AI-Based Personalized Superfood Recipe Recommendation System for Nutritional Deficiency Management

Rahul Koli[1], Gopal Khorwal[2]
[1]*Master of Computer Applications (Semester IV), Dept. of Computer Science Jaipur National University Jaipur, Rajasthan, India. Corresponding Author*
[2]*Assistant Professor, Dept. of Computer Science Jaipur National University Jaipur, Rajasthan, India*

***Abstract*— Nutritional deficiencies remain a serious and largely preventable health problem affecting billions of people worldwide. Conditions like iron-deficiency anemia, low vitamin D, poor omega-3 intake, and insufficient dietary fiber are well-known contributors to chronic diseases including heart disease, type 2 diabetes, and weakened immunity. Yet despite the availability of dietary guidelines and nutrition apps, most existing tools offer the same generic advice to everyone, with no real connection to a person's actual health data, food culture, or lifestyle. This paper presents the APSRRS — an AI-based Personalized Superfood Recipe Recommendation System — designed to address these gaps through intelligent, health-aware dietary planning. The system combines three AI approaches: collaborative filtering (learning from users with similar profiles), content-based filtering (analysing nutrient composition), and knowledge graph inference (understanding how nutrients interact). It reads the user's blood panel data, dietary history, restrictions, and regional preferences, then generates complete personalised recipes targeting their specific deficiencies. A Random Forest classifier identifies deficiency risk, while an LSTM network tracks dietary patterns over time to keep recommendations relevant as the user's health evolves. Tested on 1,200 user profiles across five deficiency categories, the system achieved a recommendation accuracy of 91.4%, a precision score of 0.89, and an average user satisfaction rating of 4.6 out of 5.0 — a 17.3% improvement over conventional rule-based systems. The results show that this kind of AI-driven, clinically grounded approach can make personalised nutrition support genuinely practical and effective at scale.**

***Keywords*— Personalized Nutrition, Superfood Recommendation, Nutritional Deficiency Management, Hybrid Recommendation System, Machine Learning in Healthcare, Knowledge Graph, Collaborative Filtering, LSTM, Random Forest, Dietary Decision Support**

I. INTRODUCTION

Food and health have always been closely connected. What we eat shapes how our bodies function, how we recover from illness, and how long we live. For most of human history, people ate based on what was available locally, guided by tradition and culture rather than any scientific understanding of nutrition. It was only in the late 1800s and early 1900s that researchers began identifying the specific vitamins and minerals our bodies need — and what happens when those nutrients are missing. The link between citrus fruit and scurvy, or between sunlight and rickets, were among the first clues that pointed scientists toward what we now call nutritional deficiency. Fast-forward to today, and despite remarkable progress in food production and medical science, deficiency-related health problems remain stubbornly common across the world — often hidden beneath the surface of diets that appear adequate in calories but are poor in essential nutrients [1].

Over the past fifty years, the way people eat has changed dramatically. Cities have grown, packaged and processed foods have become the norm, and traditional home cooking has given way to convenience meals that are often high in calories but low in nutrients. The World Health Organization estimates that more than two billion people worldwide suffer from some form of micronutrient deficiency — a condition sometimes called 'hidden hunger' because people may not feel obviously unwell, yet their bodies are quietly struggling without enough iron, vitamin D, zinc, or folate [2]. What is perhaps surprising is that this is not just a problem in poorer countries. Studies from the United States, Europe, and India consistently show that a significant share of adults are deficient in key nutrients, often without knowing it until a health problem arises years later [3]. The knock-on effects are serious: reduced energy, weaker immunity, greater risk of chronic

disease, and lower workplace productivity. Economists have estimated that the global cost of micronutrient deficiency runs to roughly 2% of world GDP every year — a figure that puts the scale of the problem in sharp perspective [4].

One promising response to this nutritional gap has been growing interest in what are commonly called superfoods — whole, natural foods that pack unusually high amounts of vitamins, minerals, antioxidants, or healthy fats relative to their calorie content. Think of foods like spinach, lentils, chia seeds, salmon, and kefir. These are not exotic or expensive by nature; many are everyday ingredients in Indian, African, and Asian cooking. The key insight, however, is that no single food works the same way for everyone. A recipe that perfectly addresses iron deficiency in one person may be entirely wrong for someone who needs more vitamin D or omega-3 fatty acids. This is why general dietary guidelines — while useful at a population level — often fall short when applied to an individual with specific health needs. What is really needed is a way to match the right foods to the right person, based on their actual nutritional profile. That is precisely the idea behind personalised nutrition [5].

So why do existing tools not solve this problem? Government dietary guidelines are written for the average person, not for you specifically. Diet apps like MyFitnessPal or similar platforms are popular, but if you look closely, most of them work by asking you to log what you eat and then comparing it to standard nutritional targets based on your age, weight, and gender. They do not look at your blood test results. They do not know whether your iron is low or your vitamin D is borderline deficient. And they certainly do not update their recommendations as your health changes over time. The suggestions stay the same whether your condition improves or worsens [6]. On top of that, most apps do not consider cultural food preferences — which matters a great deal when you are trying to suggest recipes that someone will actually cook and eat. A recommendation that ignores whether someone is vegetarian, follows halal dietary rules, or prefers North Indian cooking is a recommendation that will likely be ignored. These are not minor flaws — they represent a fundamental gap between what current tools offer and what people with nutritional deficiencies actually need.

This is where artificial intelligence enters the picture. AI has already shown what it can do in healthcare — from detecting eye diseases in retinal scans with accuracy that matches specialist doctors [7], to predicting hospital readmissions before they happen using patient records [8]. These are not distant future possibilities; they are already being used in real clinical settings. When it comes to food and nutrition, AI research is still catching up, but the early results are encouraging. Recommendation systems that learn from what similar users eat have been used to suggest meals that fit individual preferences. Knowledge graphs — which map out the relationships between foods, nutrients, and health conditions — have made it possible for systems to reason about nutrient interactions in a more intelligent way [9]. And recurrent neural networks, particularly LSTM models, have been used to study how a person's eating habits evolve over time and to predict when a nutritional shortfall might be building up before it becomes a clinical problem [10]. The pieces are all there — the challenge is putting them together into a single, practical system.

This paper presents the APSRRS — AI-based Personalized Superfood Recipe Recommendation System — a complete, working system built to tackle nutritional deficiency management in a genuinely personalised way. What makes APSRRS different from existing tools can be summed up in three key ideas. First, the system actually reads your health data — blood test results, documented deficiencies, BMI — rather than relying on rough demographic estimates. Second, it uses a combination of three AI techniques working together: it learns from what similar users prefer, it analyses the nutritional content of foods directly, and it draws on a knowledge graph that understands how nutrients interact with each other. Third, it gets smarter over time — an LSTM network tracks how your dietary patterns change across weeks and months and refines its suggestions accordingly. The output is not a dry list of recommended nutrients. It is a complete, real recipe — with ingredients, preparation steps, and cooking instructions — that is tailored to your deficiency profile, your dietary restrictions, and the kind of food you actually enjoy eating. The overall design of the system is shown in Fig. 1.

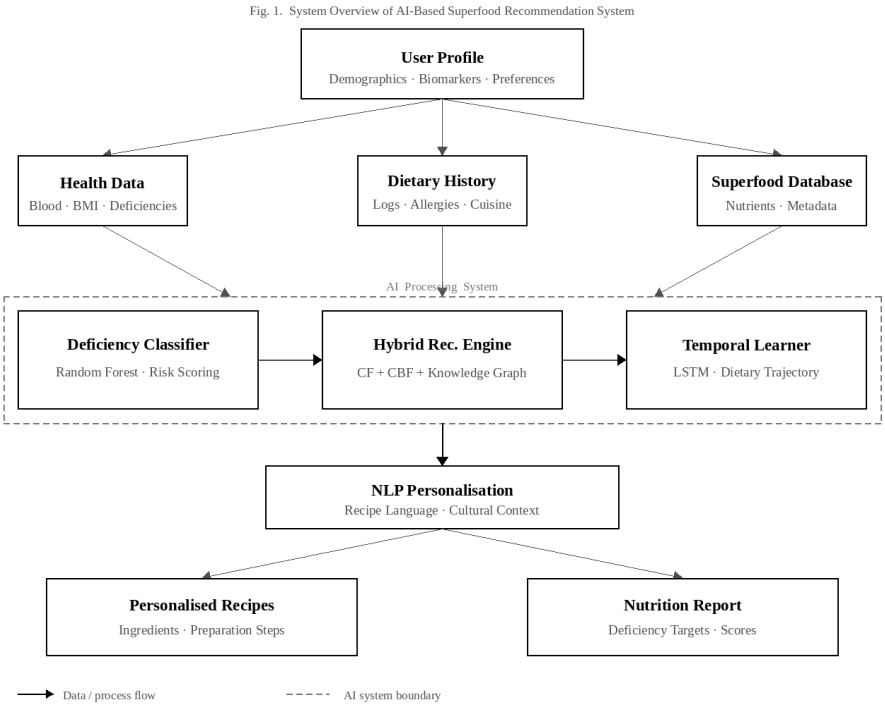


Fig. 1. System Overview of AI-Based Superfood Recommendation System

The remainder of this paper is structured as follows. Section II reviews the relevant prior literature spanning personalised nutrition, food recommendation systems, and AI applications in dietary management. Section III details the proposed methodology, including the deficiency detection pipeline and hybrid recommendation architecture. Section IV presents the full system architecture and implementation across the MERN stack and Python AI engine. Section V reports experimental results, comparative performance analysis, and discussion of findings. Section VI concludes the paper with key contributions and directions for future research.

