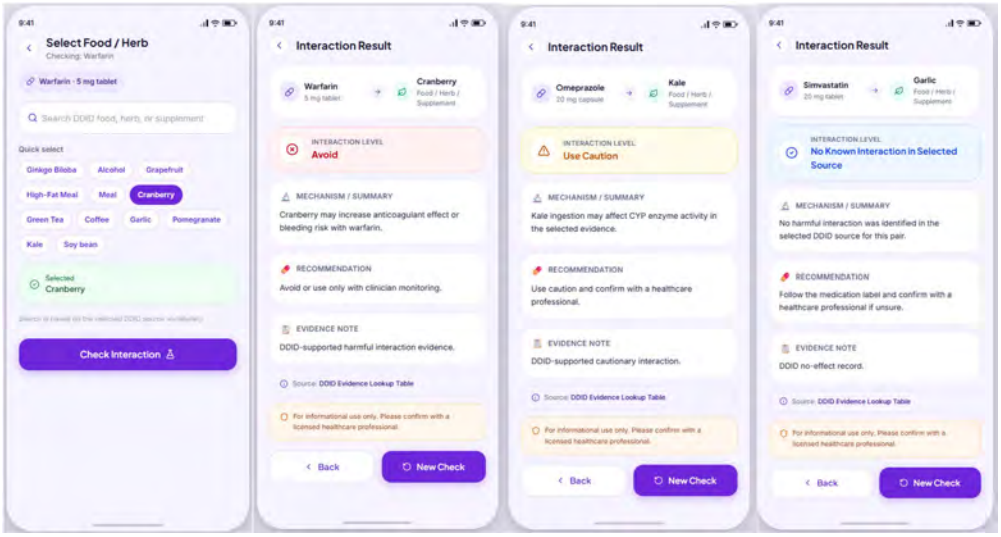

Figure 12. iMediFood-Shield Figma Prototype Screens for Food/Herb Selection and Interaction-Result Display.

Figure 12 shows representative output categories in the user interface, including "Avoid", "Use caution", and "No known interaction in selected source". The result cards also include evidence notes, recommendation text, source information, and a reminder to confirm recommendations with a licensed healthcare professional.

Table 16 summarizes representative mobile prototype demonstration cases. The cases cover avoid, caution, no-known-interaction, and insufficient-evidence outcomes.

Table 16. Representative Mobile Prototype Demonstration Cases.

Medication	Food/herb/supplement	Displayed result	Purpose in prototype
Ibuprofen	Ginkgo Biloba	Avoid	Demonstrates a high-warning interaction output.
Warfarin	Cranberry	Avoid	Demonstrates an avoid result in the mobile interface.
Omeprazole	Kale	Use caution	Demonstrates a cautionary DDID-supported interaction.
Atorvastatin	High-Fat Meal	Use caution	Demonstrates a food-related caution result.
Simvastatin	Garlic	No known interaction in selected source	Demonstrates a source-bounded no-known-interaction result.
Warfarin	Meal	No known interaction in selected source	Demonstrates no-known-interaction wording without claiming universal safety.
RandomDrugXYZ	RandomFoodXYZ	Insufficient evidence	Demonstrates the coverage guard for out-of-vocabulary inputs.

Table 16 shows that the prototype interface supports multiple user-facing outcomes. The "Insufficient evidence" case is included because unsupported inputs should not be forced through the AI model.

6.9. Overall Discussion

The combined results show that iMediFood-Shield addresses food–medication screening as an AI, safety, and security problem. The DDID-based dataset provides structured evidence for medication–food/herb pairs, while MedSafe-GATE v1 reduces false-safe predictions compared with the balanced AI baseline. The evidence lookup guard further prevents exact DDID-supported risky pairs from being overridden by model predictions. The security results show that all implemented tampering attacks were detected in the prototype evaluation, and the Raspberry Pi validation shows that local tamper-evident verification can run on resource-constrained edge hardware. The Figma mobile prototype demonstrates how the workflow can be presented to users, but the system remains a research prototype. This is not clinical decision support. Always confirm recommendations with a licensed healthcare professional.

