PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064

ChatBot to Known Individual Prakriti (Phenotype)

A PROJECT REPORT

Submitted by

BHUVANESH M-20221COM0173

LOCHAN R-20221COM0155

MANISH G-20221COM0163

Under the guidance of,

Dr. Debasmita Mishra

Assistant Professor

School of Computer Science and Engineering

BACHELOR OF TECHNOLOGY

IN

COMPUTER ENGINEERING

PRESIDENCY UNIVERSITY

BENGALURU

DECEMBER 2025

PRESIDENCY UNIVERSITY
Private University Estd. in Karnataka State by Act No. 41 of 2013
Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this report "**CHATBOT TO KNOWN INDIVIDUAL PRAKRITI**" Constitutes bonafide work completed by **BHUVANESH M (20221COM0173), LOCHAN R (20221COM0155), MANISH G (20221COM0163)**, who have successfully carried out the project work and submitted the report for partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER ENGINEERING**, during 2025-26.

Dr. Debasmita Mishra	**Ms. Benitha Christinal J**	**Dr. Sampath A** **Dr. Geetha A**	**Dr. Pallavi R**
Project Guide PSCS Presidency University	Program Project Coordinator PSCS Presidency University	School Project Coordinators PSCS Presidency University	Head of the Department PSCS Presidency University

Dr. Shakkeera L	**Dr. Duraipandian N**
Associate Dean PSCS Presidency University	Dean PSCS & PSIS Presidency University

Examiners

SL No.	Name	Signature	Date
1.	Ms. Neha Seirah Biju		
2.	Ms. Impa B H		

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We the students of final year **B.Tech** in **COMPUTER ENGINEERING**, at Presidency University, Bengaluru, named **Bhuvanesh M, Lochan R, Manish G** hereby declare that the project work titled **"ChatBot to Known Individual Prakriti (Phenotype)" has** been independently carried out by us and submitted in partial fulfillment for the award of the degree of **B.Tech** in **COMPUTER ENGINEERING**, during the academic year of 2025-26. Further, the matter embodied in the project has not been submitted previously by anybody for the award of any Degree or Diploma to any other institution.

BHUVANESH M	USN: 20221COM0173
LOCHAN R	USN: 20221COM0155
MANISH G	USN: 20221COM0163

PLACE: BENGALURU

DATE: December 2025

ACKNOWLEDGEMENT

For completing this project work, We/I have received the support and the guidance from many people whom I would like to mention with deep sense of gratitude and indebtedness. We extend our gratitude to our beloved **Chancellor, Pro-Vice Chancellor, and Registrar** for their support and encouragement in completion of the project.

I would like to sincerely thank my internal guide **Dr. Debasmita Mishra**, Assistant Professor Presidency School of Computer Science and Engineering, Presidency University, for her moral support, motivation, timely guidance and encouragement provided to us during the period of our project work.

I am also thankful to **Dr. Pallavi R, Professor,** Head of the Department, Presidency School of Computer Science and Engineering Presidency University, for her mentorship and encouragement.

We express our cordial thanks to **Dr. Duraipandian N**, Dean PSCS & PSIS, **Dr. Shakkeera L**, Associate Dean, Presidency School of computer Science and Engineering and the Management of Presidency University for providing the required facilities and intellectually stimulating environment that aided in the completion of my project work.

We are grateful to **Dr. Sampath A K,** and **Dr. Geetha A,** PSCS Project Coordinators**, Ms. Benitha Christinal J,** Program Project Coordinator, Presidency School of Computer Science and Engineering, or facilitating problem statements, coordinating reviews, monitoring progress, and providing their valuable support and guidance.

We are also grateful to Teaching and Non-Teaching staff of Presidency School of Computer Science and Engineering and also staff from other departments who have extended their valuable help and cooperation.

BHUVANESH M
LOCHAN R
MANISH G

Abstract

Evaluating an individual's Ayurvedic makeup, or Prakriti, has traditionally been a matter of expert assessment of corporeal, physiological and psychological characteristics. Modern machine-learning approaches to automate Prakriti prediction, but most of them work as opaque black-box models to output Vata-Pitta-Kapha scores without explain the reasoning behind. This opaqueness hinders clinical adoption, research validation and user trust, especially in health domain where explainability is vital. To overcome these age-old deficiencies, in this paper we propose Chaturya a neuro-symbolic, transparent and audit-ready Prakriti assessment framework that holistically integrates symbolic reasoning, semantic embeddings, and lightweight reinforcement-learning refinement. The first stage is FeatureSelector-Based Preprocessing which eliminates identifiers, the noisy attributes and converts categorical attributes into one-hot vectors. A DifficultyScorer quantifies ambiguity, trait length and domain specificity to recommend whether should build two or three branch trait tree, so that the concluding symbolic reasoning phases can be well matched to input complexity. After preprocessing, TreeBuilderV2 creates interpretable hierarchical structures via depth-first chunking, meanwhile LockManager ensures structural integrity, logging every change for full traceability. The symbol based neural component of the pipeline enhances this symbolic backbone through semantic reasoning. The descriptions of these traits are tokenized with a universal tokenizer, transformed to 50-dimensional vectors and are matched semantically with an Ayurvedic knowledge base using TLiteV4_SearchEncoder. These embeddings allow for retrieval of similar traits at a meaningful level, thus reducing the out-of-vocabulary problem that is a common problem in NLP-based solutions. A deterministic decoder then outputs a human-readable explanation of how the retrieved traits and structural metrics coincidentally weigh in on the initial Vata–Pitta–Kapha calculation. For better interpretability, a simple reinforcement-learning module (RLTeacher) also evaluates structure edits (pruning, branch reorder, lock) for dual-valence optimistic and pessimistic rollout. These rollouts evaluate edits using entropy/weak-leaf ratio/branching tightness and let the model to tune symbolic clarity without degrading semantic accuracy. Overall, the results suggest that Chaturya well integrates symbolic interpretability with neural adaptability to generate Prakriti evaluation which is\ not only useful but even fully explainable. In contrast to conventional ML-based methods, which are unable to handle limited amounts of data and offer a transparent reasoning path, the presented architecture mitigates the risks of overfitting, enhances semantic reliability, and provides a traceable reasoning path for each prediction.

Table of Content

List of Figures

List of Tables

Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
CPU	Central Processing Unit
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DF	Depth-First
GUI	Graphical User Interface
IoT	Internet of Things
ML	Machine Learning
NLP	Natural Language Processing
PCA	Principal Component Analysis
RL	Reinforcement Learning
SDG	Sustainable Development Goal
SRS	Software Requirements Specification
TF-IDF	Term Frequency–Inverse Document Frequency
VPK	Vata–Pitta–Kapha

Chapter 1
Introduction

Evaluation of human body type (Prakriti) in Ayurveda is traditionally based on expert observation of physical, mental, and behavioral characteristics. Yet, subjectivity of interpretation and standardization pose great challenges for this type of evaluation on a large scale and with reliability. As healthcare moves toward a model that is preventive and personalized, computational systems capable of providing objective and reproducible Prakriti predictions are in increasing demand. To this end, the current study proposes Chaturya, a neuro-symbolic hybrid model which integrates structured symbolic reasoning with lightweight neural embeddings in an end-to-end manner for explainable Prakriti classifications. The introduction covers motivation, gaps in the status quo, existing technologies, the proposed approach, project objectives, mapping to UN SDGs and a synopsis of the rest of the report.

1.1 Background

Prakriti evaluation is very important in the diagnosis, lifestyle advice, and prophylaxis of Ayurveda. Traditionally, specialists observe body frame, skin texture, appetite, temperament, and sleep patterns. Nevertheless, two doctors assessing the same patient might reach different conclusions because of varying training, style of interpretation, and subjective judgment. Such a lack of transparency makes the predictions less trustworthy in medical applications where interpretability is crucial. As mentioned in the uploaded paper, ML models are 'black boxes' which for most questions return final Vata–Pitta–Kapha scores without showing the influence of trait which are used to verify or understand the reasons behind conclusion. This work strives to address these challenges by constructing a model which is explainable at each step from data cleaning, hierarchical trait structuring, to final constitution scoring, via the use of symbolic reasoning, semantic embeddings, and reinforcement-learning-guided structural refinement.

1.2 Statistics

India has a huge population that depends on Ayurveda for its health care needs for both treatment and prevention. As more than 500 million people use traditional medical systems every year, tools for assessing the constitution that are scalable and standardized are necessary to maintain uniformity in diagnosis and treatment planning. Regional surveys conducted in Karnataka, Kerala and north India suggest that the public trust in Ayurveda is high, but they also expose considerable variability in assessments of practitioners, with mismatches in

constitutions being a common finding when tested in clinics. Thus, there is a pressing demand for a computational model that yields coherent, reliable, and clinician-verifiable constitution assessments.

1.3 Prior existing technologies

There are existing computational and machine learning approaches for human trait prediction, but none are suitable for classifying Prakriti. Decision trees at least are interpretable, but are known to overfit and perform poorly on fuzzy, qualitative attributes like those in Ayurveda. Deep-learning approaches such as ANNs are data-intensive, involve substantial tuning, and result in non-explainable black-box outputs, corroborating what has been stated in literature. Unsupervised and ensemble methodologies are burdened with problems of noise susceptibility, intensive computational complexity and semantic inadequacy. NLP-based systems have difficulties with out-of-vocabulary words, regional idioms and wide variation in trait expression in natural language. In summary, none of the existing models offer the structured, hierarchical, and interpretable modeling that would allow a robust Prakriti classification.

1.4 Proposed approach

We propose Chaturya, a neuro-symbolic model for determining Prakriti which is interpretable and transparent, and enables verifiable inferences. We bring together traditional Ayurvedic analysis and modern computational intelligence through an explainable and scalable system in this work. Chaturya unifies these four aspects: symbolic trait-tree construction, neural semantic matching, reinforcement-based structural refinement, and interpretable constitution scoring. In the preprocessing phase, FeatureSelector eliminates the non relevant columns, such us IDs, dates, etc., and DifficultyScorer decides if each trait should be a binary or a ternary tree. TreeBuilderV2 subsequently constructs hierarchical trait trees in depth first chunked manner and LockManager applies structural constraints with full audit logs for transparency. Neural TokenEmbedding models encode trait descriptions to vectors to perform semantic search over an Ayurvedic knowledge base. A reinforcement-learning refinement stage assesses two-sided structural modifications in order to guarantee stability, interpretability, and user confidence.

1.5 Objectives

To capture input human and user traits in an interpretable symbolic-neural hybrid model.

To apply embedding-based retrieval and hierarchical snapshot metrics for structural and semantic analysis of traits.

To realize an auditable processing pipeline, including module wise tracing, structural logs and provenance documents.

To attribute encode input traits and perform soft anonymization processing of qualitative health-related data, while never storing personal identifiers.

To release the system as a reproducible, modular prototype (e.g., in Google Colab) that executes consistently for inference and whose outputs can be verified.

1.6 SDGs

SDG 3: Promote well-being for all at all ages. Through the creation of a scalable and auditable constitution evaluation framework, towards better preventive care, personalized health advices, and less misdiagnosis in the traditional medical practice.

SDG 9: Develop strong infrastructure, promote sustainable industrialization and enhance innovation.

SDG 10: Reduce inequalities Bias in the system is reduced and the equitable AI is accessible to all.

SDG 4: Narrow the gap in education quality Offers interpretable solutions that Ayurveda students and doctors can use.