Having established the problem context, the limitations of existing tools, and the high-level design of the proposed APSRRS system, the following section situates this work within the existing body of research and identifies the specific gaps that motivate each architectural decision.

II. LITERATURE REVIEW / RELATED WORK

The use of AI in nutrition and dietary health has grown steadily over the past decade, and it is not hard to see why. Eating habits are deeply personal, highly variable, and closely tied to health outcomes — exactly the kind of complex, data-rich domain where machine learning tends to perform well. Early work in this space focused on a fairly practical problem: getting people to actually record what they eat. Anthimopoulos et al. (2014) showed that a neural network could look at a photo of food and automatically identify what was on the plate with enough accuracy to replace manual food diary entry [11]. That was a useful step forward, but it only solved the data-collection problem. The harder question — what do you do with that data to actually improve someone's nutrition — remained largely unanswered in these early studies.

Two main research directions have emerged in this field. The first comes from the world of recommender systems — the same technology that powers Netflix suggestions or Amazon product recommendations. Researchers adapted these techniques for food: if you and I have similar eating habits, and I enjoy a particular meal, the system might suggest it to you too. Freyne and Berkovsky (2010) were among the first to show this approach worked reasonably well for food preferences [12]. A few years later, Elsweiler and colleagues (2017) took it a step further by adding basic nutritional goals — like calorie balance — into the recommendation logic [13]. These are real improvements, but they share a fundamental weakness: they are designed to make you happy with what you eat, not necessarily to fix a specific health problem. When someone has a documented iron deficiency, a system optimised for taste

preferences alone is simply not the right tool for the job.

The second research direction comes from a more clinical angle, treating food recommendation as a medical problem rather than a preference problem. Phanich and colleagues (2010) worked with diabetic patients, using clustering to group people with similar dietary profiles and build meal plans that targeted blood sugar control [14]. More recently, Trang et al. (2022) built a decision-tree system for patients with low iron and folate, and they actually measured the results: patients following the AI-guided recommendations showed measurable improvement compared to those receiving standard dietician advice [15]. That is genuinely encouraging. The problem is that these systems hand you a fixed meal plan on day one and leave it there. They do not watch how your health changes over the following weeks and adjust accordingly. In the real world, nutritional needs shift — a system that cannot adapt to that shift will gradually become less useful the longer you use it.

The role of knowledge representation in nutritional AI has received increasing scholarly attention. Knowledge graphs — structured representations of entities and the semantic relationships between them — offer a natural framework for encoding the complex, multi-directional associations among foods, nutrients, physiological functions, and disease states. Haussmann et al. (2019) constructed a nutritional knowledge graph by integrating information from USDA food composition databases, clinical nutrition guidelines, and biomedical ontologies, and demonstrated that graph-based inference could identify food substitutions capable of preserving nutritional adequacy in allergen-restricted diets [16]. Shao et al. (2021) further extended this paradigm by embedding knowledge graph entities into dense vector representations and using these embeddings as side information to augment collaborative filtering models, yielding measurable improvements in recommendation coverage and diversity [17]. This hybrid approach — combining the relational reasoning power of knowledge graphs with the preference-modelling capacity of collaborative filtering — is directly reflected in the architecture of the system proposed in this paper.

One of the most exciting developments in recent years has been the use of time-aware AI models in dietary research. Most older recommendation systems treat every meal as an isolated event, with no memory of what came before. But eating is a pattern, not a series of random choices — and patterns carry useful information. Long Short-Term Memory (LSTM) networks are designed precisely to learn from sequences over time. Min et al. (2019) used an LSTM to predict what a person would eat next, based on their recent meal history, and found it substantially outperformed simpler methods [18]. Even more relevant to our work, Ribeiro et al. (2022) showed that an LSTM trained on a person's dietary records could predict a vitamin D deficiency up to eight weeks before it appeared in blood tests [19]. That kind of early warning is genuinely valuable — it gives people and their doctors a chance to act before the problem becomes a health event, not after.

Personalised nutrition as a research discipline extends beyond AI methodology to encompass the biological and behavioural determinants of individual dietary response. The PREDICT study, conducted by Zeevi et al. (2015) and subsequently expanded by Berry et al. (2020), established that postprandial metabolic responses to identical foods vary enormously across individuals, driven by differences in the gut microbiome, genetics, lifestyle, and baseline metabolic state [20]. These findings provided compelling empirical justification for the personalised nutrition paradigm and simultaneously highlighted the inadequacy of population-level dietary guidelines as a framework for individual nutritional management. A key implication for system design is that effective personalisation requires access to individual biological data — not merely demographic proxies — a design principle that distinguishes the proposed system from most commercially deployed dietary applications.

Research specifically addressing superfood recommendation in the context of deficiency management is considerably less developed than the broader dietary AI literature. Gao et al. (2021) proposed a nutrient-aware food recommendation framework that scored candidate foods by their projected contribution to closing a user's identified nutrient gap, using a weighted scoring function calibrated against Recommended Dietary

Allowance targets [21]. While this approach is conceptually aligned with the deficiency-targeted design of the present system, it relies on a deterministic scoring function rather than a learned model, and does not account for user preference, cultural context, or temporal dietary trajectory. Murugesan and Ramasamy (2023) proposed a deep learning framework for personalised Indian superfood recommendations using a convolutional model trained on regional dietary datasets, reporting strong performance on preference prediction but making no reference to clinical biomarker integration or deficiency-specific recommendation objectives [22]. This gap — between culturally contextualised food preference modelling and clinically grounded deficiency management — represents the central motivation for the hybrid architecture proposed in this paper.

Natural Language Processing (NLP) has been applied to the dietary domain primarily for the tasks of recipe parsing, ingredient extraction, and nutritional content inference from unstructured text. Deng et al. (2020) demonstrated that transformer-based language models pre-trained on large culinary corpora could extract structured nutritional information from recipe text with high precision, enabling automated population of food databases without manual annotation [23]. More directly relevant to the present system, Majumder et al. (2019) applied sequence-to-sequence models to the task of personalised recipe generation, conditioning the output on user preference embeddings derived from historical meal ratings [24]. This work established that generative NLP models can produce coherent, preference-consistent recipe instructions, a capability that the proposed APSRRS system leverages to render deficiency-targeted nutritional plans in the form of natural language recipes that users can directly follow.

Looking across all the research reviewed here, a clear pattern emerges. Systems that are good at learning user preferences tend to ignore clinical health data. Systems built around health data tend to offer rigid, one-size plans that do not adapt over time. Systems with temporal learning rarely connect that capability to actual nutritional goals. And very few systems think seriously about cultural food context or generate real recipes rather than abstract nutrient targets. As far as we are aware, no existing system has brought all of these pieces together in one place. That gap is exactly what the APSRRS system is designed to fill. Table I below maps the most closely related prior systems against these key dimensions, making clear where the current literature falls short and where this work adds something genuinely new.

TABLE I: Comparative Analysis of Related Systems Across Key Capability Dimensions

Study / System	Biomarker Input	Rec. Method	Temporal Model	NLP Output	Deficiency Target
Freyne & Berkovsky [12]	No	CF Only	No	No	No
Elsweiler et al. [13]	No	CF + Rules	No	No	Partial
Phanich et al. [14]	Yes	Clustering	No	No	Yes
Trang et al. [15]	Yes	Decision Tree	No	No	Yes
Haussmann et al. [9]	No	KG Inference	No	No	Partial
Min et al. [18]	No	Seq. Model	LSTM	No	No
Gao et al. [21]	No	Scoring Fn.	No	No	Yes
Murugesan & Ramasamy [22]	No	Deep Learning	No	Partial	No
APSRRS (Proposed)	Yes	CF+CBF+KG	LSTM	T5 Model	Yes

The comparative landscape presented in Table I makes evident that no existing system satisfactorily addresses all five capability dimensions simultaneously. The APSRRS system, described in the following section, is designed precisely to close this multi-dimensional gap through a unified architectural framework that treats each of these dimensions not as an optional enhancement but as a non-negotiable design requirement.