7. Conclusion and Future Work

iMediFood-Shield presents a secure edge AI framework for food–medication interaction screening by combining DDID-based evidence lookup, RxOCR-Guard prescription processing, input coverage checking, MedSafe-GATE v1 safety-gated prediction, signed recommendation outputs, tamper-evident verification, Raspberry Pi edge validation, and a Figma-based mobile prototype interface. The DDID-based evaluation produced 23,950 cleaned row-level records and 17,693 unique medication–food/herb pairs. MedSafe-GATE v1 strict-screening mode achieved 90.99% accuracy and reduced the false-safe rate from 5.29% to 0.62%, lowering false-safe cases from 128 to 15. The evidence-first lookup guard blocked all 15 remaining exact-evidence false-safe cases, and the prototype security layer detected all five implemented tampering attacks. Raspberry Pi validation showed local verification times of 167.790 ms for verified cases and 159.709 ms for failed-verification cases over 30 runs.

Future work will extend iMediFood-Shield with additional curated food, herb, supplement, and medication interaction resources, stronger RxOCR-Guard validation using larger prescription-image datasets, improved medication-name recognition, and pharmacist-reviewed testing. Future security work will strengthen the current software HMAC/SHA-256 prototype with hardware-backed key storage, secure boot, rollback protection, device attestation, broader attack testing, and additional edge-device evaluations. The current system remains a research prototype. This is not clinical decision support. Always confirm recommendations with a licensed healthcare professional.

Acknowledgments

The authors acknowledge the use of OpenAI ChatGPT, GPT-5.5 Thinking, for grammar polishing, language editing assistance, LaTeX formatting support, and generation of the illustrative conceptual images used in Figure 1 and Figure 2 based on author-provided prompts. The research idea, methodology, dataset processing, Python OCR implementation, AI model development, security implementation, Raspberry Pi validation, result interpretation, and final manuscript decisions are the authors' original work. All AI-assisted text and figures were reviewed, edited, and approved by the authors, who take full responsibility for the manuscript content.