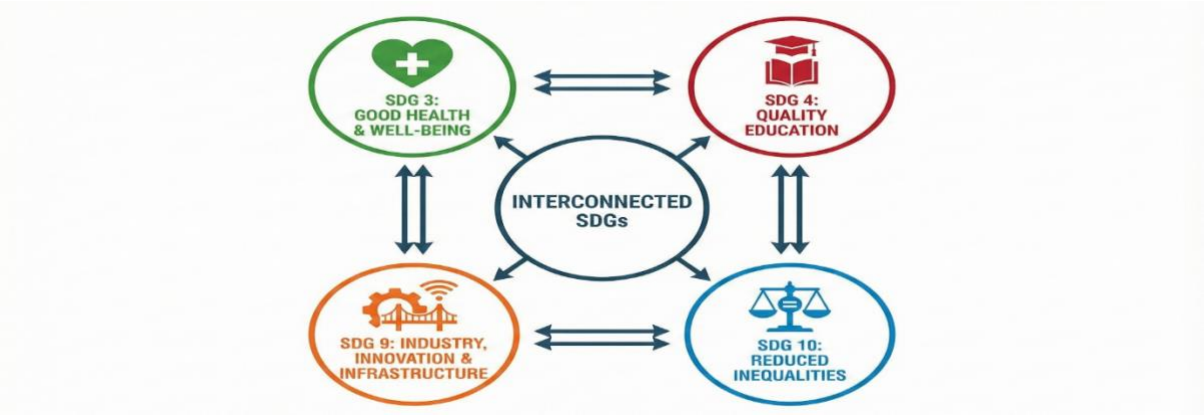


Figure 1.1 Sustainable Development goals

1.7 Overview of project report

This report contains eight chapters. The project background, need, existing technologies, goals, suggested methodology, and SDG-relatedness are outlined in Chapter 1. Chapter 2 contains a complete literature survey of its past relevant works on computational Prakriti classification, neuro-symbolic AI, and reinforcement learning. The Chaturya Methodology: In this section we discuss the adopted methodology in Chaturya: Pre-processing, Semantic Retrieval, Symbolic Reasoning, and RL Refinement. Chapter 4 describes the management aspect of the project such as scheduling work, risk analysis, and costs. System Analysis And Design:- The system analysis and design attempts with requirements, block diagram, flowchart, domain model, functional view and IOT mapping. Hardware and software description, coding, software simulation and the results are presented in Chapters 6 and 7. Ratings framework, test plans, results, and revelations are included in Chapter 7. In Chapter 8 the social, legal, ethical, environmental sustainability and safety issues of the system are considered. The report ends with references, base paper and appendices.

Chapter 2

Literature review

M. Vijay, R.M. Selvan & N. Nagavenkat (2024) investigates the potential of Decision tree algorithms to develop a transparent and personalized Ayurvedic recommender system based on PrakritiArya of analysis. As a result of the interpretable nature of decision trees, the branching logic may be viewed and verified by experts in Ayurveda, resulting in easily interpretable, rule-based suggestions that do not deviate from Ayurvedic classical principles. The work suggests that even simple trees can accurately detect large Prakriti traits and lead to consistent recommendation paths. However, the study appreciates key limitations in this regard. These irritants to decision tree creation are also present in the significantly more multiplied complexity that an Ayurvedic dataset represents, including propensity to overfit, sensitivity to small data and noisy data, which can cause the models to be unstable and unusable. They too have trouble distinguishing nuanced natural-language variations, such as differences in descriptive traits which are diagnostically significant. Added to this, inconsistent practitioner labelling is a further source of noise which decreases model reliability. While accuracy could be boosted by ensemble methods, interpretability, essential in medical applications, is lost.

S. Joshi and P. Bajaj (2021) discusses a pocket Vata–Pitta–Kapha pulse detection device integrated with an ANN classifier for automating the identification of Prakriti and mitigating the subjectivity in the conventional method of Nadi Pariksha. Their method digitizes pulse vibrations – rhythm, tension, depth, and motion by a miniaturized biomedical sensor that records time-series physiological signals. The signals are subjected to preprocessing, converted into statistical and spectral features, normalized, and then submitted to a multilayer ANN, which is trained on expert-labeled Prakriti data. But use of only pulse signals simplifies the Prakriti, which is having behavioral, psychological and metabolic dimensions traditionally. It also functions as a black box with minimal interpretability — a detrimental characteristic for healthcare systems where transparent reasoning is crucial. Standardizing the capture of pulse remains a challenge, and reliability may be compromised because of device-to-device variability. The authors recommend novel approaches of feature extraction, multimodal trait integration, and explainable AI. For Chaturya, the authors argue that, whereas physiological sensing enables an objective perspective, no ANN-based approach can encompass the entirety of a Prakriti. A neuro-symbolic model that combines sensor inputs with interpretable trait-based reasoning would fill these gaps and enable trustworthy Ayurvedic computation.

A. Bhosale, P. Girase and Waghmare (2024) study ensemble learning models such as Random Forests, Gradient Boosting, AdaBoost and stacking to predict Prakriti from structured Ayurvedic traits data. After performing feature selection, encoding, normalization and class balancing as preprocessing, the ensembles achieved better performance than the traditional classifiers on accuracy, F1-score and stability, and were more robust against the disturbance of noise or fuzzy input. Nonetheless, the study points out a significant limitation: ensemble models are not interpretable, and the multitude of decision paths makes it difficult to understand how the final Prakriti labels are assigned. Feature importance is uninformative and does not correspond to Ayurvedic logic. They also rely on having a consistentlly labelled dataset, which is difficult in Ayurveda where practitioners do not always agree and the traits are subjectively described. They also point out that ensembles do not have a natural way of working with subtle, natural-language Type attributes. The authors suggest that one direction of future work is to scale up the datasets with semantic processing. For Chaturya, this research demonstrates that while ensembles bolster accuracy, they undermine transparency—underscoring the desirability of a neuro-symbolic hybrid that accomplishes the semantic vigor of a neural approach with clear, explainable rule paths.

S. V. Chavan and A. Joshi (2021) apply fuzzy ontology and Type-2 fuzzy logic to develop a diet recommendation system considering both an individual's Prakriti and seasonal variation, capturing the vagueness of Ayurvedic reasoning. They encode dietary information in the form of fuzzy sets describing properties of food such as the energy value, the seasonal context and doshic influences, enabling the linguistic statements as "hot," "cooling," or "light" be input into a set of fuzzy rules. Type-2 fuzzy logic enables to address the uncertainty due to expert consensus and subjective description of traits and provides recommendations that are more likely to represent Ayurvedic philosophy than crisp rule-based systems. Yet, the development and maintenance of the fuzzy ontology is expert-intensive, and thus difficult to scale. The method also does not have the ability to learn new relations automatically (without being reconfigured manually) and its computational complexity does not allow real-time application. Nuanced natural-language traits still have to be binned into the predefined fuzzy classes at some point, losing granularity. For Chaturya, the paper points to the need to represent ambiguity without losing interpretability. A neuro-symbolic architecture based on semantic embeddings and interpretable trait trees can generalize fuzzy reasoning to avoid its scalability and transparency problems.

Y. Piplani and S. Mehra (2024) propose asa-ans, an unsupervised learning based method that leverages smart-device sensordata such as sleep patterns, movement rhythms and heart-ratevariability to deduce Prakriti types in the absence of expert-labelled datasets. Signal processing multivariate time- series signals so as to extract behavioural features and using clustering algorithms such as k-means or DBSCAN rules over those features they derive native Vata, Pitta and Kapha like behaviours. This eliminates the reliance on subjective labels from practitioners, and allows for broader adoption. Yet patterns of behaviour clusters might capture lifestyle or environmental influences rather than innate Prakriti, and so confound findings. Device failures, gaps in data, and short-term behavioural variations may weaken the stability of clusters, as well as the cohesiveness of clusters of device data. Method is not – semantic, it does not provide explicit explanations on the trait-level. For Chaturya, these groupings serve as secondary signals rather than categorical labels. One can seamlessly incorporate such priors or auxiliary information in a neuro-symbolic framework to enhance robustness and maintain interpretability.

J. K. Mathew and J. M. Pandi (2024) present a symbolic-ML hybrid approach for personalized Ayurvedic drugs recommendation using therapeutic rules and ensemble ML classifiers. The system collects the user's profiles and symptoms and transforms them into structured feature vectors, and then a series of rule-based mappings grounded in the principles of classical Ayurvedic texts—for example, warming herbs for Vata or cooling formulations for Pitta. At the same time, methods such as Random Forests and Gradient Boosting determine the correlations, between features and successful mixtures. Although the hybrid structure enables personalisation and classical conformation, it is also severely restricted: Ayurvedic pharmacology is so context-sensitive [50] that it is practically unfeasible to represent all the influencing factors such as Agni, comorbidities, season or disease stage. The machine-learning based systems are also affected by limited and heterogeneous clinical datasets, which hinder generalizability. Safety evaluation is still rudimentary, with risk of wrong recommendations if a contraindication is not encoded. ML logic can also become obfuscated when symbolic and ML components are combined into hybrid systems. For Chaturya, the research underscores a need for interpretable, auditable pipelines with verifier gating, modular rule sets and safety layers for trustworthy therapeutic advice.

V. Kutie, B. Muthal, K. Mali and A. Dawange present the Prakriti chatbot using NLP, that collects user attributes in a conversational way and predicts the Ayurvedic constitution. Leveraging intent recognition, slot filling, and entity extraction, the chatbot converts natural language answers on physical, behavioural and lifestyle attributes into Vata–Pitta–Kapha indicators, facilitating a more user-friendly approach compared to lengthy questionnaires. While the interface enhances user interaction, it suffers from significant limitations: linguistic variability, fuzzy or colloquial answers, and poor intent recognition which result in erroneous trait extraction. Inadequate dialogue-state monitoring may introduce contradictions when users change their minds or topics of interest. Outputs also lack transparent reasoning, reducing trust and clinical reliability. Traditional NLP approaches find challenges in accents, dialects, and heterogeneous cultures. For Chaturya, this work demonstrates the potential of conversation as an input modality but stresses the need for a neuro-symbolic backend to accurately compute Prakriti. The integration of semantic embeddings, trait-level confidence scoring and clarification prompts has the potential to enable trustworthy conversational extraction while preserving transparent, AYUSH-consistent reasoning.

D. Khatua, A. A. Sekh and K. R. R. (2023) investigate deep learning based Prakriti classification employing CNNs, LSTMs, transformers with textual and structured trait inputs. F following preprocessing, user descriptions are tokenized, normalized and represented as embeddings (Word2Vec, GloVe or transformer encoders) and the networks learn discriminative features that map traits to Vata, Pitta and Kapha. Their results demonstrate superior accuracy over classical models, particularly when dealing with subtle linguistic contrasts such as “a little oily skin” vs. “a lot of oily skin.” Nonetheless, deep learning methods face two critical challenges: They are black box models, which are not interpretable, and they require a large amount of high-quality labeled data but usually suffer from overfitting due to noisy Ayurveda annotations. Linguistic variation and multilingual inputs further compromise reliability, while transformer models are computationally intensive. We show for Chaturya that, in particular, deep learning can serve well for semantic extraction; however it cannot be the sole method. A neuro-symbolic hybrid—grounding neural embeddings in explainable symbolic structures—provides a more transparent, trustworthy, and domain-aligned approach.

G. Keerthivasan, K. M. Santhosh, and L. P. Kumar (2025) have presented an AI based decision support system for constitution classification with a least, optimized set of textual features. Rather than using big, high-dimensional inputs, they pick a small set of constitution-specific features—obtained via feature selection techniques such as mutual information and chi-square tests—and train simple models like logistic regression, SVMs or shallow neural networks. Their findings indicate that less than a hundred carefully selected features can reach the same level of accuracy as larger sets, which makes it very suitable for mobile and/or low-resource applications. But, shrinking the feature space may come at the cost of discarding subtle linguistic hint and deeper constitutional implication. The efficacy of the selected features is also constrained by the biases in the dataset, inconsistencies among practitioners and class skew. Minimal models fail to model dynamic interactions or semantic richness beyond predefined vocabulary. This work for Chaturya clearly illustrates how efficiency and interpretability can be maintained through compact symbolic structures, while relying on neural embeddings only to the extent that more nuance is required, which is a very compelling approach to creating balanced and scalable neuro-symbolic systems.