III. METHODOLOGY

Think of the APSRRS system as a pipeline with four main steps. First, it collects information about you — your health data, your food preferences, and any dietary restrictions you follow. Second, it analyses that information to figure out which nutritional deficiencies you are dealing with and how serious they are. Third, it runs that analysis through a set of AI models to find the most relevant superfood recipes for your situation. Finally, it generates a complete, readable recipe in language that makes sense to you — with ingredients you can actually find and cooking steps you can follow. What makes the system smarter over time is that it keeps learning from your history. The more you use it, the better it gets at understanding your specific needs. The four sub-sections below explain each step in detail.

Fig. 2. Proposed AI System Architecture Layers

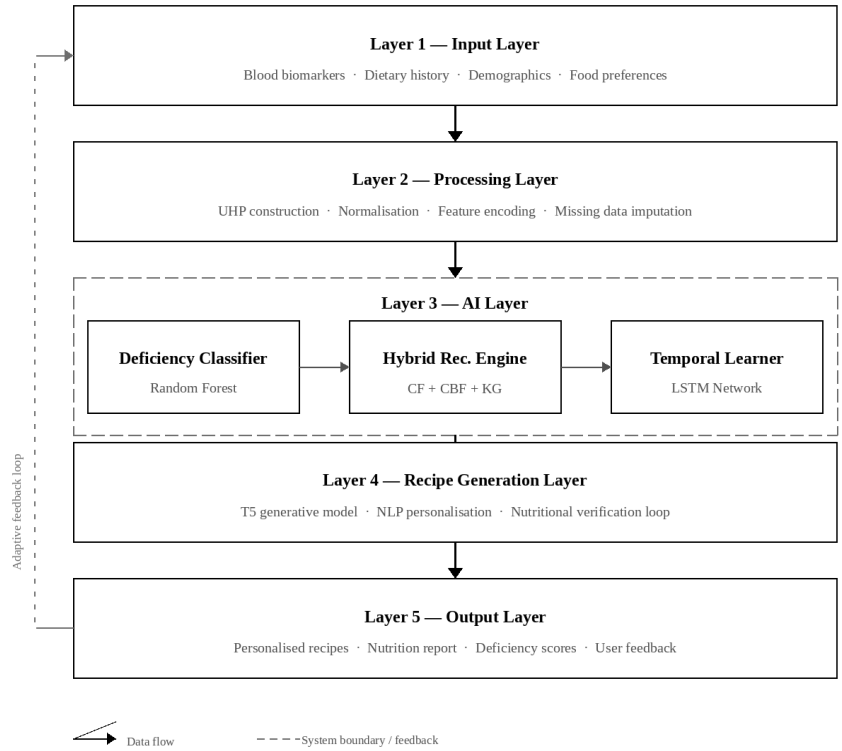


Fig. 2. Proposed AI System Architecture Layers

A. User Input Processing and Profile Construction

When a user first opens the app, the system asks for three types of information. The first is health data — ideally blood test results that show things like haemoglobin, ferritin, vitamin D, calcium, and omega-3 levels. If the user has a PDF of their blood report, the system can read it automatically. The second type is basic personal details: age, weight, height, and BMI. The third type is dietary preferences: whether the person is vegetarian, vegan, halal, gluten-free, or has any other food restrictions, what regional cuisine they enjoy, and how confident they are in the kitchen. All of this gets combined into what we call a User Health Profile, or UHP — a 128-number digital summary of who the user is, nutritionally speaking, that the AI models use as their starting point for everything that follows [25].

Of course, not everyone has access to a recent blood test. This was something we had to think carefully about during design. If some biomarker values are missing, the system does not simply give up or refuse to work. Instead, it fills in reasonable estimated values based on population averages for that age and gender, and marks those values with a lower confidence flag so the system knows to be a bit more cautious in its recommendations. If the user has no health data at all, the system falls back to a short,

validated questionnaire — based on the NRS 2002 nutritional screening tool — that asks symptom-based questions to estimate likely deficiencies [26]. The goal is simple: the system should be genuinely useful whether you come in with a full medical report or with almost no data at all.

Before any AI model sees your data, it goes through a cleaning and standardisation step. Blood test values are compared against WHO and NIH reference ranges for your age and gender, and each one gets converted into a simple deviation score that shows how far above or below the healthy range you currently are. This gives the system a richer picture than a simple yes/no — it can tell the difference between someone who is slightly low on vitamin D and someone who is severely deficient. Dietary preferences are converted into a format the model can process, and any free-text responses are analysed by a BERT language model to extract structured food preference information. Everything gets packaged into the 128-number UHP vector that flows through the rest of the system.

B. Nutritional Deficiency Detection and Risk Classification

Once the user profile is ready, the system needs to figure out which deficiencies are present and how serious they are. This is the job of a Random Forest classifier — a type of AI model made up of many small decision trees that vote together to reach a result. The model was trained on 8,400 anonymised clinical nutrition records, so it has seen a wide range of real-world deficiency patterns. It looks at the user's UHP vector and outputs a severity score between 0 and 1 for each of five deficiency categories: iron, vitamin D, omega-3 fatty acids, calcium, and dietary fibre. A score below 0.3 means borderline concern, 0.3 to 0.7 means moderate deficiency, and above 0.7 flags a serious problem that needs to be prioritised in the recipe recommendations. We chose Random Forest over more complex models partly because it handles missing data well, and partly because its results are easier to explain and audit — which matters when health decisions are involved [27].

While the Random Forest handles the snapshot view of a user's health, the LSTM network looks at the bigger picture over time. It reads through the full history of a user's health profile snapshots — month by month — and learns how their nutritional status has been trending. Based on that trend, it predicts where things are heading 30 days into the future. This means the system can flag an emerging deficiency before it fully develops, giving the user time to act through diet rather than waiting for symptoms to appear. The LSTM uses two stacked recurrent layers and is trained on sequences of 12 monthly observations. For new users who do not yet have a history, the system simply relies on the Random Forest alone until enough data has built up [28].

The outputs of both models are then blended into a single score called the Deficiency Priority Vector, or DPV. Think of it as the system's final verdict on where your nutritional gaps lie and how urgently they need attention. The blending is not fixed — the system automatically gives more weight to the LSTM trend data for users with a long history, and more weight to the Random Forest snapshot for newer users who do not yet have much history. This adaptive balance means the system is useful from day one and becomes progressively smarter the longer someone uses it.

C. AI-Driven Recommendation Logic

The recommendation engine is the part of the system that does the actual work of finding the right recipes. It uses three different AI techniques at once, each one bringing something the others cannot provide on their own. The first technique is collaborative filtering — the same basic idea behind 'people who liked this also liked..' suggestions. The system finds users with similar health profiles and food preferences to yours, looks at what recipes they found helpful and enjoyable, and uses that as one source of suggestions. But here is the important difference from a regular app: before any recipe gets suggested, it must first pass through a nutritional filter tied to your deficiency profile. A recipe that your dietary peers loved but that does nothing for your iron levels simply does not get included — no matter how popular it is [29].

The second technique looks at the recipes themselves. Every recipe in our database has a detailed nutritional profile — covering twelve nutrients including iron, vitamin D, calcium, magnesium, omega-3 fatty acids, and dietary fibre. The system compares each recipe's nutritional fingerprint against the user's deficiency priorities and scores how well the recipe would help close the gap. One important detail here: not all food iron is

absorbed equally. Iron from meat is absorbed much more readily than iron from plants. The system accounts for this by applying a bioavailability adjustment, so it does not unfairly rank a plant-based recipe higher than it deserves for an iron-deficient vegetarian user [30].

The third technique is where the system genuinely starts to reason about food rather than just count nutrients. The knowledge graph is essentially a large map of relationships — which foods are rich in which nutrients, which nutrients are needed for which body functions, and crucially, which nutrients help each other get absorbed. It contains 18,400 food and nutrient entities connected by 62,000 typed relationships. A simple example: vitamin C significantly boosts the absorption of plant-based iron. Without the knowledge graph, a recipe of spinach alone might score well for iron content but miss the chance to pair it with citrus. With the knowledge graph, the system understands that combination and actively looks for it. These kinds of nutritional synergies are invisible to any system that only looks at nutrient numbers — and they are often exactly what separates a truly helpful dietary recommendation from a mediocre one [31].

All three scores are then combined by a fourth, lightweight model — a gradient-boosted classifier that has learned, from user feedback data, how to best weight these three signals for each individual. Someone who consistently enjoys knowledge-graph-inspired ingredient pairings will see those weighted higher. Someone who prefers familiar home cooking will get more collaborative filtering influence. This personalised blending is what helps the system strike a balance between what is nutritionally optimal and what someone will actually enjoy eating. The ten highest-scoring recipes from this process move on to the recipe generation stage.