References

1. Centers for Disease Control and Prevention. FastStats: Therapeutic Drug Use. https://www.cdc.gov/nchs/fastats/drug-use-therapeutic.htm, 2026. Accessed: Jan. 20, 2026.
2. MedlinePlus. Drug Reactions. https://medlineplus.gov/drugreactions.html, 2026. Accessed: Jan. 31, 2026.
3. Bushra, R.; Aslam, N.; Khan, A.Y. Food Drug Interactions. *Oman Medical Journal* **2011**, *26*, 77–83. https://doi.org/10.5001/omj.2011.21.
4. Ased, S.; Wells, J.; Morrow, L.E.; Malesker, M.A. Clinically Significant Food-Drug Interactions. *The Consultant Pharmacist* **2018**, *33*, 649–657. https://doi.org/10.4140/TCP.n.2018.649.
5. U.S. Food and Drug Administration. Grapefruit Juice and Some Drugs Don't Mix. https://www.fda.gov/consumers/consumer-updates/grapefruit-juice-and-some-drugs-dont-mix, 2025. Accessed: Jan. 30, 2026.
6. Bailey, D.G.; Dresser, G.; Arnold, J.M.O. Grapefruit-Medication Interactions: Forbidden Fruit or Avoidable Consequences? *CMAJ* **2013**, *185*, 309–316. https://doi.org/10.1503/cmaj.120951.
7. Amadi, C.N.; Mgbahurike, A.A. Selected Food/Herb-Drug Interactions: Mechanisms and Clinical Relevance. *American Journal of Therapeutics* **2018**, *25*, e423–e433. https://doi.org/10.1097/MJT.0000000000000705.
8. Izzo, A.A.; Ernst, E. Interactions between Herbal Medicines and Prescribed Drugs: An Updated Systematic Review. *Drugs* **2009**, *69*, 1777–1798. https://doi.org/10.2165/11317010-000000000-00000.
9. Koziolek, M.; Alcaro, S.; Augustijns, P.; Basit, A.W.; Grimm, M.; Hens, B.; Hoad, C.L.; Jedamzik, P.; Madla, C.M.; Maliepaard, M.; et al. The Mechanisms of Pharmacokinetic Food-Drug Interactions: A Perspective from the UNGAP Group. *European Journal of Pharmaceutical Sciences* **2019**, *134*, 31–59. https://doi.org/10.1016/j.ejps.2019.04.003.
10. Posadzki, P.; Watson, L.; Ernst, E. Herb-Drug Interactions: An Overview of Systematic Reviews. *British Journal of Clinical Pharmacology* **2013**, *75*, 603–618. https://doi.org/10.1111/j.1365-2125.2012.04350.x.
11. Fugh-Berman, A. Herb-Drug Interactions. *The Lancet* **2000**, *355*, 134–138. Erratum in: Lancet 2000, 355(9208), 1020, https://doi.org/10.1016/S0140-6736(99)06457-0.
12. Rajkomar, A.; Dean, J.; Kohane, I. Machine Learning in Medicine. *New England Journal of Medicine* **2019**, *380*, 1347–1358, [https://www.nejm.org/doi/pdf/10.1056/NEJMra1814259]. https://doi.org/10.1056/NEJMra1814259.
13. Kelly, C.J.; Karthikesalingam, A.; Suleyman, M.; Corrado, G.; King, D. Key Challenges for Delivering Clinical Impact with Artificial Intelligence. *BMC Medicine* **2019**, *17*, 195. https://doi.org/10.1186/s12916-019-1426-2.
14. National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). Technical Report NIST AI 100-1, National Institute of Standards and Technology, 2023. Accessed on 26 Feb 2026, https://doi.org/10.6028/NIST.AI.100-1.
15. Harsha, C.S.S.; Mitra, A.; Mohanty, S.P.; Kougianos, E. Deepfake Audio Detection: Differentiating Modulated and Deepfake Audio. In Proceedings of the 2025 IEEE MetroCon, 2025, pp. 1–3. https://doi.org/10.1109/Metrocon66796.2025.11405771.
16. U.S. Food and Drug Administration. Cybersecurity in Medical Devices: Quality System Considerations and Content of Premarket Submissions, 2023. Accessed on 26 June 2026.