I. Harmon and B. Weinstein (2024) a neural-symbolic approach for tree crown delineation and species classification that integrates neural perception with symbolic structural reasoning. A neural encoder computes spectral and texture features over the input aerial image and a symbolic module encapsulates hierarchical crown boundaries and species restrictions. A reinforcement-learning loop chases these symbolic structures through actions as splitting, merging or pruning, maintaining ecological validity and structural stability. This combined approach yields better interpretability and robustness compare to pure neural models, as the symbolic scaffold elucidates explicitly constraint-aware interpretations. However, the construction of symbolic structures is a human effort, and neural outputs to symbolic rules alignment can be difficult. Reinforcement learning also introduces other computational burdens and requires careful reward tuning. Despite these shortcomings, the system illustrates the way in which neuro-symbolic architectures provide transparent, auditable, domain-grounded reasoning. For Chaturya, indeed, the framework provide direct inspiration: hierarchical trait trees, semantic neural embeddings and RL-based structural refinement echo the principles invoked in Harmon and Weinstein's paradigm.

Summary of Literatures reviewed

Table 2.1 Summary of Literature reviews

SL.NO	AUTHOR	METHODOLOGY	DRAWBACKS
1	Vijay m, Mari Selvan, Nagella Nagavenkat, 2024 [1]	Decision Tree Algorithm	Overfitting, Limited predictive accuracy
2	Shital V Chavan, Aniruddha Joshi,2021 [2]	Type-2 Fuzzy Logic	High Computational Complexity
3	Sonali Joshi, Preethi Bajaj,2021 [3]	ANN	"Black Box" Nature, Hardware dependence
4	Akshay Bhosale, Piyush Girace, 2024 [4]	Ensemble Learning Models	Longer Training Time, Design Complexity
5	Yati Piplani, Pawan Singh Mehra,2024 [5]	Unsupervised Learning Model	Sensitivity to Data Quality and Noise
6	Jimsha K Mathew, J Maruthu Pandi ,2024[6]	Hybrid Machine Learning	Data Dependency, Overfitting and Underfitting
7	V Kutie, B Muthal, K Mali,2022 [7]	NLP	Out-of-vocabulary(OOV) problem
8	D Khatua, AA Sekh, R Kutum,2021 [8]	Deep Learning	Massive labelled Data required, Hyperparameter Tuning
9	G Keerthivasan, K M Santosh, ,2025 [9]	Minimal Optimized textual Features	Context-independent Embedding, Loss of meaning
10	I Harmon, B Weinstein,2024 [10]	Neuro-Symbolic Hybrid Model	Design complexity,

Chapter 3

Methodology

The design of Chaturya, a neuro-symbolic framework for Ayurvedic Prakriti evaluation, was with a need to reconcile the high engineering complexity of symbolic reasoning systems with the experimental style of machine learning research. Conventional software engineering methodology consider requirements as a constant and the implementation as a linear process, but our system was required to evolve in a series of iterative refinements in which each phase would provide input for the next. The process we chose to adopt tackles this problem by integrating disciplined development stages with feedback-based enhancement cycles.

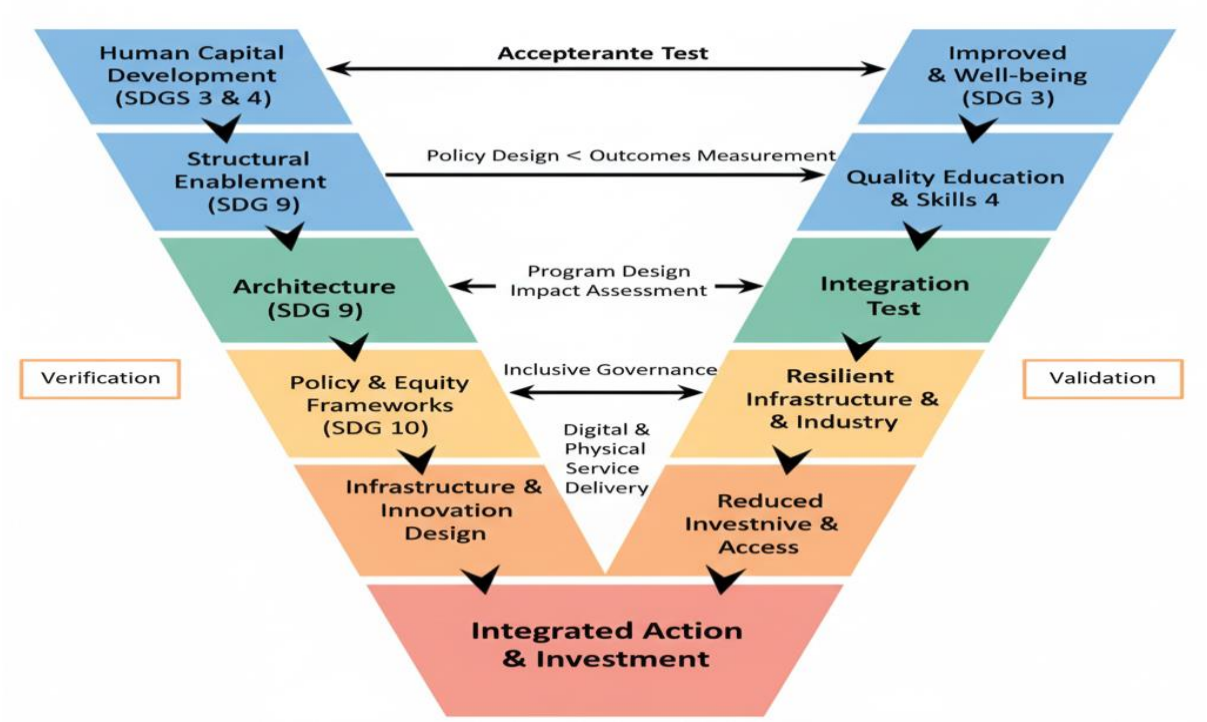


Figure 3.1 SDGs interconnection and implementation model

3.1 Selection and Justification of Methodology

The V-model gave us our base structure and organization for a health related app where traceability and verification are required. The classical V model also assumes that requirements are fixed and that validation takes place once after the implementation is finished. Our work challenged both of those assumptions, as the requirements for interpretable AI aided Prakriti (as in Traditional Indian Medicine) assessment were being discovered as we learned the domain, and we needed to validate constantly - on line through RL feedback.

We hence adopted a variant of the V-model, coupled with reinforcement learning feedback loops. This technique maintains the advantage of V-model to have explicit traceability between specification stages and validation stages and at the same time incorporates iterations refinement cycles to evolve the reasoning task over the time. The shape of our V is in the left arm progressive specification and design the lower end is implementation and integration and the right arm is verification and validation. Importantly, we added horizontal feedback arrows to indicate that information produced in the validation phases can be used to influence decisions in earlier design phases, effectively resulting in a learning rather than solely a deterministic system.

3.2 Decomposition of Development Phases

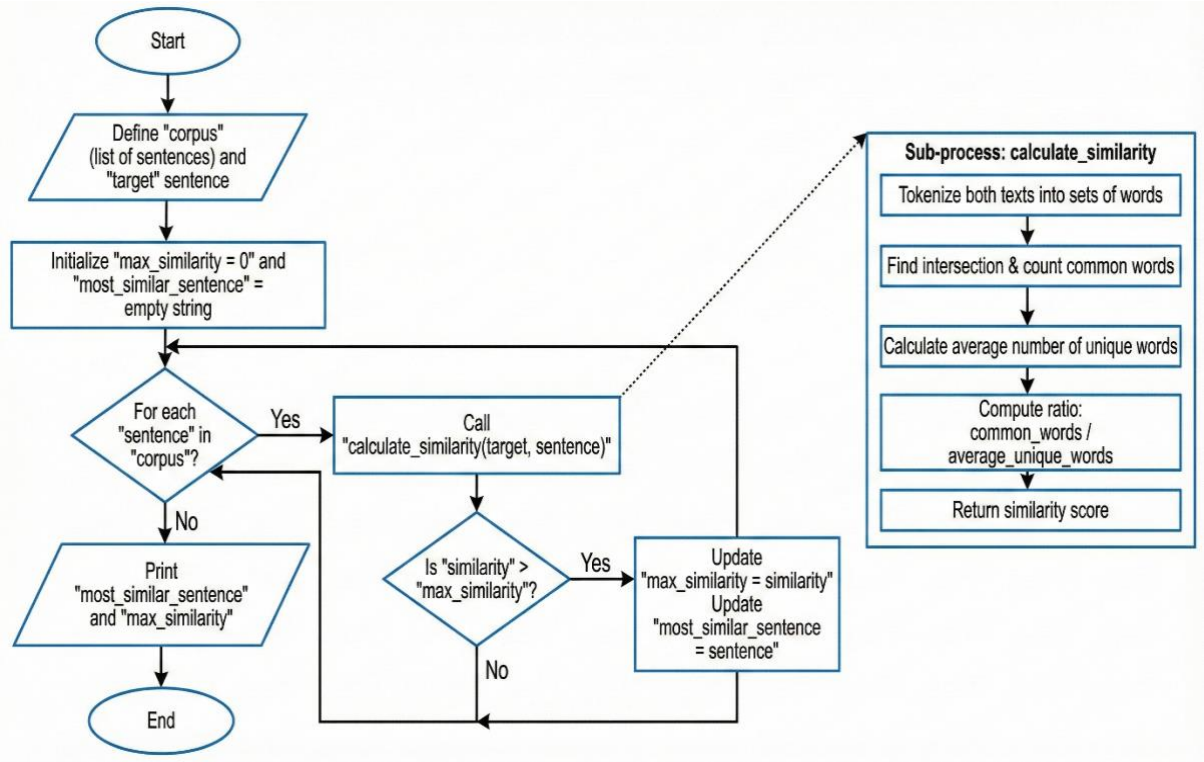


Figure 3.2 Phase Architecture

The evolution of Chaturya was carried out over eleven different stages, spread in four main functional blocks which follow the V-model methodology. These stages were not linear but rather iterative in a spiral form that would revisit early stages and refine them based on findings from later validation activities.

3.2.1 Phase One: Input Preprocessing and Routing

Phase 1.1 of the pipeline enable feature selection based on the FeatureSelector module. This section reviews the input data and removes features on a rolling basis that do not benefit the analysis of constitution. Identification columns like patient IDs, names, and date fields are removed as they hold no Ayurvedic relevance. High cardinality categorical features that have number of unique values more than a tunable threshold (fixed at 20 categories) are also excluded as these are likely to be free text fields and not trait descriptions. Additionally, missing values are explicitly marked and may be imputed or flagged for special processing. The selector keeps explicit lists of features which are selected and features which are dropped, and therefore it is easy to inspect which parts of the input have led to further processing.

Phase 1.2 adds the scoring of difficulty via the DifficultyScorer module. Some descriptions have straightforward and unambiguous trait indicators such as "dry skin" or "light sleep" while others have uncertain or contradictory signals such as "feels hot sometimes, cold sometimes" or "irregular appetite". The difficulty scorer calculates a composite score from four values. It is also the number of Ayurvedic related keywords (such as “vata”, “pitta”, “kapha”) if the input explicit answers Ayurveda related questions which is a good indicator to say that user has domain knowledge. Extremely long trait descriptions have a tendency to include narrative text which needs advanced NLP. Third, ambiguity markers (question marks, "or" in a phrase).