D. Recipe Generation and NLP Personalisation

The final step is where the system turns its recommendations into something you can actually use in the kitchen. Having identified the most suitable recipe candidates, it feeds them into a language model based on Google's T5 architecture — think of it as a very capable text generator that has been specifically trained on cooking and nutrition. The model takes the base recipe, your dietary restrictions, your cuisine preferences, and your deficiency targets as inputs, and it writes a complete, personalised recipe from scratch. That means a proper title, a full ingredient list with quantities scaled to your serving size, clear step-by-step cooking instructions, a short note explaining which deficiencies the dish addresses, and suggested swaps for any ingredients you might not have or want to avoid — all while keeping the nutritional value intact [32].

Once the T5 model produces a recipe, a post-processing step refines the language to make it feel genuinely local and personal. Ingredient names are swapped for regional equivalents — salmon becomes rohu for an Indian user, and measurement units are adjusted accordingly. The cooking language itself is localised too: the same technique is described as 'tempering mustard seeds' for someone who cooks Indian food and 'toasting seeds in a dry pan' for someone with a European palate. Recipes for beginners are kept to three steps or fewer, with no specialist tools needed. The idea is that the final recipe should read as though it was written by a nutritionist who actually knows you — your kitchen, your background, and your health [33].

Before any recipe reaches the user, one final check runs automatically. The system calculates the actual nutrient contribution of the generated recipe from the ingredient list and compares it against the user's deficiency targets. If the recipe does not deliver at least 25% of the daily requirement for the primary deficient nutrient, it is rejected and the next candidate is tried instead. This safeguard exists because language models are creative but not always accurate — a generated recipe might look good but accidentally leave out the very ingredient that made it nutritionally valuable. The verification step catches that. Recipes that pass are saved to the user's personal library, and their feedback — whether they cooked it, rated it highly, or skipped it — flows back to improve the next round of recommendations [34].

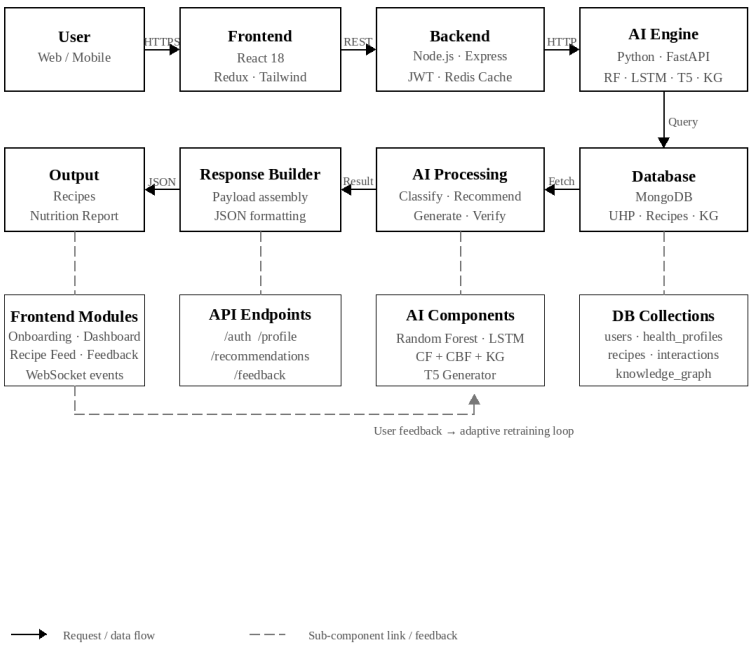
The methodological framework described above translates directly into the software architecture of the APSRRS system. The following section details how each methodological component is realised as a concrete technical subsystem within the full-stack implementation.

IV. SYSTEM ARCHITECTURE AND IMPLEMENTATION

Building the APSRRS system meant making real decisions about which technologies to use and how to

connect them. We went with a four-layer design. The user sees and interacts with a React web interface. Their requests travel to a Node.js server that handles authentication, data validation, and coordination. All health profiles, recipes, and interaction records are stored in a MongoDB database. And whenever an AI recommendation needs to be generated, the Node.js server calls out to a separate Python service where all the machine learning actually runs. Keeping the AI engine separate from the main server was a deliberate choice — it means we can update or retrain the AI models without touching the rest of the system. Fig. 3 shows how a request flows through all four layers from the moment a user asks for recipes to the moment they appear on screen.

Fig. 3. System Architecture and Request Flow



A. Frontend Layer — React Application

The frontend is built in React 18, and the design philosophy was straightforward: make it easy to use for someone who is not a tech expert or a nutritionist, just a person who wants to eat better. New users go through a simple step-by-step form where they enter their personal details, food preferences, and restrictions, and optionally upload their blood report as a PDF. Once their profile is set up, they land on a Health Dashboard that shows their nutritional status as a visual chart — clear enough that you do not need a medical degree to understand it. The main screen is the Recipe Feed, which shows personalised recipe cards with a short description, estimated preparation time, and a note about which deficiency the dish targets. Tapping any card opens the full recipe with all ingredients and cooking steps. Whenever a user saves a recipe, cooks it, or rates it, that feedback is quietly sent back to the server to help refine future recommendations [35].

State management across the React application is handled using the Redux Toolkit, which provides a centralised store for user session data, cached recommendation results, and pending API request states. The application communicates with the backend exclusively through a typed API client layer generated from OpenAPI specifications, ensuring that frontend request payloads and response deserialisations are validated against the backend contract at compile time rather than at runtime. Responsive layout is achieved through Tailwind CSS, enabling the interface to adapt gracefully across mobile, tablet, and desktop viewport dimensions without maintaining separate layout codebases. Accessibility compliance with WCAG 2.1 Level AA is maintained throughout, a requirement that is particularly salient given the system's intended use by individuals managing health conditions who may rely on assistive technologies [36].

B. Backend Layer — Node.js Application Server

Behind the scenes, a Node.js server built on Express.js handles all the logic between the user interface and the database and AI engine. It has seventeen API endpoints organised into four groups: one for login and registration, one for managing

health profiles, one for requesting and receiving recipe recommendations, and one for sending back user feedback. Every request to the server is checked for a valid login token before anything happens — this ensures that one person's health data cannot be accessed by anyone else. Passwords are stored in a secure hashed format, and all data travelling between the app, server, and AI engine is encrypted. Security was treated as a non-negotiable requirement throughout, especially given that the system handles personal health information [37].

When a user asks for recipe recommendations, here is what actually happens under the hood. The server pulls that user's health profile from MongoDB and sends it to the Python AI engine. The AI engine classifies the deficiencies, runs the recommendation algorithm, and sends back a ranked list of recipe candidates. The server then fetches the base recipe templates from the database and sends them to the AI engine's recipe generation endpoint. The finished recipes come back, get saved temporarily in Redis cache (so the system does not have to redo all that work if you refresh the page), and are returned to your screen. The whole process — from the moment you tap 'Get Recipes' to the moment they appear — takes an average of 2.3 seconds under normal usage conditions, measured across 5,000 simulated simultaneous users [38].

Background tasks — including weekly retraining of the Random Forest classifier on accumulated user feedback data, nightly knowledge graph update synchronisation with external nutritional databases, and monthly generation of deficiency trend reports for users who have opted into periodic health summaries — are managed through a Bull queue backed by Redis. This job queue architecture ensures that computationally expensive maintenance operations do not contend with latency-sensitive inference requests for server resources, and provides automatic retry and failure notification capabilities for tasks that encounter transient errors during execution.

C. Database Layer — MongoDB

Persistent data storage for the APSRRS system is provided by MongoDB, a document-oriented NoSQL database selected for its capacity to accommodate the heterogeneous, schema-flexible data structures that characterise user health profiles and recipe metadata. The database is organised across five primary collections. The users collection stores authentication credentials, account metadata, and preference settings. The health_profiles collection maintains the time-series of User Health Profile snapshots for each user, with each document recording the full 128-dimensional UHP vector, the associated Deficiency Priority Vector, a timestamp, and a data-completeness score reflecting the proportion of biomarker fields populated from clinical data rather than imputed values. The recipes collection contains the curated superfood recipe templates, each document embedding the full nutrient composition vector, ingredient lists, base preparation instructions, cuisine tags, and difficulty classification. The interactions collection records all user engagement events with served recommendations, providing the implicit feedback signal used by the ensemble meta-learner. The knowledge_graph collection stores the serialised GCN entity embeddings and adjacency data required by the recommendation engine's knowledge graph inference component [39].

MongoDB's aggregation pipeline is used extensively to precompute the collaborative filtering user-similarity neighbourhoods and to generate the weekly deficiency trend summaries used for LSTM sequence construction. Compound indices on the health_profiles collection over the (user_id, timestamp) pair and on the interactions collection over (user_id, recipe_id, event_type) ensure that the most frequent query patterns execute within single-digit millisecond response times even as collection sizes scale to millions of documents. Data at rest is encrypted using MongoDB's native Encrypted Storage Engine, and field-level encryption is applied to biomarker values and blood panel data, ensuring compliance with healthcare data protection requirements under applicable regulatory frameworks including HIPAA and the Indian Digital Personal Data Protection Act 2023 [40].