17. Halperin, D.; Heydt-Benjamin, T.S.; Ransford, B.; Clark, S.S.; Defend, B.; Morgan, W.; Fu, K.; Kohno, T.; Maisel, W.H. Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses. In Proceedings of the 2008 IEEE Symposium on Security and Privacy (sp 2008), 2008, pp. 129–142. https://doi.org/10.1109/SP.2008.31.
18. National Institute of Standards and Technology. Secure Hash Standard (SHS). Technical Report FIPS PUB 180-4, National Institute of Standards and Technology, 2015. Planning note dated 7 March 2023 states that NIST decided to revise FIPS 180-4; accessed on 26 June 2026, https://doi.org/10.6028/NIST.FIPS.180-4.
19. Sönmez Turan, M.; Brandão, L.T.A.N. Keyed-Hash Message Authentication Code (HMAC): Specification of HMAC and Recommendations for Message Authentication. NIST Special Publication 800-224 (Draft), National Institute of Standards and Technology, 2024. https://doi.org/10.6028/NIST.SP.800-224.ipd.
20. Diet–Drug Interaction Database. Diet–Drug Interaction Database, 2026. Accessed on 26 Feb 2026.
21. Hong, Y.; Xu, H.; et al. DDID: A Comprehensive Resource for Visualization and Analysis of Diet-Drug Interactions. *Briefings in Bioinformatics* **2024**, *25*, bbae212. https://doi.org/10.1093/bib/bbae212.
22. Rachakonda, L.; Mohanty, S.P.; Kougianos, E.; Sundaravadivel, P. Stress-Lysis: A DNN-Integrated Edge Device for Stress Level Detection in the IoMT. *IEEE Transactions on Consumer Electronics* **2019**, *65*, 474–483. https://doi.org/10.1109/TCE.2019.2940472.
23. Rachakonda, L.; Mohanty, S.P.; Kougianos, E. iLog: An Intelligent Device for Automatic Food Intake Monitoring and Stress Detection in the IoMT. *IEEE Transactions on Consumer Electronics* **2020**, *66*, 115–124. https://doi.org/10.1109/TCE.2020.2976006.
24. Veeramreddy, M.; Pradhan, A.K.; Ghanta, S.; Rachakonda, L.; Mohanty, S.P. NUTRIVISION: A System for Automatic Diet Management in Smart Healthcare, 2024, [arXiv:cs.CV/2409.20508].
25. Siripurapu, I.D.; Rachakonda, L.; Mohanty, S.P.; Kougianos, E. iLog 2.1: Calorie Estimation for Pizza via Mask R-CNN and Federated Learning. In Proceedings of the 2025 IEEE MetroCon, 2025, pp. 1–3. https://doi.org/10.1109/Metrocon66796.2025.11405824.
26. Siripurapu, I.D.; Mitra, A.; Mohanty, S.P.; Kougianos, E. iLog 3.0: Estimating Food Volume from 2D Images Using Mask R-CNN and Monocular Depth Estimation. In Proceedings of the 2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2025, Vol. 1, pp. 1–6. https://doi.org/10.1109/ISVLSI65124.2025.11130217.
27. Siripurapu, I.D.; Rachakonda, L.; Mohanty, S.P.; Kougianos, E. iLog 2.2: Volume and Nutrition Estimation for Mixed Foods via Mask R-CNN and Federated Learning. *Electronics* **2026**, *15*. https://doi.org/10.3390/electronics15071460.
28. Scheife, R.T.; Hines, L.E.; Boyce, R.D.; Chung, S.P.; Momper, J.D.; Sommer, C.D.; Abernethy, D.R.; Horn, J.R.; Sklar, S.J.; Wong, S.K.; et al. Consensus Recommendations for Systematic Evaluation of Drug-Drug Interaction Evidence for Clinical Decision Support. *Drug Safety* **2015**, *38*, 197–206. https://doi.org/10.1007/s40264-014-0262-8.
29. Grizzle, A.J.; Horn, J.; Collins, C.; Schneider, J.; Malone, D.C.; Stottlemyer, B.; Boyce, R.D. Identifying Common Methods Used by Drug Interaction Experts for Finding Evidence About Potential Drug-Drug Interactions: Web-Based Survey. *Journal of Medical Internet Research* **2019**, *21*, e11182. https://doi.org/10.2196/11182.
30. Hochheiser, H.; Jing, X.; Garcia, E.A.; Ayvaz, S.; Sahay, R.; Dumontier, M.; Banda, J.M.; Beyan, O.; Brochhausen, M.; Draper, E.; et al. A Minimal Information Model for Potential Drug-Drug Interactions. *Frontiers in Pharmacology* **2021**, *11*, 608068. https://doi.org/10.3389/fphar.2020.608068.
31. Medisafe. Download the App | Medisafe Medication Management. https://medisafe.com/download-the-app, 2026. Accessed: May. 30, 2026.
32. MyTherapy. Medication Reminder and Pill Tracker App. https://www.mytherapyapp.com, 2026. Accessed: May. 30, 2026.
33. MyFitnessPal. Calorie Tracker & BMR Calculator to Reach Your Goals. https://www.myfitnesspal.com, 2026. Accessed: Feb. 10, 2026.
34. Cronometer. The Most Accurate Nutrition Tracking App & Calorie Counter. https://cronometer.com/index.html, 2026. Accessed: May. 30, 2026.