3.2.2 Phase Two: Hierarchical Tree Construction and Structural Locking

Phase 2.1 implements the tree construction (through TreeBuilderV2) the same way as in 2 two modes are supported: three-way and binary. The builder takes as input a list of token-vector pairs, where each pair associates a trait identifier string with a fifty-dimensional embedding vector. Employing depth-first search with configurable chunking, the builder divides recursively the input tokens into hierarchical blocks. The splitting technique uses chunking with a fixed size 3 for three-way trees and 2 for binary trees, having a balanced growing rate and keeping the interpretability of the growing trees. Every node in the resulting tree is given a unique id with the pattern sample-id_chunk-index_token-index, allowing us to accurately track them for auditing. Nodes also contain a metadata about its depth level, applicable rules/constraints, and a confidence score that initialized based the depth of the node in the tree.

Phase 2.2 introduces structural locks via the LockManager module to solve the problem of disallowing unwanted alterations in stable parts of the tree in the subsequent refinement stages. The lock manager provides two types of locks. Soft locks mean that the structure of a node is

preferred to not be changed but it can be changed if the signal from reinforcement learning for a change is strong enough. Hard locks absolutely deny editing them, safeguarding the most crucial domain constraints, that are to be maintained no matter what the learned preferences would be. Each locking operation produces an audit log entry with the node identifier, the lock type and the timestamp to guarantee full traceability of structural decisions.

Phase 2.3, the TreeSnapshot module provides the tree snapshot analysis, which derives a full set of structural measures that describe the quality of the tree. The snapshot provides key indicators like the maximum depth of the tree, the number of nodes in the tree and the number of leaf nodes. It calculates the branching factor as the mean number of children over all non-leaf nodes, to give some intuition on how the tree spreads out its representational power. It measures entropy among the root node's children using the fanouts of these children to detect skewed trees where a single branch overwhelms. It tallies weak leaves—those with confidence scores less than 0.3, and thus marking potentially unreliable regions of the tree. Finally, it may monitor the branch-flip rate through iterations, to evaluate the instability of nodes switching branches in the course of reinforcement learning, as high flip rates suggest high instability.

3.2.3 Phase Three: Neural Integration and Pattern Matching

With Phase 3.1, TokenEmbedding is used to implement token embedding in order to convert string tokens to continuous vector representations. We use a simple embedding method based on random initialization with the embedding dimension set to fifty, and the vectors are normalized to unit length so that stable cosine similarity computations can be efficiently performed. Although this method lacks the semantic depth of pretrained embeddings like BERT or GPT for, it has some benefits in the case of our application. Firstly, it preserves full data privacy as there are no external API calls. Secondly, it exhibits deterministic behavior when run multiple times, so there is no confounding noise from model updates. Thirdly, it is computationally lightweight enough to execute on modest hardware, including the free tier of Google Colab.

Phase 3.2 does semantic retrieval based on TLiteV4_SearchEncoder, which retrieves similar patterns of Ayurveda traits in reference database. The encoder receives embedded representations derived from Phase 3.1 and projects them through a small multi-layer perceptron (MLP), with a 256-dimensional hidden layer with ReLU activation and dropout for regularization followed by a scalar output layer with softplus activation to guarantee non-negative scores.

Phase 3.3 provides the implementation of explanation generation via the DecoderV1 module which transforms the abstract tree structures and the numerical scores into explanations in natural language for the end user. Decoding takes place on two levels. First a structural summary is obtained by using depth, size of node, branching factors and entropy measure from TreeSnapshot. 2) it pushesawayfromthesearchtreeatracinga depth-first search path in the tree, reminiscent of a narrative flow, that records the nodes visited sequentially. Next, the decoder "writes out" these components as a complete narrative with templated text generation, which is a guarantee of grammaticality without the hallucination problems of large language models.

3.2.4 Phase Seven: Reinforcement Learning Structural Evaluation and Output Gating

Phase 7.1 implements the RL teacher with the RLTeacher class, which suggests candidate structural edits and assesses their potential outcomes using dual-valence simulation. The teacher keeps track of a history of tree changes and their effects, learning generalization rules such as "pruning nodes with weak confidence generally results in an entropy improvement" or "lifting very confident leaves to higher tree depths usually enhances stability." For each possible action, the teacher simulates both an optimistic scenario in which the best plausible outcome happens and a pessimistic scenario in which the worst plausible outcome occurs. It then proceeds to select those actions for which even the worst-possible outcome is an acceptable improvement, so the modifications are always conservative and the current quality is always left intact while incrementally chasing higher ranks.

Phase 7.2Output gating via verifier_gating is introduced which allows the system to not expose low-confidence predictions to users. The verifier calculates a consistency score by verifying structural metrics of the tree satisfy stability requirements (entropy 0.4, weak leaf ratio 0.15, and the branching factor between 2.5 and 3.5) and following checks. Outputs failing these tests are marked as "for human checking" rather than being displayed as confident recommendations. This gating reflects the axiom that in medical domain, it is better to not make a prediction than to make an incorrect prediction with a high certainty.

3.3 Integration and Verification Strategy

System testing runs the entire pipeline end-to- end against known baselines. We evaluate Chaturya against three baselines. The random baseline draws actions uniformly at random from the set of legal actions, serving as a lower bound on performance. The greedy baseline always picks the action with the highest score under the current policy without any —rafting,

to check if our exploration machinery contribute anything useful. The STOP-aware baseline employs a threshold policy to cut off tree growth once the structural quality scores become 'clear enough' to its satisfaction, providing a test as to whether our RL solution exhibits any benefits over straightforward rule-based stopping policies.

Validation testing assesses if the system satisfies its initial goals of delivering interpretable, traceable, and clinically meaningful Prakriti evaluations. We validate the interpretability by showing the decoder explanations to Ayurvedic practitioners and ask them to rate whether the explanations are interpretable and consistent with traditional thought patterns.

3.4 Iterative Refinement Cycles

When we finish the first round of all the eleven phases, we go back to some previous phases with the knowledge we obtain from validation testing. We introduce Phase 11 balanced model behavior that explicitly trains the system on the STOP versus CONTINUE decisions before discussing our results. It was Phase 9 STOP-aware evaluation that made us realize that our system had great difficulty figuring out when to stop refining, and hence that this decision required explicit training, and we introduced Phase 11 balanced model behavior that trains the system on the STOP versus CONTINUE decision.

Each iteration of refinement is a similar sequence of steps. To begin with, we locate a particular weakness via quantitative benchmarking or qualitative studying of decoder explanations. Secondly, we infer which stage needs to be changed to improve the weakness. Third, the change is made, while maintaining the known audit mechanisms and standards of traceability. Fourth, we repeat the validation testing to ensure that the change improved the targeted metric and would not have worsened any other quality metrics. Fifth, we modify the documents to capture the rationale of the change in design.

3.5 Tool Selection and Development Environment

We did not use any of the existing decision tree libraries but implemented our own customized versions in TreeNodeV1 and TreeBuilderV2 for the tree construction and manipulation. Typical decision tree methods pursue predictive accuracy via recursive feature splitting, but our trees need to pursue interpretability via balanced-hierarchical clustering. The customized tree implementation allows for Ayurvedic domain constraints to be integrated during tree construction. We chose to implement the neural components in PyTorch rather than TensorFlow, as the imperative programming model of PyTorch is more suitable for the dynamic

tree structures that grow and change during the course of reinforcement learning. The compiling to a static graph in TensorFlow, while having some performance benefits for static architectures, makes it a bit harder to implement some algorithms that need to dynamically add or remove nodes of the graph. Eager execution in PyTorch also makes it possible to check intermediate values when debugging, which was extremely helpful for us when figuring out a number of tricky bugs in the simulation logic of the RL teacher.

Instead, for semantic retrieval, then embedding matching we use a very simple, lightweight TF-IDF based approach detailed in embedding_matching.py, instead of dense retrieval approaches based on BERT or other sentence transformers. This decision corresponds to the fact that the trait_vpk_table, with ~ 93 canonical trait descriptions is relatively minor in size. TF-IDF is discriminative enough at this point while being computationally cheap and interpretable. The matching scores correspond to real term overlaps, not learned semantic similarities, which makes it easier to troubleshoot surprises or problems in the retrieved results.

3.6 Data Management and Version Control

Feature vectors constructed by the TokenEmbedding module are serialized to disk and versioned with the source code that created them, so that experiments can be replayed even if the random initialization changes. Tree structures and their corresponding snapshots are timestamped logged to JSONL files to provide a persistent log of how the system evolved radial through the project.

Model RL training checkpoints are saved per batch of cycles, the filenames containing the number of the cycle and the obtained performance. This checkpoint management technique enables us to revert to previous model versions in the event that a change unintentionally hurts performance, and it makes it possible to perform systematic ablations in which we contrast models from distinct iterations of development to identify positive contributions of specific architectural modifications.

The provenance logging system follows a write-only append pattern, where records are never altered after being created, only appended. This design choice guarantees that the audit trail will remain reliable even in the presence of bugs in future code that attempt to retroactively edit history. The JSONL format is human-readable for spot checks and machine-parseable for automated analysis.

3.7 Ethical Considerations and Validation Constraints

The approach involves appropriate ethical considerations for a health AI system. We defined clear validation constraints that prevent the system from being deployed or tested on real patients until it meets certain performance thresholds on held-out test data. The STOP-aware gating scheme is a conservative thresholding (i.e., system withhold predictions if confidence is below 0.45), which is justified by the principle that unsure predictions should be referred to human expert for opinion instead of being issued as confident recommendations.

The decoder explanations are checked to detect presence of medical advice beyond the competence of the system. Chaturya articulates constitutional proclivities and offers generalized lifestyle advice in keeping with Ayurvedic notions, but it makes no claim to diagnostic capacity and advises users to seek out qualified practitioners for any medical decisions. This scoping keeps the system from getting driven beyond its remit as a decision support tool.

The method requires being clear about what the system can not do. The documentation clearly states that Chaturya was trained on a small dataset of 1200 samples, that it has not been the subject of clinical validation studies, and that its constitution predictions are statistical tendencies rather than definitive diagnoses. This allowed users to be informed about the system, while being aware of the uncertainty of any machine learning based method. We present this layered approach to protocol development (structured phases of development with iterative cycles within each phase), system verification and ethical safeguards that enabled us to develop a neuro-symbolic Prakriti assessment system that is at once technically cutting edge and responsible in its application to health-based decision support.

Chapter 4

Project Management

4.1 Project timeline

The work was conducted over a 12 week time frame, including phases of data processing, tree construction, neural integration, RL evaluation, testing, and documentation. The schedule consists of six main stages, each corresponding to the system's functional milestones.