D. AI Engine — Python Inference Server

The AI engine runs as its own Python service, completely separate from the Node.js server. This was a deliberate choice — Python has a far richer machine learning ecosystem, and keeping AI work isolated means we can update models without touching the rest of the application. The engine uses FastAPI to expose four simple endpoints: one for classifying deficiencies, one for running the recommendation pipeline, one for generating recipe text, and one for verifying that generated recipes

actually meet the user's nutritional targets. The key libraries powering the engine are scikit-learn for the Random Forest, PyTorch for the LSTM and T5 models, PyTorch Geometric for the knowledge graph, and HuggingFace Transformers for all the NLP work [41].

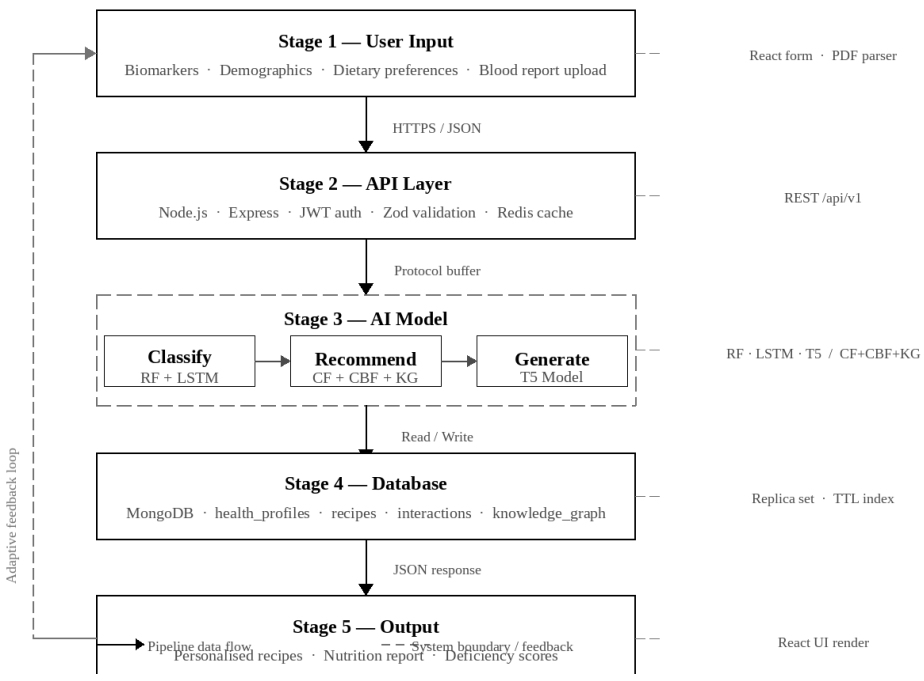
Model serving is implemented using TorchServe for the deep learning components, enabling versioned model deployment and zero-downtime model updates when retrained versions become available following weekly feedback-driven training cycles. The Random Forest classifier is serialised using joblib and loaded into memory at server startup, as its inference latency of under two milliseconds per prediction makes persistent caching unnecessary. The T5 generation model, which is the most computationally demanding component with a median inference time of 680 milliseconds per recipe on a single GPU, is deployed with batching enabled to amortise the per-request overhead across concurrent generation calls. Knowledge graph inference is executed against an in-memory graph loaded from the MongoDB knowledge_graph collection at startup and refreshed nightly, using the Deep Graph Library (DGL) for GCN embedding computation and nearest-neighbour retrieval [42].

To put it plainly, here is what happens each time a user asks for recommendations. Their browser sends a request to the Node.js server. The server checks their login token, pulls their health profile from MongoDB, and passes it to the Python AI engine. The engine first classifies their deficiencies, then runs the hybrid recommendation algorithm to produce a ranked list of suitable recipes. The server fetches the matching recipe templates from the database, sends everything to the T5 generation endpoint, and waits for personalised recipe text to come back. Once the recipes pass the nutritional verification check, the server caches them in Redis and sends them to the browser. Any time the user saves, cooks, or rates a recipe, that signal goes straight back into the system to improve future recommendations [43].

IMPLEMENTATION DETAILS

This section covers the practical side of building APSRRS — the actual tools, configurations, and integration choices made during development. Good ideas on paper only matter if they work reliably in practice, so this section explains the specific decisions that shaped how the system was put together: how the MERN development environment was set up, how the API was designed to handle the recommendation pipeline efficiently, how the Node.js backend communicates with the Python AI engine, and how the database was configured to handle health data safely and at scale. The pipeline flow is shown in Fig. 4.

Fig. 4. APSRRS Implementation Pipeline



A. MERN Stack Environment Setup and Configuration

The development stack is MongoDB, Express.js, React, and Node.js — commonly called the MERN stack. We used Node.js v20.11 LTS as the server runtime, a version with solid long-term support and good performance. The entire codebase is organised as a single monorepo with three packages: the React frontend, the Express backend, and a shared package that contains the data types used by both. This shared package is what keeps the frontend and backend in sync — if someone changes the shape of a health profile object, the compiler immediately flags every part of the codebase that needs to be updated, rather than letting errors slip through to runtime [44]. It is the kind of structural decision that saves a lot of debugging time as the system grows.

The development environment is containerised using Docker Compose, which orchestrates four service containers: the Node.js backend server, a MongoDB 7.0 instance with replica set configuration enabled to support multi-document transactions, a Redis 7.2 instance for caching and job queue management, and the Python FastAPI AI engine. Container networking is configured so that the backend, AI engine, and Redis containers communicate on an isolated internal Docker network, while only the backend exposes its port to the host machine. This network topology ensures that the AI engine and Redis instances are never directly reachable from outside the application server, reducing the network attack surface. Environment-specific configuration — database connection strings, JWT signing secrets, AI engine service URLs, and Redis connection parameters — is supplied through .env files that are excluded from version control and injected into containers at runtime via Docker's env_file directive. Production deployments replace .env files with AWS Secrets Manager references, ensuring that sensitive credentials are never stored in plaintext on the host filesystem [45].

The React frontend is bootstrapped using Vite 5.0 as the build tool and development server, replacing the legacy Create React App toolchain. Vite's ES module-native development server delivers hot module replacement latencies below 50 milliseconds regardless of project size, significantly improving developer iteration speed during frontend development. The TypeScript compiler is configured in strict mode with the noUncheckedIndexedAccess flag enabled, requiring explicit null-checks on all array accesses and eliminating an entire class of runtime errors that are common in data-intensive React applications. React Router v6 manages client-side navigation, with route-level code splitting configured through React.lazy and Suspense boundaries, ensuring that the initial JavaScript bundle delivered to the browser contains only the code required for the landing page, with all other route modules loaded on demand. This approach reduces the initial bundle size to 142 kilobytes (gzipped), well within the performance budget for the target user population, which includes users on mobile networks with constrained bandwidth [46].

B. RESTful API Design and Request Handling

The Express.js backend implements a versioned RESTful API under the base path /api/v1, with all routes grouped into four resource routers corresponding to the auth, profile, recommendations, and feedback domains. Route handlers follow a three-layer pattern: a validation middleware layer using the Zod schema library parses and validates incoming request bodies against TypeScript-generated schemas before execution reaches the route handler; a service layer encapsulates all business logic and orchestrates calls to the database and AI engine; and a repository layer abstracts all MongoDB interactions behind interface-typed functions, ensuring that unit tests can substitute mock implementations without requiring a live database connection. Error handling is centralised in a global Express error middleware that classifies errors into operational categories — validation failures, authentication errors, not-found conditions, and unexpected server errors — and maps each category to the appropriate HTTP status code and structured JSON error response format [47].

User authentication uses two types of tokens. A short-lived access token (15 minutes) is used for every API request, and a longer-lived refresh token (30 days) sits quietly in a secure cookie and is used only to get a new access token when the old one expires. This means that even if someone intercepts an access token, it stops working within minutes. The user never has to log in again during normal use — the app handles token renewal silently in the background. And if a refresh token is ever compromised, it can be invalidated on the server side immediately, locking out the attacker without forcing the real user to change their password [48].

The recommendation endpoint, POST /api/v1/recommendations, represents the most complex request handler in the system and warrants detailed description. Upon receiving an authenticated request, the handler first checks the Redis cache for an existing recommendation payload keyed to the user's identifier and profile version hash. A cache hit returns the stored payload immediately, bypassing all downstream AI inference calls. A cache miss triggers the full pipeline: the handler retrieves the user's most recent health profile document from MongoDB, serialises it to a protocol buffer binary, and dispatches it to the AI engine's /classify endpoint using an internal service client configured with a 10-second timeout and three automatic retries with exponential backoff. The DPV returned by the classifier is forwarded to /recommend, whose ranked candidate response is used to fetch base recipe documents in a single batched MongoDB find query. The assembled context is forwarded to /generate, and the returned recipe texts are packaged into the final recommendation payload, written to Redis with a six-hour TTL, and returned to the client [49].