35. Lose It!. Lose It! - Calorie Counting Made Easy. https://www.loseit.com/, 2026. Accessed: May. 30, 2026.
36. Drugs.com. Drug Interaction Checker - Find Unsafe Combinations. https://www.drugs.com/drug_interactions.html, 2026. Accessed: May 26, 2026.
37. Medscape. Drug Interactions Checker - Medscape Drug Reference Database. https://reference.medscape.com/drug-interactionchecker, 2026. Accessed: May 26, 2026.
38. Kim, B.Y.B.; Sharafoddini, A.; Tran, N.; Wen, E.Y.; Lee, J. Consumer Mobile Apps for Potential Drug-Drug Interaction Check: Systematic Review and Content Analysis Using the Mobile App Rating Scale (MARS). *JMIR mHealth and uHealth* **2018**, *6*, e74. https://doi.org/10.2196/mhealth.8613.
39. Ghubaish, A.; Salman, T.; Zolanvari, M.; Unal, D.; Al-Ali, A.; Jain, R. Recent Advances in the Internet-of-Medical-Things (IoMT) Systems Security. *IEEE Internet of Things Journal* **2021**, *8*, 8707–8718. https://doi.org/10.1109/JIOT.2020.3045653.
40. Tesseract OCR. Tesseract Documentation, 2026. Accessed on 26 Mar 2026.
41. He, H.; Garcia, E.A. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering* **2009**, *21*, 1263–1284. https://doi.org/10.1109/TKDE.2008.239.
42. Saito, T.; Rehmsmeier, M. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE* **2015**, *10*, 1–21. https://doi.org/10.1371/journal.pone.0118432.
43. Navarro, G. A guided tour to approximate string matching. *ACM Comput. Surv.* **2001**, *33*, 31–88. https://doi.org/10.1145/375360.375365.
44. Salton, G.; Buckley, C. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.* **1988**, *24*, 513–523. https://doi.org/10.1016/0306-4573(88)90021-0.
45. Cortes, C.; Vapnik, V. Support-Vector Networks. *Machine Learning* **1995**, *20*, 273–297. https://doi.org/10.1007/BF00994018.
46. Joachims, T. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In Proceedings of the Machine Learning: ECML-98. Springer, 1998, Vol. 1398, *Lecture Notes in Computer Science*, pp. 137–142. https://doi.org/10.1007/BFb0026683.
47. Niculescu-Mizil, A.; Caruana, R. Predicting good probabilities with supervised learning. In Proceedings of the Proceedings of the 22nd International Conference on Machine Learning, New York, NY, USA, 2005; ICML '05, p. 625–632. https://doi.org/10.1145/1102351.1102430.
48. Bellare, M.; Canetti, R.; Krawczyk, H. Keying Hash Functions for Message Authentication. In Proceedings of the Advances in Cryptology – CRYPTO '96; Koblitz, N., Ed., Berlin, Heidelberg, 1996; Vol. 1109, *Lecture Notes in Computer Science*, pp. 1–15. https://doi.org/10.1007/3-540-68697-5_1.
49. Bellare, M. New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In Proceedings of the Advances in Cryptology – CRYPTO 2006; Dwork, C., Ed., Berlin, Heidelberg, 2006; Vol. 4117, *Lecture Notes in Computer Science*, pp. 602–619. https://doi.org/10.1007/11818175_36.
50. Schneier, B.; Kelsey, J. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security* **1999**, *2*, 159–176. https://doi.org/10.1145/317087.317089.
51. Figma. Figma: The Collaborative Interface Design Tool. https://www.figma.com/, 2026. Accessed: Jun. 20, 2026.