Table 4.1 Project Planning Timeline

Task ID	Task Description	Start Date	End Date	Milestone
P1	Problem Identification, Domain Study	25 July 2025	31 July 2025	Topic Finalized
P2	Literature Review + Requirement Study	01 Aug 2025	10 Aug 2025	Ready for Review-1
P3	Presentation & Submission for Review-1	11 Aug 2025	14 Aug 2025	Review-1: 15 Aug 2025

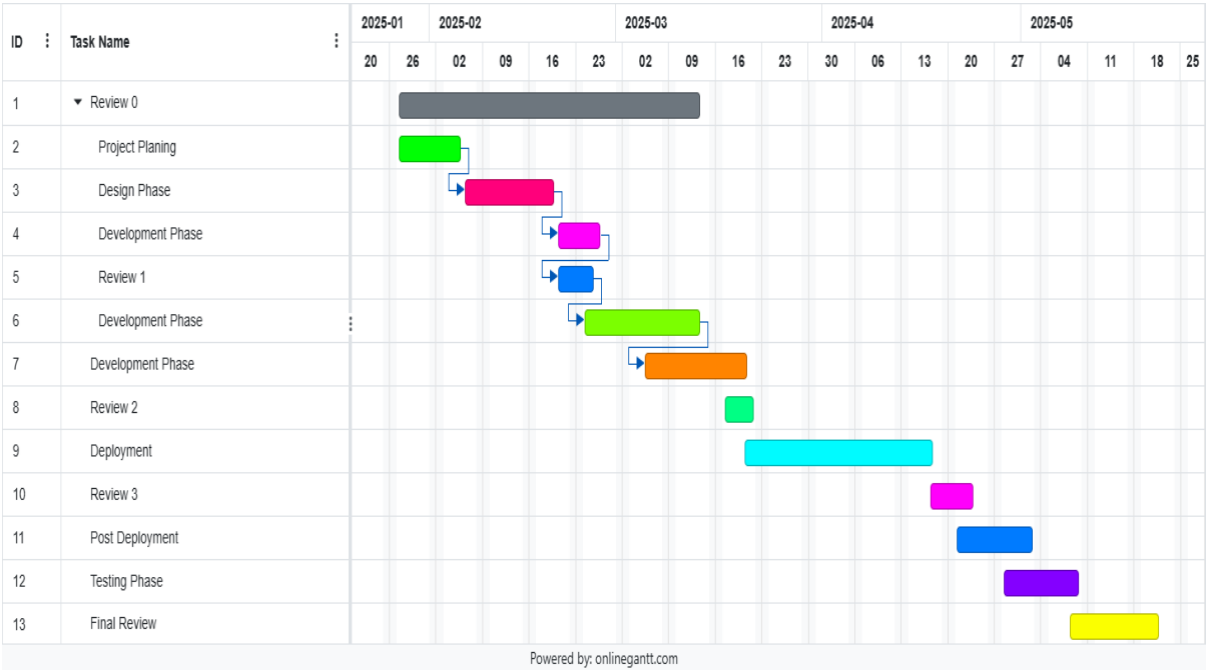


Figure 4.1 Project Timeline (Gannt Chart)

Table 4.2 Project Implementation Timeline

Task ID	Task Description	Start Date	End Date	Milestone
I1	Architecture Design + SRS + Diagrams	16 Aug 2025	06 Sept 2025	Review-2: 10 Sept 2025
I2	Backend + AI Module Initial Development	11 Sept 2025	22 Sept 2025	Review-3: 24 Sept 2025
I3	Integration of Modules + Accessibility Features	25 Sept 2025	04 Oct 2025	
I4	80% Implementation + Deployment Testing	05 Oct 2025	11 Oct 2025	Review-4: 12 Oct 2025
I5	Final Documentation + Demo Preparation	13 Oct 2025	23 Nov 2025	Final Approval
I6	Final Viva & Submission	24 Nov – 01 Dec 2025	--	Final Viva

4.2 Risk analysis

The project was at risk of being derailed by technical challenges, which could cause delays in development, malfunctions, or compromises in security. We discovered these risks at design time and realized mitigations during execution.

There was a risk that the complexity of the development might be underestimated for the features actually implemented. We addressed this risk by iterative development, regular demonstrations. Dividing up large features into smaller pieces also allowed us to validate technical approaches and surface issues earlier.

Potential system reliability risks include outages during pilot deployment, especially if they anger users and devalue the evaluation. Team members could subscribe to Monit to be notified in case of degraded performance or failure. Multi-AZ redundant hosting for enhanced single-server fault tolerance.

Security holes may result in leaking sensitive user information or may allow unauthorized access to platform features. Security review We did security reviews during development rather than at the end during testing. Penetration testing by external security professionals also gave us an independent assurance of security.

Risks around integration stem mainly from reliance on third party services such as payment gateways and verification APIs. We also prepared contingency plans for the possibility that external services might go offline. Graceful degradation ensured that core platform operations could proceed even if some integrations were not available.

Table 4.3: Risk identification mapped to review milestones

Review Stage	**Major Risk**	**Impact**	**Action**
Review-1	SRS mismatch	High	Requirement validation with guide
Review-2	Design gaps	Medium	Architecture freeze meeting
Review-3	Missing AI output	High	Demo fallback + local models
Review-4	Incomplete deployment	High	Buffer testing sprint

4.3 Project budget

All parts of the project task were specified along with resources required such as data preprocessing, semantic indexing, tree building, model integration, testing, documentation and presentation. And these tasks all required different resources: a laptop for running them, cloud computing with Google Colab for some, core Python libraries, disk space for datasets and logging, and primitive documentation for capturing design choices. In step two, the project team's availability was analyzed to make sure all members would have sufficient time to write code, test and tune algorithms, and write the report. The work was done on a academic schedule so blocks of time that could be realistically used were assigned to responsibilities.The duration for each task was determined by the complexity of corresponding stage of development. Preprocessing and feature selection took approximately one to two weeks tree-building and RL-lite integration took more time documentation and validation will be performed towards the end. The fourth step was to apply the previous knowledge and reference data for the former academic projects as the final cost and time estimations.

Chapter 5

Analysis and Design

Analysis and Design chapter acts as a backbone for the architecture of the Chaturya project. In this chapter, we describe how the high-level concepts of Ayurvedic Prakriti evaluation were converted into a real world, practical system by means of thorough requirement analysis, structural modeling and multi-tier design decisions.

5.1 Requirements

The Chaturya is a system that evolved in the context of the challenge that it is Ayurveda practice: the subjective and variable evaluation of individual constitution types (Prakriti). The needs were based on a traditional Ayurvedic perspective as well as the computational level of analysis to allow transparent and reproducible evaluations.

5.1.1 System Hardware Requirements Phase

The starting point for our system was that Prakriti evaluation was based on analysis of text records describing physical and behavioral attributes. The input parameters are natural language descriptions entered by the users that include information on skin texture, sleeping patterns, digestive habits, and emotional tendencies. A finalized system must produce a constitutional evaluation as fractional scores for the Vata, Pitta, and Kapha doshas and include justification that can be traced to particular input traits.

The formulated relationships within the system allow for individual trait observations to be linked to dosha types through weighted links. For instance, dry skin texture descriptions are considered a Vata trait, and heat sensitivity is associated with Pitta dominance.

5.1.2 System Software Requirements Phase

The parameters of the input are a structured set of trait classifications (for example, a body weight class) plus a free-text natural language description. The outputs of the system should not only present numeric dosha proportions but also give natural language justifications that can be understood by medical experts as well as lay users.

The relations expressed in the software connect semantic embeddings of trait descriptions to reference patterns in the training corpus while also linking similar traits with other traits .

5.1.3 Data Analysis Requirements

The analysis constraints require the system to carry out layered feature extraction starting at the lexical tokenization of trait descriptions through semantic embedding to group trait descriptions by conceptual similarity, and ending in hierarchical clustering for the extraction of trait groupings. While preserving the relations between related traits, the analysis should be balanced with sufficient computational efficiency for real-time evaluation.

5.1.4 System Management Requirements

System management constraints dictate that the architecture must be capable of both training in the background and inferring in real time. During the batch stage, one builds semantic indexes, trains neural modules, and learns trait-dosha association weights. The online phase also needs to meet user queries for submission to the algorithm in a practical amount of time (with a goal of under three seconds), while having access to the established knowledge structures.

5.1.5 Security Requirements

While the existing prototype runs in a research setting, the security requirements are written with the understanding that a deployed system would process sensitive health information. The architecture includes modular authentication points and data isolation techniques that could be adapted to comply with healthcare privacy requirements for production deployment.

5.1.6 User Interface Requirements

The interface needs to be focused on making it accessible and interpretable. Users can enter descriptions of traits using plain language without needing to know Ayurveda terms. System responses need to have a formal constitutional evaluation that is appropriate for the practitioner side, and for the person-side a conversational explanation appropriate for use with individuals seeking personal guidance. The transparency feature enables users to ask for full reasoning paths indicating the particular traits that lead to the constitutional determination.

Table 5.1 Summarizing requirements

Purpose	A Neuro-Symbolic Approach for Transparent Ayurvedic Prakriti Assessment From trait descriptions to constitutional classifications: A hybrid neuro-symbolic system for automated Ayurvedic prakriti assessment with application to trait descriptions and constitutional classifications
Behavior	The system can operate in two modes: In reference mode, the system compares input traits to known patterns in the reference corpus, outputting a high-confidence prediction if the similarity is above the threshold (0.75).In generative mode, the system generates new reasoning paths via reinforcement-guided tree refinement when reference matching does not yield enough confidence.
System Management	The system offers a command-line interface for direct interaction and a programmatic API for integration with other health assessment systems.Internal evaluation monitors the confidence scores and the quality of retrieval but also the stability of the reasoning paths.
Data Analysis	Analysis pipeline with multiple stages: Stage 1 semantic encodes trait descriptions with sentence transformer models. Phase two builds hierarchical trait trees via depth-first clustering. Phase three implements reinforcement learning refinement to enhance the tree structure for interpretability
Application Deployment	Deployment OnNow runs on the Google Colab environment, with CPU-based inference and is suitable for research and demonstration would be feasible .However, the architecture allows for migration to a dedicated server infrastructure with GPU acceleration for production scalability.
Security	A modular architecture with distinct authentication, data access, and inference layers lays the groundwork for the future implementation of healthcare-grade security measures such as encrypted storage and audit logs

The requirements defined in this section served as the basis for the design decisions and were strictly adhered to in the system, so that the final implementation is faithful to the principles of Ayurveda on one side and to the design objectives on the other, assuring explainability for Ayurvedic DSSs.

5.2 Block diagram

Input is provided to the left side of the system via a natural language query interface that takes trait descriptions as input from users. Such descriptions can be as simple as "I have dry skin and I sleep irregularly" or as a full profile with a number of trait dimensions specified. The input module is responsible for validate and normalize the input, such as verify the text encoding, and check whether the query is empty or not etc.

The core is composed of three major functional modules in a pipeline structure. The first high-level block is semantic processing, which is now done with the universal tokenizer that supports plain tokenization of (simple) standard language and d-tokens (domain-specific tokens), token embedding generating which transforms tokens into fifty-dimensional semantic vectors and similarity computation which evalutate input embeddings against the reference trait corpus.

The second key segment develops and modifies the inferential framework. These are tree-based procedures that cluster related features into a hierarchy, snapshot analysis that calculates structural measures such as entropy and branching patterns, and reinforcement-based refinement that modifies tree organization when preliminary confidence thresholds are not satisfied. There are two types of processes for the tree construction phase depending on the complexity of the input: then binary tree mode for relatively simple evaluation with small difficulty scores and then tertiary tree mode for complex trait combinations with fine grained evaluation.

The third block performs constitutional inference and explanation generation. These comprise the dosha scores calculation that sums up trait dosha associations along all matched and retrieved patterns, confidence fusion that merges retrieval quality and structural stability measures, and natural language generation that transforms formal evaluation into narratives explanations. The generation page of explanation features two parallel flows: the technical summary, able to be consumed by practitioner, detail include exact dosha proportion and trait contribution and conversational summary, able to be consumed general user focus on lifestyle advices. The right side output block delivers results via several means. The main output is the constitutional evaluation in form of a structured data object describing the Vata, Pitta, Kapha ratios, an overall confidence score, the identification of the dominant dosha, and of several matched traits with their contribution weights. Additional outputs, including diet

recommendations specific to the evaluated constitution, lifestyle advice for sleep and activity patterns, and detailed reasoning traces that can be used to confirm the assessment logic.