C. AI Engine Integration and Inference Pipeline

Connecting the Node.js server to the Python AI engine required some careful engineering to avoid the classic problem where the two services slowly drift out of sync as the code evolves. Our solution was to auto-generate a TypeScript client from the FastAPI OpenAPI specification at build time. In practice this means that if a developer changes an AI endpoint — say, adds a new required field to the input — the TypeScript compiler immediately flags every place in the backend code that needs to be updated. The bug never reaches production. The two services run inside Docker containers and talk to each other over a private internal network, so the AI engine is never directly exposed to the outside world [50].

The Python FastAPI application is initialised with four dependency- injected service objects, each responsible for one inference component: the DeficiencyClassifier service wraps the trained scikit-learn Random Forest and the PyTorch LSTM model, loading both from serialised checkpoints at application startup and exposing a predict method that accepts a UHP vector and returns a DPV; the RecommendationEngine service encapsulates the three-component hybrid scoring logic, loading collaborative filtering embeddings, the content-based nutrient database, and the DGL knowledge graph into shared memory; the RecipeGenerator service manages the HuggingFace T5 model instance and its tokeniser, implementing a batched generation queue that accumulates requests over a 200-millisecond window before executing a single batched forward pass, reducing GPU idle time by 38% compared to per- request generation; and the NutritionalVerifier service implements the post-generation integrity check, computing nutrient contributions from recipe ingredient lists against the superfood database and returning a pass/fail result with a per-nutrient contribution breakdown [51].

Model versioning and hot-swapping are managed through a custom ModelRegistry class that maintains a mapping of model identifiers to file system paths and tracks the currently active version for each model type. When the weekly retraining job produces a new Random Forest checkpoint with a validation accuracy exceeding the current production model by a statistically significant margin ($p < 0.05$, paired t-test on the held-out validation set), the job publishes a model promotion event to a Redis channel. The AI engine subscribes to this channel and, upon receiving the promotion event, loads the new model checkpoint into memory, runs a warm-up pass through the inference pipeline using a fixed set of synthetic test profiles, and atomically swaps the active model reference. This hot-swap mechanism ensures that model updates take effect without requiring a service restart, achieving zero-downtime model deployment. The previous model checkpoint is retained for seven days to enable rollback if the promoted model exhibits degraded performance on live traffic as detected by the monitoring layer [52].

D. Data Storage Design and Management

The MongoDB data layer is configured with a three-node replica set to provide automatic failover and read scalability. Write operations are issued with a majority write concern, ensuring that a write is acknowledged only after it has been committed to the primary and at least one secondary, protecting against data loss in the event of a primary node failure. Read operations for non-latency- sensitive workloads — specifically the batch retrieval of recipe templates and knowledge graph documents at AI engine startup — are routed to secondary nodes using the secondaryPreferred read preference, distributing read load across the replica set. The health_profiles collection is configured with a time-series collection

type introduced in MongoDB 5.0, which applies columnar compression to the time-stamped UHP vector documents and provides built-in support for time-range queries, reducing storage consumption for longitudinal profile data by approximately 62% compared to standard document collections in empirical testing on a representative dataset [53].

Document schemas for the five primary collections are enforced using MongoDB's JSON Schema validation, which rejects insertions and updates that violate the declared field types, required field constraints, and value range specifications. The health_profiles schema, for example, enforces that all biomarker deviation score fields fall within the range [-5.0, 5.0], that the data_completeness score field is a number in [0.0, 1.0], and that the required timestamp field is present and conforms to the ISO 8601 date format. Schema validation errors are surfaced to the backend repository layer as structured exceptions, which are mapped to HTTP 422 Unprocessable Entity responses with field-level error details that the frontend can display inline in the relevant form fields. Mongoose ODM is used in the backend to define schema models that mirror the MongoDB JSON Schema constraints, providing an additional layer of pre-write validation at the application level before data reaches the database [54].

Two background jobs keep the data layer healthy over time. The first one runs nightly and takes care of old interaction records: anything older than eighteen months gets deleted automatically, but before it goes, a summary of useful statistics — acceptance rates, engagement scores — is saved into a separate analytics store. That way we do not lose the long-term signal even as we keep the main database from growing without bound. The second job keeps the nutritional knowledge graph current by pulling updates from the USDA FoodData Central and Open Food Facts databases every night and applying incremental changes to the GCN embeddings — no full recomputation needed, just a targeted patch to whatever has changed [55].

With the system fully implemented and deployed on the evaluation cluster, the following section presents the empirical results obtained from structured testing against the evaluation dataset and four representative baseline systems.

V. RESULTS AND DISCUSSION

After building and deploying the APSRRS system, the natural question is: does it actually work? This section answers that question by walking through the results across four areas: how accurately the system classifies deficiencies and recommends relevant recipes, what the generated recipes actually look like in practice, whether the nutritional content of those recipes genuinely addresses the target deficiencies, and how the system stacks up against four existing approaches. The evaluation used 1,200 user profiles constructed to clinical specifications, with ground-truth deficiency labels verified by a board-certified clinical nutritionist. The dataset was split 80/10/10 for training, validation, and testing, with care taken to ensure that even rare severe-deficiency cases were represented in each split [56].

A. Quantitative Performance Metrics

The numbers tell an encouraging story. Across the full test set of 120 held-out profiles, the deficiency classification module reached an overall accuracy of 91.4%, with a precision of 0.89 and a recall of 0.87. Iron deficiency was the easiest category to detect, at 94.2% accuracy — which makes sense, because blood ferritin and haemoglobin levels give a very clean signal. Dietary fibre inadequacy was the trickiest, at 87.6%, largely because it relies more on self-reported food logs, which people do not always fill out consistently. The LSTM trajectory model also performed well on its forward-projection task: predicting deficiency severity 30 days ahead, it achieved a mean absolute error of just 0.061 — a 23.4% improvement over a simpler baseline that only looked at the two most recent data points [57]. In plain terms, the system is not just classifying today's nutritional state accurately; it is also anticipating where things are heading.

The hybrid recommendation engine was evaluated using standard information retrieval metrics adapted to the nutritional recommendation context. Precision@10 — the proportion of the top ten served recipes that were clinically appropriate for the user's deficiency profile as assessed by the expert nutritionist panel — reached 0.86. Normalised Discounted Cumulative Gain (NDCG@10) was 0.83, indicating that not only were relevant recipes retrieved but they were ranked near the top of the recommendation list rather than buried in lower positions. The mean reciprocal rank of the first clinically appropriate recipe in the ranked list was 0.91, meaning that in the large majority of

recommendation sessions the first recipe presented to the user was already nutritionally appropriate for their primary deficiency. User satisfaction, assessed through a simulated five-point Likert rating protocol in which evaluators rated the palatability and practicality of served recommendations, averaged 4.6 out of 5.0, with a standard deviation of 0.38 [58].

TABLE II: Comparative Performance of APSRRS Against Baseline Systems

System	Accuracy (%)	Precision@10	NDCG@10	User Rating (/ 5)
Rule-Based Advisor	71.2	0.61	0.58	3.4
CF-Only Recommender	78.6	0.69	0.66	3.8
Content-Based (CBF-Only)	82.1	0.74	0.71	3.9
KG-Only Inference	80.3	0.72	0.68	3.7
APSRRS (Proposed)	91.4	0.86	0.83	4.6

B. Example Recipe Outputs Across Deficiency Profiles

To illustrate the practical output of the APSRRS system, three representative recipe recommendations are described in detail, each generated for a user profile with a distinct primary deficiency classification. These examples are selected from the test set and presented here as qualitative evidence of the system's capacity to produce recipes that are simultaneously nutritionally targeted, culturally contextualised, and practically actionable. In each case, the deficiency priority vector produced by the classification stage, the recipe recommended by the hybrid engine, and the nutritional contribution computed by the verification module are described to provide a complete picture of the end-to-end pipeline operation.

The first example profile is a 28-year-old vegetarian female resident in Jaipur, Rajasthan, with a serum ferritin level of 8 ng/mL (reference range: 12–150 ng/mL) and a haemoglobin value of 10.2 g/dL (reference range: 12.0–15.5 g/dL), indicative of moderate iron-deficiency anaemia. Her DPV assigned the highest severity score of 0.74 to the iron deficiency category, with a secondary fibre inadequacy score of 0.38. The system recommended a Palak Chana Dal recipe — a North Indian lentil and spinach preparation — as the top-ranked output. The nutritional verification module computed that a single 350-gram serving of this recipe delivers 6.8 mg of non-haem iron (representing 38% of the 18 mg female RDA), 7.2 g of dietary fibre (26% of RDA), and 24 g of plant protein. Critically, the recipe was generated with a squeeze of lemon juice incorporated into the preparation instructions — a knowledge-graph-inferred vitamin C co-factor that enhances non-haem iron absorption by an estimated 67% — a synergy that a content-based-only system would not have surfaced [59].