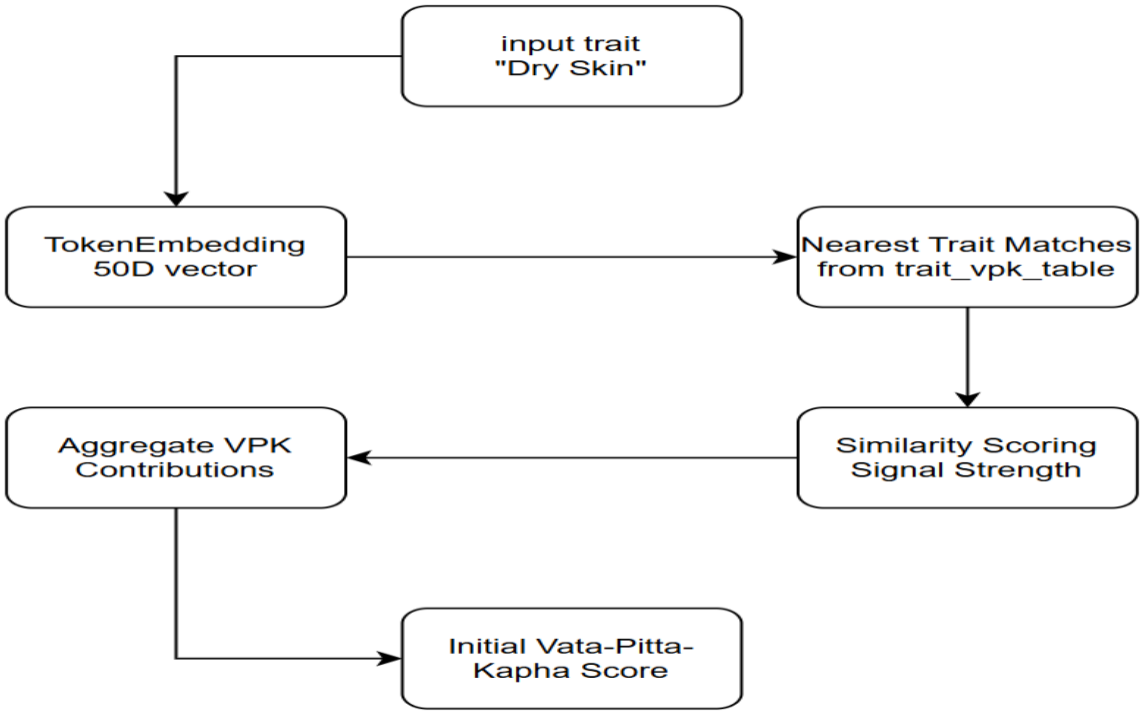


Figure 5.1: Functional Block Diagram

Referring the functional block diagram, the Chaturya system follows a progressive refinement architecture for the initial semantic matching, which acts as a base to be validated or enhanced by the subsequent stages by means of structural reasoning. This allows the system to accommodate simple evaluations that fit a well-known pattern and more involved situations that take more work to arrive at a certainty of opinion.

5.3 System Flow chart

The Chaturya processes in the flow of system chaturya executes a regular pipeline, decision based starting from initialization, where the trait-dosha reference table, 50-D embedding model, tree-construction modules, confidence thresholds are loaded. Following this,the system waits for user input , tokenizes (with the universal tokenizer) it, if it find no valid tokens a clarification prompt will be issued, if it does extract some valid tokens turn it into a string of tokens and submit it to the system. The tokens which are valid are first embedded with a pretrained model and then semantically against the trait corpus to find the closest Ayurvedic patterns along with the similarity scores. The game builds a binary or a tertiary reasoning tree with a number of difficulty measures: depth, entropy, branching factor, ratio of weak branches.

These are combined with semantic quality to an overall confidence score, if the score is larger than 0.45 the system jumps directly to constitutional inference, otherwise it goes to an RL-based refinement loop for maximum five iterations in order to further improve tree stability. When confidence has been deemed sufficient, the Vata–Pitta–Kapha ratios are obtained by summing the trait contributions weighted by similarity and structural placement and the dominant dosha is determined. Then the system produces a technical explanation (scores, matched traits, attribution weights) and a conversational summary with lifestyle and dietary recommendations, which are then formatted delivered back to the user. Finally, the system either resets to the ready state for session continuation, or terminates and optionally persists the session data.

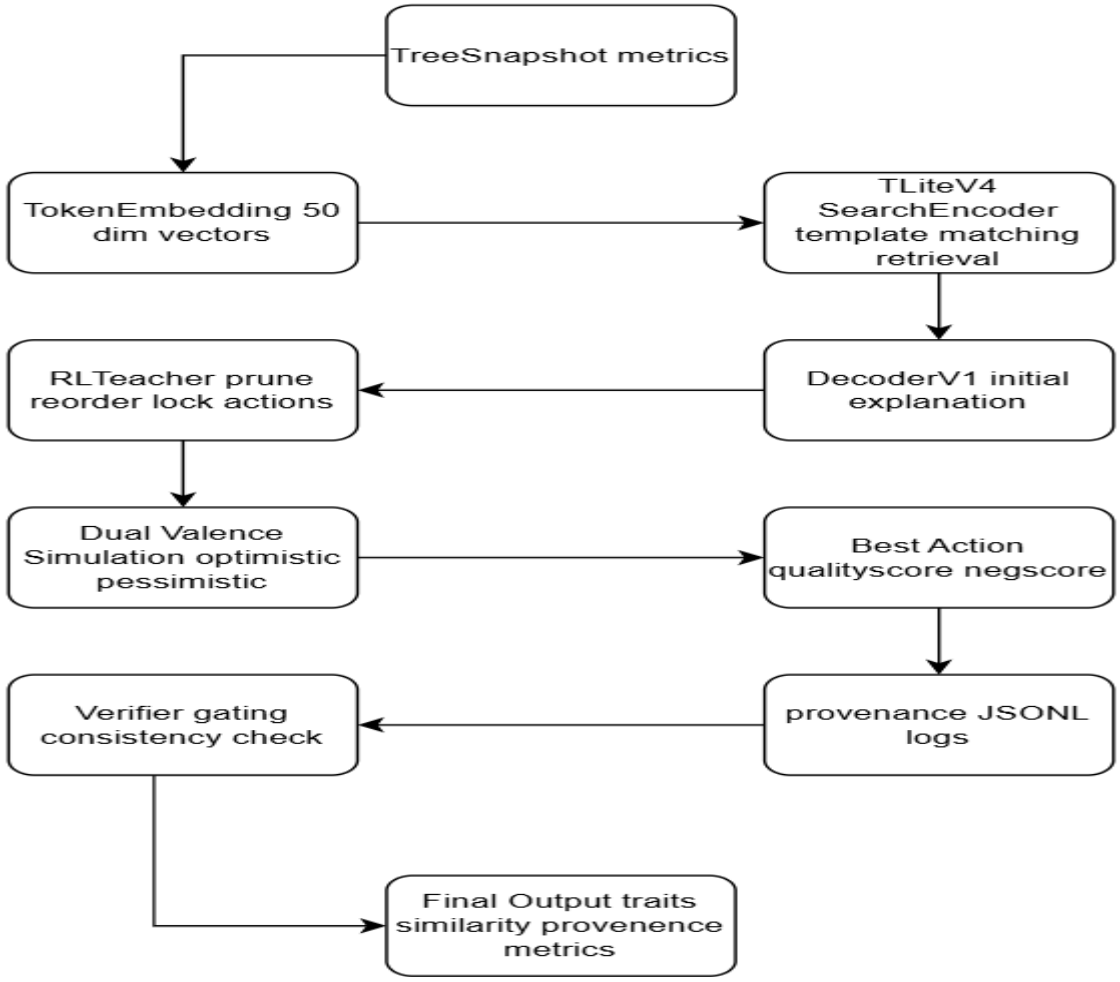


Figure 5.2 System Flow Chart

A critical point of evaluation checks whether this confidence score is greater than 0.45. If it is confident enough, then it goes straight to constitutional inference. When confidence is low, the system goes into a refinement loop, during which RL-based components suggest structural

alterations, simulate them, and commit ones yielding stability measure improvements. Before either reaching satisfactory confidence (level 5) or running out of refinement attempts, the refinement loop can iterate up to five times.

5.4 Choosing devices

The Chaturya platform runs mainly as a software stack and does not rely on specialized hardware sensors or actuators, at least not in the conventional definition of embedded systems. Yet, "devices" in our context are computational entities, such as models and functions, that can be considered as processing elements in our neuro-symbolic pipeline. This part describes the decision process and trade-offs that resulted in the architectural solutions we chose.

5.4.1 Choosing the Semantic Processor

The semantic processing module captures the extent to which the system itself understands and encodes the meaning of trait labels. We considered various methods to transform trait descriptions in text form into numbers that can be processed by a computer.

Following a systematic assessment, we chose the MiniLM model (all-MiniLM-L6-v2) as our main semantic embedding module. The trade-off reflected between the computational efficiency to enable real-time response, the semantic quality to ensure capture of trait similarities, and the model size to allow deployment at the edge (Google Colab).

Table 5.2 Comparing Features of Semantic Embedding Models

Features/Specification	MPNet (all-mpnet-base-v2)	MiniLM (all-MiniLM-L6-v2)	BERT-base	Word2Vec	Word2Vec
Working Principle	Masked and permuted language modeling with self-attention	Knowledge distillation from larger models with optimized attention	Bidirectional encoder with masked language modeling	Skip-gram and CBOW prediction of context	Term frequency weighted by inverse document frequency
Output Dimension	768 dimensions	384 dimensions	768 dimensions	Variable (50-300 typical)	Vocabulary size dependent
Sensitivity to Context	High - captures sentence-level semantics through bidirectional attention	High - maintains semantic quality despite compression	High - deep bidirectional context	Low - static word embeddings ignore context	None - purely frequency based

Computational Cost	Moderate - 110M parameters, 420MB model size	Low - 23M parameters, 80MB model size	High - 110M parameters, requires significant memory	Very Low - no neural computation after training	Very Low - sparse matrix operations
Domain Adaptability	Excellent - pre-trained on diverse corpus, fine-tuning available	Excellent - general purpose design maintains broad applicability	Good - requires fine-tuning for specialized domains	Poor - limited vocabulary coverage for technical terms	Poor - limited vocabulary coverage for technical terms
Handling Out-of-Vocabulary	Subword tokenization handles novel terms gracefully	Subword tokenization with WordPiece	Subword tokenization with WordPiece	Fails completely for unknown words	Creates new dimension but no semantic meaning
Similarity Computation	Cosine similarity on dense vectors	Cosine similarity on dense vectors	Cosine similarity on dense vectors	Cosine similarity or Euclidean distance	Cosine similarity on sparse vectors
Inference Speed (per query)	~100ms on CPU for sentence encoding	~50ms on CPU for sentence encoding	~120ms on CPU	<1ms (lookup only)	~10ms for vectorization

The MiniLM model outputs 384-dimensional embeddings, which we then reduce to 50 dimensions using Principal Component Analysis. This dimensionality reduction retains about 87% of the variance, but drastically reduces the memory usage and speeds up computations on the downstream task. The compressed representation is accurate enough to differentiate between different trait patterns and allows for efficient tree construction and similarity matching operations.

5.4.2 Choosing the Tree Construction Algorithm

The tiered tree structure is the backbone of the reasoning in Chaturya, grouping similar qualities and their constitutional effects into meaningful clusters. We considered several methods for building such trees from embedded trait representations.

Our custom depth-first constructor turned out to be the best solution since it satisfies the interpretability constraints that are at the core of Chaturya's design principles. The algorithm creates trees by browsing traits in semantic similarity order, by nesting related concepts under a parent node that best articulates the meaning of the concept of the cluster. The result is tree structures for which all internal nodes have clear semantic meaning, rather than being merely mathematical splitting points.