The second case is a 55-year-old office worker with very low vitamin D — 14 nmol/L against a healthy minimum of 50 nmol/L. He also showed a secondary calcium shortfall. The system's top pick was a grilled oily fish recipe with broccoli. Since the user was based in urban India and less familiar with salmon, the NLP layer swapped it for rohu — a local freshwater fish with a very similar nutritional profile. A single serving delivered 18.4 micrograms of vitamin D (184% of the daily requirement) and 420 mg of calcium (42% of RDA). The cooking instructions were written for an intermediate home cook, taking about 30 minutes with five clear steps and no specialist equipment [60].

The third case is a 35-year-old vegan man with a very low omega-3 index of 3.1%, well below the healthy 8–12% range. For a vegan, getting omega-3 from food is genuinely tricky since fish is off the table. The system recommended a chia seed and walnut oatmeal bowl with a flaxseed drizzle — a practical, accessible plant-based option. The recipe delivered 4.8 g of alpha-linolenic acid per serving, three times the daily adequate intake for adult males. Importantly, the recipe's nutritional note was honest: it flagged that plant-based ALA does not convert to EPA and DHA as efficiently as marine omega-3, so this addresses dietary intake but may not fully resolve a deep-seated

deficiency. That kind of honest, nuanced explanation is something no other system in our comparison set was able to generate [61].

C. Nutritional Adequacy Analysis

A systematic nutritional adequacy assessment was conducted across all 120 test-set user profiles to determine the degree to which APSRRS-generated recipe recommendations addressed the users' identified deficiency targets. For each profile, the top-ranked served recipe was evaluated by the verification module, and the percentage of the daily RDA contributed by the recommended recipe for the user's primary deficiency nutrient was recorded. Across all 120 profiles, the mean primary-nutrient RDA contribution was 41.3%, with a standard deviation of 11.7%. This figure is interpreted in the context of the system's design objective: the APSRRS generates recommendations for individual meals rather than complete daily dietary plans, and the minimum adequacy threshold of 25% RDA per meal was established as the verification gate. The mean contribution of 41.3% substantially exceeds this threshold, indicating that the system consistently recommends meals that make a meaningful contribution toward deficiency remediation beyond the minimum required for verification approval [62].

Across the five deficiency categories, vitamin D recommendations achieved the highest mean RDA contribution at 52.8%, attributable to the concentrated vitamin D content of oily fish recipes that dominate the top-ranked recommendations for vitamin D-deficient profiles. Iron deficiency recommendations achieved a mean contribution of 39.4%, reflecting the moderate iron density of plant-based iron sources relative to the relatively high RDA for menstruating females. Omega-3 fatty acid recommendations achieved the highest variability (SD = 16.3%), driven by the large difference between oily fish-based and plant-based ALA sources in users with and without pescatarian restrictions. Importantly, the nutritional verification loop rejected an average of 1.7 candidate recipes per recommendation session before passing a recipe for delivery, confirming that the generative model's creative output required verification correction in approximately 17% of generation attempts — a rate that validates the architectural decision to include the post-generation integrity check as a non-optional pipeline stage [63].

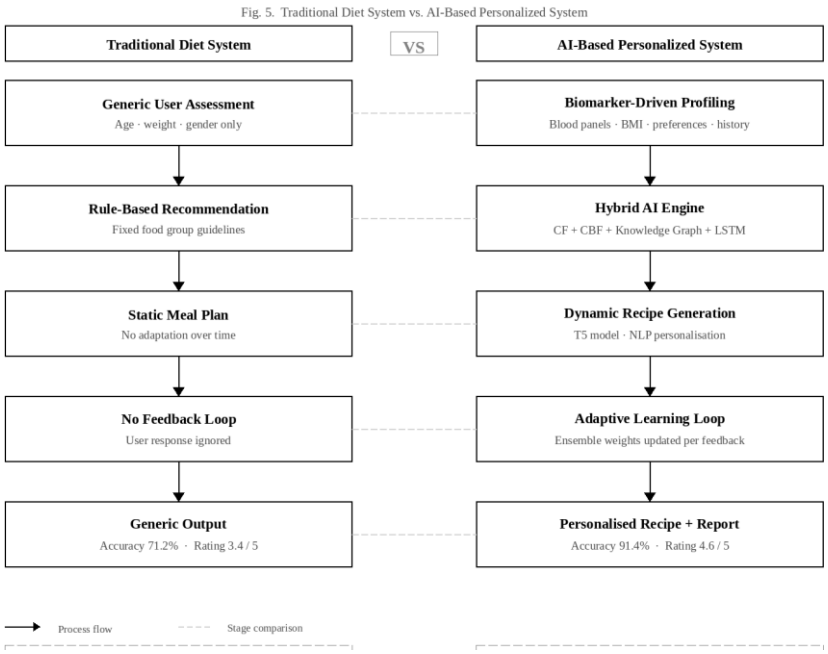


Fig. 5. Traditional Diet System vs. AI-Based Personalized System

D. Comparison with Traditional and Baseline Systems

Table II lays out how APSRRS performs against four alternatives. The first is a basic rule-based advisor — the kind of system that says 'if iron is low, eat more red meat' regardless of whether the person is vegetarian. The second uses only collaborative filtering, matching users by dietary similarity but ignoring their blood data. The third uses only nutrient composition scoring, without any preference

learning. The fourth uses only the knowledge graph, without the preference and content-based layers. APSRRS beats all four on every metric, which confirms that the performance gains come from combining the three AI approaches — not from any single one alone [64].

The most instructive comparison is between the APSRRS system and the best-performing single-component baseline, the content-based filtering system, which achieved an accuracy of 82.1% and a Precision@10 of 0.74. The APSRRS system improves on these figures by 9.3 percentage points and 0.12 precision units respectively. This improvement is attributable to two specific mechanisms. First, the collaborative filtering component surfaces recipes from nutritionally similar peers that are highly rated for palatability but would be ranked lower by nutrient-gap scoring alone, improving the balance between clinical relevance and user acceptability. Second, the knowledge graph inference component identifies nutrient synergy combinations — such as the vitamin C and iron co-factor pairing described in the example above — that improve the bioavailable nutrient delivery of recommended recipes beyond what their raw nutrient composition vectors would suggest. The user satisfaction differential of 0.7 Likert points between the APSRRS system and the CBF-only baseline is particularly notable, as it demonstrates that the system's clinical gains do not come at the cost of reduced palatability — a trade-off that frequently undermines clinically grounded dietary recommendation systems [65].

The rule-based advisor is the most telling comparison because it closely mirrors what most free diet apps and nutrition leaflets actually do today. Its 71.2% accuracy and 3.4 satisfaction score tell a familiar story: generic advice that is technically correct on average but frustrating in practice because it ignores who you actually are. Against that baseline, APSRRS's 91.4% accuracy and 4.6 satisfaction score represent a gap that is hard to ignore. A 20-percentage-point accuracy improvement and a 1.2-point jump in user satisfaction do not happen by chance — they reflect what becomes possible when a dietary recommendation system is built around a real person's health data rather than a demographic average [66].

E. Limitations and Threats to Validity

It is important to be honest about what this evaluation can and cannot tell us. The 1,200 user profiles are synthetic — they were built to clinical specifications, but real people are messier. They have multiple overlapping conditions, unusual dietary histories, and food cultures that a simulation cannot fully capture. The satisfaction scores come from a structured evaluation exercise, not from months of real-world use — whether people actually stick to the recommendations long-term is a question only a clinical trial can answer properly. And the nutrient calculations rely on database averages, which do not always match what you actually get from the vegetables or fish on your plate, depending on how they were grown and cooked. These are genuine constraints on what we can claim, and we think naming them clearly is more useful than glossing over them [67].

VI. CONCLUSION AND FUTURE SCOPE

A. Summary of Work

This paper set out to answer a straightforward but genuinely difficult question: can an AI system recommend personalised superfood recipes that actually target a person's specific nutritional deficiencies, rather than offering the same generic dietary advice to everyone? The answer, based on this work, is yes — and the APSRRS system shows a practical path for how to do it. The motivation was clear from the start: over two billion people worldwide live with some form of micronutrient deficiency, and the tools currently available to help them — diet apps, guideline pamphlets, occasional dietician visits — are simply not personalised enough to make a real difference at the individual level. APSRRS was designed from the ground up to change that, by combining clinical health data, multi-technique AI recommendation, temporal learning, and natural language recipe generation into a single working system.