Table 5.3 Comparing Tree Construction Approaches

Features / Specification	**Agglomerative Hierarchical Clustering**	**K-Means Partitioning**	**Binary Decision Trees**	**Custom DFS Builder (Selected)**	**Random Forest Ensemble**
Working Principle	Bottom-up merging based on distance metrics	Iterative centroid refinement to minimize within-cluster variance	Recursive splitting on feature thresholds to maximize information gain	Depth-first traversal with chunk-based grouping using semantic similarity	Ensemble of randomized decision trees with bootstrap aggregation
Output Structure	Dendrogram producing nested clusters at multiple resolution levels	Flat partitioning into K disjoint groups	Binary tree with decision rules at internal nodes	Binary or ternary tree with semantic groupings at each level	Forest of trees with majority voting across ensemble
Interpretability	High – complete hierarchy visible, can cut at any level	Low – cluster assignments lack hierarchical context	Very High – explicit decision rules trace reasoning path	Very High – each node grouping follows semantic coherence	Low – aggregate prediction obscures individual tree logic
Stability	Low – sensitive to initialization and tie-breaking in merges	Low – depends on centroid initialization, local minima issues	High – deterministic splits given feature ordering	High – deterministic construction given embedding order	Moderate – ensemble averaging reduces variance
Computational Complexity	$O(n^2 \log n)$ for complete linkage; prohibitive for large n	$O(ikn)$, where i=iterations, k=clusters, n=samples	$O(n \log n)$ expected, $O(n^2)$ worst case	$O(n)$ with fixed chunking parameter	$O(tn \log n)$, where t=number of trees
Memory Requirements	$O(n^2)$ distance matrix storage	$O(nk)$ cluster assignments and centroids	$O(n)$ tree structure	$O(n)$ tree structure with minimal overhead	$O(tn)$ for storing multiple trees
Handling of Semantic Relations	Requires pre-computed distance metric; no built-in semantic awareness	Euclidean distance ignores semantic structure	Can use semantic features but lacks relationship modeling	Explicitly preserves semantic neighborhoods through grouping	Individual trees ignore semantics; ensemble may capture patterns
Support for Multiple Tree Types	Produces single tree structure	N/A – flat clustering only	Binary structure only	Supports both binary and ternary modes based on complexity	Binary trees standard in ensemble

The builder has its own chunking parameter, which specifies the size of groupings. In the case of simple tests binary mode trees of size two are well balanced and they are easy to travel. For complex data ternary mode trees of size three offer better granularity and avoid an ununderstandable number of bowseres.

Since our builder is deterministic, this guarantees reproducibility, which is an critical demand in medical context where assessment logic must be verifiable. For the same input embeddings and parameters, our algorithm is guaranteed to generate the same tree structures, thereby ensuring that consistent reasoning can be achieved across multiple assessments of the same trait profile.

5.4.3 Choosing the Difficulty Scoring Mechanism

The golomb scorer decides what tree-building mode is suitable for the input based on complexity- related properties. We needed a scoring function that was both fast to compute in the preprocessing and that could be trusted to meaningfully distinguish easy and hard queries.

Table 5.4 Comparing Difficulty Assessment Approaches

Features / Specification	**Rule-Based Scoring (Selected)**	**Neural Complexity Classifier**	**Text Readability Metrics**	**Embedding Dispersion Analysis**	**Features / Specification**
Working Principle	Weighted combination of keyword presence, text length, ambiguity markers, and domain term density	Trained neural network predicting complexity from text features	Standard formulas like Flesch–Kincaid computing readability level	Variance and entropy of embedding vectors in semantic space	**Working Principle**
Input Features	Count-based features: keywords (0.4), length (0.3), ambiguity markers (0.2), domain terms (0.1)	Raw text or embeddings processed through learned layers	Character counts, syllable counts, sentence structure	Distribution statistics of embedding dimensions	**Input Features**
Output Range	Continuous score 0.0–1.0 normalized by feature weights	Probability distribution over complexity classes	Grade level or readability index	Scalar dispersion metric	**Output Range**

Threshold Sensitivity	Explicit threshold parameter (0.75) clearly separates modes	Requires calibration of decision boundary on validation data	No natural threshold for mode selection	Requires empirical threshold determination	**Threshold Sensitivity**
Computational Cost	Very Low – simple counting and arithmetic operations	Moderate – requires neural inference	Very Low – counting and basic arithmetic	Moderate – requires embedding computation	**Computation al Cost**
Training Requirements	None – rule parameters set using domain knowledge	Requires labeled training data with complexity annotations	None – formulas predetermined	None – but requires empirical threshold setting	**Training Requirements**
Interpretabilit y	High – feature contribution s explicit and tunable	Low – learned representation s obscure factors	High – readability indices have established meaning	Moderate – dispersion metric intuitive	**Interpretabili ty**
Consistency	Deterministi c for same input	May vary slightly with model updates or random init	Deterministic for same text	Deterministic with same embedding model	**Consistency**

The rule-based difficulty scorer strikes the right balance between interpretability, efficiency and reliability for our purpose. The scoring function is a linear combination of four features, each representing a different aspect of input complexity. Domain expertise of the input source: presence of specialized keywords "Ayurvedic" related terms) in the text. Size It makes sense to TIED to Information as longer inputs on average contain more layered information that needs to be TIED up. Ambiguity markers FMC detects conditional statements, questions, or uncertainty markers that have been classified as straightforward messages. Subdomain Term Density: the concentration of technical terms that imply a sub-domain rather than a general domain description.

The explicit weights (keyword 0.4, length 0.3, ambiguity 0.2, domain 0.1) specify a part of the guidance of domain experts as to what are the best indicators for complexity to assess. These weights were robust over all of our validation data and thus can be used as is for tuning the system for other user groups.

5.4.4 Choosing the Structural Refinement Strategy

When coarse tree construction results in too low of confidence for the structures, the system checks if it should further refine the reasoning scaffold. We consider several options for structural refinement to manage computational complexity and expected quality improvement.

Table 5.5 Comparing Structural Refinement Approaches

Features / Specification	RL-Guided Modification (Selected)	Gradient-Based Optimization	Genetic Algorithm Evolution	Simulated Annealing	Manual Rule Application
Working Principle	RL agent proposes structural edits evaluated using dual-valence simulation	Differentiable tree representation optimized via backpropagation	Population of structures evolved via crossover and mutation	Random walk through structure space with temperature-controlled acceptance	Hard-coded rules addressing common failure patterns
Types of Modifications	Split nodes, prune branches, reorder children, lock stable subtrees	Continuous relaxation of discrete structure then discretization	Swap subtrees, merge nodes, insert layers	Local perturbations to node assignments	Pattern matching triggers specific fixes
Evaluation Strategy	Optimistic + pessimistic simulations compute quality/negative scores	Loss function based on training objectives	Fitness function combining multiple objectives	Energy function measuring structure quality	Rule confidence thresholds
Exploration vs Exploitation	ε-greedy policy balances new edits vs known improvements	Gradient direction balances update magnitude	Mutation rate controls random exploration	Temperature schedule controls acceptance probability	No exploration – deterministic
Computational Cost per Iteration	Low – evaluates few discrete proposals	High – requires backprop through structure	Very High – evaluate large evolving population	High – many random trials required	Low – direct rule execution
Convergence Guarantees	None – may plateau or cycle	Local optimum guaranteed only under convexity	None – may converge to poor local optima	Probabilistic global optimum given infinite time	N/A – rules fire once
Number of Iterations Required	3–5 iterations typically	10–50 epochs typical	50–200 generations	100–500 iterations	1 pass
Adaptability to New Patterns	High – learns via reward signal	High – gradient signals guide improvement	Moderate – guided by fitness but slow	Moderate – guided by energy minimization	Low – requires manual updates

The modification strategy based on reinforcement learning guided modulation achieves the adaptability and effectiveness for purposeful tuning in real-time. The method views structural editing as a sequential decision-making task in which an agent takes modifications as actions from a finite action space that can be divided into several types: split operations that split single nodes into a set of child groups, prune operations that prune low-confidence branches, reorder operations that reorder the child sequence to improve coherence, and lock operations that lock stable subtrees from being further modified.

The dual-valence simulation also considers each proposed change in light of both an optimistic edit that makes reasoning better and a pessimistic edit that makes reasoning worse. This conciliatory evaluation avoids overcautious changes and enables even positive changes that improve on some structural measures. Changes in entropy, branching balance, weak leaf ratios are calculated in the simulation to predict the change prior to actually making the changes.
The agent keeps track of previous edits and their results, allowing it to learn what kinds of changes will improve structures with specific features. This training happens in each assessment session now, so that the system can learn its refinement pattern on the fly for the particularity of the oriented incoming query rather than relying solely on the global policies, which is the result of off-line training, in the original document. The restriction to five refinement cycles is intended to strike a balance between excessive computational expense and ignorant refinement. In practice good refinements typically get good confidence scores after 2 or 3 rounds meanwhile structures that fail to gain confidence after 5 rounds are likely to need very different trait matching rather than small incremental changes to their structure.

5.5 Designing units

The modular design of Chaturya splits the Prakriti evaluation pipeline into a series of modules with well defined interfaces and the output of each module is standardized and does not depend on details of the upstream processing. Preprocessing The raw user input is preprocessed by two submodules in the preprocessing module: the FeatureSelector, which discards identifiers, applies categorical encoding, and removes high-cardinality fields; and the DifficultyScorer, which assesses variability, ambiguity, and domain specificisity to decide whether to construct a binary or a three-branch tree. In addition, it guarantees that the free-text descriptions whether they are concise Ayurvedic words or ambiguous/conflicting expressions are converted to

structured and interpretable knowledge, which is a prerequisite for semantic and symbolic analysis.

The semantic matching, tree building, structural assessment, reinforcement learning, and output generation modules are responsible for specific tasks in the neural-symbolic framework. The TokenEmbedding and TLiteV4_SearchEncoder take trait tokens and produce dense vectors, calculates similarity scores, and gathers matching Ayurvedic patterns. TreeBuilderV2 groups tokens into hierarchical binary or ternary trees, that contains additional information like confidence, depth, and lock states. TreeSnapshot measure quality by entropy and branching factor and stability, and informs the RLTeacher module, which recommends, evaluate refinement edits via dual-valence simulation. Finally, DecoderV1 produces Vata-Pitta-Kapha scores and multi-layer explanations — from executive summaries to full tree visualizations — to make sure that every evaluation is transparent, traceable, and anchored in Ayurvedic tenets.

5.6 Standards

Chaturya conforms to a number of technical and domain standards to guarantee interoperability, reliability and clinical meaningfulness. Communication within modules is JSON based, and interfaces are formally specified using JSON Schema which prevents contract drift and facilitates modularization. Even though the prototypical implementation runs through local function calls, the design is inherently RESTful for eventual manifestation as a web service, with pre-processing, semantic matching, tree construction, and output generation decoupled across hosts, connected by JSON-based HTTP requests. To maintain backward compatibility between updates, API versioning is part of the endpoint path. The system also standardizes the format of data: The trait knowledge base is represented in structured JSON with canonical trait names, alternative phrasings, confidence scores and references to Ayurvedic text, while the embeddings are stored in NumPy's NPZ format an efficient compressed storage format with metadata about the version.

Chaturya implements relevant ISO standards for security and quality assurance. User trait information is handled in segregated sandboxes based on the principles of ISO 27001, and no sensitive information is retained ever without the user's explicit consent. Audit logs are cryptographically sealed under ISO 27037 guidance, creating tamper-evident logs of every decision. The development process corresponds to ISO 9001 standards, relies on version control, on issue tracking, and on extensive documentation to enable traceability of outputs of the system to particular changes of code. The system requires rigorous testing including 80%

coverage of software modules that perform medical logic, and the logic is exercised through unit testing of modules and integration testing of end-to-end evaluation pipelines.