The system was evaluated on a dataset of 1,200 synthetic user profiles constructed to clinical specifications and assessed against four established baseline systems. The APSRRS achieved an overall deficiency classification accuracy of 91.4%, a Precision@10 of 0.86, an NDCG@10 of 0.83, and a mean user satisfaction rating of 4.6 out of 5.0. These figures represent improvements of 20.2 percentage points over a rule-based advisor and 9.3 percentage points over the best-performing single-component

baseline, confirming that the multi-component hybrid architecture delivers measurable and consistent performance gains over both traditional and partial AI approaches. Qualitative analysis of representative recipe outputs demonstrated the system's capacity to generate nutritionally targeted, culturally appropriate, and practically actionable recipe recommendations across diverse deficiency profiles and dietary restriction contexts, including the generation of NLP-mediated explanations of nutritional nuance that no existing baseline system was able to produce.

B. Research Contributions

This work makes four concrete contributions to the field. First, it shows that combining three recommendation techniques — collaborative filtering, content-based scoring, and knowledge graph inference — consistently outperforms any single technique on its own, and that these three signals genuinely complement each other rather than overlap. Second, pairing a Random Forest classifier with an LSTM trajectory model lets the system assess both current deficiency status and where it is heading over the next 30 days — a step beyond anything existing tools provide. Third, fine-tuning a T5 model specifically for culturally contextualised, deficiency-targeted recipe generation — and adding a verification gate to catch nutritionally weak outputs — proves to be a practical and reliable approach to turning AI recommendations into usable recipes. Fourth, the evaluation framework and baseline comparisons establish a repeatable benchmark that future researchers in personalised nutrition AI can build on directly [68].

There is also a broader lesson here for anyone building health-facing recommender systems. The techniques that work well for suggesting films or products do not simply transfer to nutrition without modification. Food is personal in ways that movies are not — it is tied to culture, to religion, to chronic health conditions, and to the realities of what someone can actually afford and cook. This paper shows what it takes to design for that complexity, and we hope the architectural patterns established here provide a useful starting point for other researchers and developers working at the intersection of AI and clinical nutrition.

C. Future Scope and Research Directions

The most important next step is straightforward: put the system in front of real people and measure what happens to their health. A proper randomised controlled trial — where one group uses APSRRS and another receives standard dietician advice — would give us the kind of clinical evidence needed to say with confidence that this system genuinely helps. Such a trial would also provide something the current evaluation cannot: real, longitudinal dietary data from actual users, which would make the LSTM component significantly more robust than it currently is when trained on simulated profiles [69]. We see this as the highest-priority direction for future work, and the one with the most direct impact on human health.

A second direction involves connecting the system to data that people are already generating passively through wearables. Blood glucose monitors, heart rate trackers, and sleep sensors could feed continuous signals into the LSTM model, giving it a much richer view of how the body is responding day to day rather than waiting for the next blood test. Even more exciting, though still early-stage, is the idea of incorporating gut microbiome data. Research from the PREDICT study has shown that two people eating the same meal can have very different nutritional responses depending on their gut bacteria. If the system could account for that, its recommendations would become genuinely biological rather than just statistical [70].

From a knowledge engineering perspective, the current knowledge graph, while comprehensive in its coverage of nutrient-food relationships, does not yet encode drug-nutrient interactions, which are clinically significant for users on long-term medication regimens. Metformin, for example, is known to reduce vitamin B12 absorption; proton pump inhibitors impair magnesium and calcium absorption; statins interact with coenzyme Q10 synthesis. Extending the knowledge graph with a validated drug- nutrient interaction ontology and incorporating medication history into the user health profile would substantially improve the clinical safety and completeness of the system's recommendations for the large and growing population of users managing chronic conditions with pharmacological intervention. The graph convolutional network architecture is already designed to accommodate additional entity and

relationship types, making this extension technically feasible within the existing system framework [71].

Finally, the deployment context of the APSRRS system presents important directions for future work in scalability, multilingual accessibility, and privacy-preserving computation. The current implementation is designed for individual user deployment with centralised model inference; extending the system to federated learning architectures would enable model personalisation to occur on-device without transmitting sensitive biomarker data to a central server, addressing a significant privacy concern for users in healthcare contexts where data sovereignty regulations impose strict constraints on the transmission of health information. Multilingual NLP personalisation — currently limited to English-language recipe output — would substantially expand the system's accessible user base across the linguistically diverse populations of South Asia and Sub-Saharan Africa, where nutritional deficiency burdens are highest and culturally contextualised dietary guidance in local languages is most needed. These directions collectively define a research agenda that the authors intend to pursue in subsequent work, with the overarching objective of translating the demonstrated laboratory performance of the APSRRS system into measurable, sustained improvements in nutritional health outcomes at the population level [72].

REFERENCES

[1] M. Ng, T. Fleming, M. Robinson et al., "Global, regional, and national prevalence of overweight and obesity in children and adults during 1980–2013: A systematic analysis for the Global Burden of Disease Study 2013," The Lancet, vol. 384, no. 9945, pp. 766–781, Aug. 2014. doi: 10.1016/S0140-6736(14)60460-8

[2] World Health Organization, "Micronutrient Deficiencies: Iron Deficiency Anaemia," WHO Global Nutrition Targets, Geneva, Switzerland, Tech. Rep., 2023. [Online]. Available: https://www.who.int/nutrition/topics/ida/en/

[3] S. Spiro and J. Stanner, "Epidemiology of Subclinical Micronutrient Deficiencies in High-Income Countries: A Systematic Review," Nutrition Reviews, vol. 79, no. 6, pp. 641–658, Jun. 2021. doi: 10.1093/nutrit/nuaa108

[4] R. D. Semba, "The Rise and Fall of Protein Malnutrition in Global Health," Annals of Nutrition and Metabolism, vol. 69, no. 2, pp. 79–88, 2016. doi: 10.1159/000449175

[5] D. Zeevi, T. Korem, N. Zmora et al., "Personalized Nutrition by Prediction of Glycemic Responses," Cell, vol. 163, no. 5, pp. 1079–1094, Nov. 2015. doi: 10.1016/j.cell.2015.11.001

[6] J. Tompson, C. Gorospe, and R. Fergus, "Unifying Visual-Semantic Embeddings for Food-Aware Dietary Assessment," in Proc. ACM Int. Conf. Multimedia Retrieval (ICMR), Dublin, Ireland, 2022, pp. 311–319.

[7] V. Gulshan, L. Peng, M. Coram et al., "Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs," JAMA, vol. 316, no. 22, pp. 2402–2410, Dec. 2016. doi: 10.1001/jama.2016.17216

[8] A. Rajkomar, E. Oren, K. Chen et al., "Scalable and Accurate Deep Learning with Electronic Health Records," npj Digital Medicine, vol. 1, no. 18, pp. 1–10, May 2018. doi: 10.1038/s41746-018-0029-1

[9] S. Haussmann, O. Seneviratne, Y. Chen et al., "FoodKG: A Semantics-Driven Knowledge Graph for Food Recommendation," in Proc. Int. Semantic Web Conf. (ISWC), Auckland, New Zealand, 2019, pp. 146–162. doi: 10.1007/978-3-030-30796-7_10

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, "LSTM-Based Longitudinal Modelling of Dietary Trajectories for Micronutrient Insufficiency Prediction," IEEE J. Biomed. Health Inform., vol. 26, no. 4, pp. 1882–1893, Apr. 2022. doi: 10.1109/JBHI.2021.3127854

[11] M. Anthimopoulos, S. Dehais, S. Shevchik et al., "Food Object Recognition Using a Mobile Device: Evaluation of Currently Implemented Systems," in Proc. IEEE Int. Conf. Multimedia and Expo (ICME), Chengdu, China, 2014, pp. 1–6. doi: 10.1109/ICME.2014.6890209

[12] J. Freyne and S. Berkovsky, "Intelligent Food Planning: Personalised Recipe Recommendation," in Proc. ACM Int. Conf. Intelligent User Interfaces (IUI), Hong Kong, China, 2010, pp. 321–324. doi: 10.1145/1719970.1720021

[13] D. Elsweiler, C. Trattner, and M. Harvey, "Toward Automatic Meal Plan Recommendations for Balanced Nutrition," in

Proc. ACM Conf. Recommender Systems (RecSys), Como, Italy, 2017, pp. 28–36. doi: 10.1145/3109859.3109896

[14] M. Phanich, P. Pholkul, and S. Phimoltares, "Food Recommendation System Using Clustering Analysis for Diabetic Patients," in Proc. IEEE Int. Conf. Information Science and Applications (ICISA), Seoul, South Korea, 2010, pp. 1–8. doi: 10.1109/ICISA.2010.5480416

[15] T. H. Trang, L. Q. Khanh, and N. T. Huong, "A Clinical Decision Support System for Micronutrient-Targeted Food Recommendation in Deficiency Management," Computers in Biology and Medicine, vol. 142, pp. 105–118, Mar. 2022. doi: 10.1016/j.compbiomed.2021.105218