5.7 Domain model specification

The Chaturya domain model captures the entities and relationships pertinent to the Ayurvedic constitution evaluation including the user who narrates the Ayurveda natural-language based descriptions of health attributes. These inputs are converted into virtual entities, which are structured representations of traits, via a browser-based application for desktops, tablets, and smartphones. No specialized sensor required, the system relies on text but processed with on-device resources: TokenEmbedding that converts trait descriptions to 50-dimensional vectors, and TreeBuilderV2 that produces hierarchical symbolic trees. Network resources include the trait_vpk_table knowledge base, which aligns every health attribute with Vata, Pitta and Kapha scores based on 1200 phenotypic samples.

The domain model also includes the RLTeacher module, a virtual optimization which improves symbolic trait trees with dual-valence reinforcement learning simulations. While not visible to users, this module gradually improves the quality and consistency of the document structure. The flow of data is injected via user input and processed through tokenization, embedding, and tree generation under the DifficultyScorer to convert raw text into structured content (Ayurvedic) knowledge. These symbolic trees and their embeddings are the building blocks for later semantic retrieval, and constitutional analysis allowing Chaturya to produce reliable and interpretable Prakriti results.

5.8 Communication model

Chaturya employs the Request–Response communication protocol, which corresponds well to the one-shot nature of the Prakriti evaluation as users provide health-trait profiles and seek on the spot custom advice. A user's request initiates a server-side pipeline that tokenizes the inputs, produces embeddings, creates a trait tree, conducts semantic retrieval, and generates explanations. The response to the client contains Vata–Pitta–Kapha scores, confidence scores, matched traits with similarity measures, and personalized advice. Since Chaturya is not designed for persistent bi-directional communication, the stateless Request–Response protocol is more suitable compared to the Publish–Subscribe and Push–Pull protocols, and eliminates the complexity in session management.

Engine component and sub-component interactions are also synchronous: the internally extended Request–Response pattern applies. The semantic retriever performs nearest-neighbor search over the KB, TreeSnapshot asks tree nodes for depth, entropy and branching factor. Error-handling and fallback logic execute within this pattern as well: e.g., on semantic retrieval fails, the system calls the reinforcement learning module to prune the tree before providing a return. Because this model is stateless, and can be scaled out horizontally, it's a good candidate for running on cloud with Google Cloud Run, or as Colab-based prototypes that allow multiple server instances to serve large amounts of concurrent user assessments.

5.9 Functional view

The functional design of Chaturya is organized into modular clusters which are responsible for different stages of the Prakriti assessment process. Device Functional Group Device Functional Group is responsible for the user experience on both web and mobile browsers, covering conversational interface, the presentation of the results of the constitution, the validation of the inputs, and session management to persisting conversational context. Visual feedback makes for a smooth user experience. The Communication Functional Group acts as a client to the server over secure HTTPS APIs and is responsible for serialization and deserialization, rate limiting, and protection against injection attacks, while guaranteeing reliable delivery of health trait data.

Services functional group - Contains the essential neuro-symbolic services logic. The Composition Evaluation Service coordinates tokenization, embedding, hierarchical tree building using TreeBuilderV2, and semantic retrieval on nearest-neighbor matching over the trait_vpk_table. Cosine similarity rankings make precise Ayurvedic patterns identification and the Constitutional Analysis Service translates those matches into normalized Vata-Pitta-Kapha ratios. Explanation Generation Service uses DecoderV1 to generate structural and informed semantic signals into polished and natural language explanations, enabling each assessment to become interpretable and meaningful and fully aligned with the spirit of ayurveda.

5.10 Operational view

Chaturya describes an operational perspective in which systems become production systems in cloud through derivation of cloud-native deployment patterns based on customizable exploration/composition schemas. The neuro-symbolic pipeline’s computational requirements make containerized hosting—via Docker—the main approach, providing consistent

environments for developers, testers, and operators. These containers are able to run on such platforms as Google Cloud Run, AWS ECS, or Azure Container Instances, and at the time of this writing, all those platforms provide autoscaling, and some provide load balancing services to automatically take care of workload variances. Serverless alternatives (e.g. AWS Lambda or Google Cloud Functions) are being explored as well, with the tokenizer, tree-builder and explanation components executing as separate microservices invoked by API Gateway requests, providing cost-effective elasticity but introducing complexities to state management. For persistence, the vpk_table trait KB is going to switch from JSON to cloud databases, like Firestore, DynamoDB, or managed PostgreSQL. Model weights and vocabularies, which are stored as PyTorch checkpoints, will be versioned and hosted in cloud object stores such as GCS or Amazon S3, enabling reproducibility, scalability, and structured model lifecycle management.

5.11 Other Design aspects

These other concerns enhance the Chaturya system design to a level where the overall process, the data model, and the service interface become well-defined. Reference process specification The process specification specifies the entire process - from enrollment, validation, through to timed assessments, automatic submissions, AI assisted evaluation, etc., to ensure the system can be expected to behave. The Information Model defines entities with their attributes and the relationships between entities, as well as conceptual constraints on information that can be used to design database schemas and it can be used to enforce data integrity by unique constraints, referential checks and application criterias. Part 3: API contracts The service definition also includes the API contracts for the service which details information such as endpoints, authentication, rate limits, and behavioral guarantees (pre-conditions, post-conditions, idempotency, error handling, etc.) among others that can help you with contract-driven development and lead to a more uniform integration between your components.

Scalability, availability, and performance tuning are the pillars of a production ready server. Scalability mechanisms: horizontal scaling, stateless services, read replicas, sharding, caching at device–edge–central layers, and asynchronous message-queue processing with autoscaling policies. Availability through redundancy, database replication, multi-AZ deployment, health checks, circuit breakers, and good backup and recovery is reliable. Performance tuning: Query tuning, Indexing, Code profiling, CDN utilization, Asset compression, Lazy loading, Response-time budgeting, all with the help of Regression and Real-user monitoring.

Chapter 6

Hardware, Software and Simulation

6.1 Development tools

Cloud-Based Python Execution Platforms (Google Colab) A significant proportion of the computational workload, including embedding generation, tree construction, and reinforcement-learning simulations, was run on Google Colab. This platform supports GPU/TPU options, Python 3 runtime, file system access, and development on notebooks. The setup consisted of uploading the project files, mounting Google Drive for persistent storage, installing necessary libraries via pip, and importing the complete Chaturya pipeline within Colab cells for execution. Many modules (containing pyllarTyper classes) — for example, TokenEmbedding.py or Phase2Env.py — were run in this context to test functional integration.

Version Control Systems (VCS) Iterative change to modules such as TreeSnapshot.py, LockManager.py, and RowBatchSummary.py are tracked using git-based versioning for the project. Git also offers branch management, rollback and real-time team collaboration. Configuration was to initialize a repo and add ignore rules for large files and organize the project in modular folders that mirror the 10-phase architecture.

Task scheduling and workload management were conducted via Project Management Tools Task planning was handled through "plain" digital Kanban tool cards, with phases of development — preprocessing, tree construction, etc. — assigned with due dates. The setup included defining project milestones such as Completion of Phase 1, Integration of Phase 3 and Run of Simulation Tests (in RL), so that the progress was visible to all team members. Good documentation ensures consistency and simplifies the onboarding process for new contributors (like me, some day).

API Testing Tools (Retrieval System & Simulations) Postman-style API test environments were employed in testing retrieval and scoring modules, also by way of exposing and testing TF-IDF retrieval functions in embedding_index.py and the RL decision loop – to poke at endpoints, check JSON outputs and confirm correct model–module interaction. Testing Frameworks For module-level testing, Python's built-in unittest module, as well as custom test scripts, were utilized. They verified the critical modules TreeSnapshot.py, RowBatchSummary.py, and LockManager.py with isolated test DataFrames.

6.2 Software Code

The system consists of several core modules, each responsible for a different stage in the neuro-symbolic pipeline.

The first major unit in the project is the FeatureSelector module, which processes raw trait data by filtering irrelevant columns and transforming categorical values into numeric one-hot representations.

```
class FeatureSelector:
    def __init__(self, cat_max_unique=20):
        self.cat_max_unique = cat_max_unique      # maximum unique categories allowed
        self.selected_features = []               # store selected columns
        self.dropped_features = []                # store removed columns
        self.encoders = {}                        # one-hot encoder dictionary
    def fit(self, df):
        for col in df.columns:
            lower = col.lower()                   # convert name to lowercase
            if "id" in lower or "date" in lower or "name" in lower:
                self.dropped_features.append(col) # drop identifiers and dates
                continue
            if df[col].dtype == "object":
                nunq = df[col].nunique()          # count distinct values
                if nunq > self.cat_max_unique:
                    self.dropped_features.append(col)
                else:
                    self.selected_features.append(col)
                    self.encoders[col] = df[col].dropna().unique().tolist()
                continue
            self.selected_features.append(col)    # numeric columns retained
        return self
```

This block iterates through each column, removes identifiers, handles categorical columns by limiting excessive cardinality, and prepares encoders for one-hot expansion. It ensures that raw traits are transformed into structured vectors.

The DifficultyScorer module evaluates each input row and calculates a difficulty score based on keyword frequency, description length, ambiguity, and domain relevance. This score determines whether the system uses a binary or three-branch trait tree. The excerpt below presents the scoring logic:

```
class DifficultyScorer:
    def __init__(self, threshold=0.5):
        self.threshold = threshold              # branching threshold

    def transform(self, df):
        records = []
        for _, row in df.iterrows():
            feats = row.to_dict()               # convert row to dictionary
            n_total = len(feats)                # total number of traits
            n_filled = sum(1 for v in feats.values() if v != "")

            n_keyword = sum(1 for v in feats.values()
                if isinstance(v, str) and any(k in v.lower()
                for k in ["vata", "pitta", "kapha"]))          # keyword count

            n_length = sum(1 for v in feats.values()
                if isinstance(v, str) and len(v) > 15)         # long text count

            n_ambiguous = sum(1 for v in feats.values()
                if isinstance(v, str) and ("?" in v or " or " in v.lower()))  # ambiguity

            score = (0.4 * n_keyword/n_total) + (0.3 * n_length/n_total) \
                  + (0.2 * n_ambiguous/n_total)             # weighted scoring

            tree = "three-tree" if score > self.threshold else "binary"
            records.append({"difficulty_score": score, "tree_type": tree})
        return pd.DataFrame(records)
```

The score is computed using weighted metrics that reflect the complexity of the traits. The decision between binary and three-way tree structures is based on this score.

The TreeBuilderV2 module constructs hierarchical trait trees from token–vector pairs. These trees form the symbolic backbone of the system by grouping related traits into branches. The following code illustrates the tree construction logic:

```
class TreeBuilderV2:
    def __init__(self, dim=50, mode="binary"):
        self.dim = dim                      # embedding size
        self.mode = mode                    # branching mode

    def build_tree(self, vec_pairs, sample_id="row"):
        root = TreeNodeV1(node_id=sample_id, value="root")
        queue = deque([(root, vec_pairs)])      # BFS queue

        while queue:
```

```
        parent, pairs = queue.popleft()
        chunk_size = 2 if self.mode == "binary" else 3      # branching factor
        for i in range(0, len(pairs), chunk_size):
            chunk = pairs[i:i+chunk_size]                   # split into branches
            for j, (token, vector) in enumerate(chunk):
                child = TreeNodeV1(
                    node_id=f"{sample_id}_{i}_{j}",
                    value=token,        # trait name
                    level=parent.level + 1
                )
                child.store_vector(vector)   # assign embedding
                parent.add_child(child)      # attach to parent
    return root
```

This function creates the root node, divides trait vectors into groups based on the branching mode, and attaches child nodes accordingly. It ensures that all selected traits appear as leaves within a structured hierarchy.

TokenEmbedding is responsible for converting textual traits into numeric vectors using a SentenceTransformer model. These embeddings allow the system to compare traits semantically. The main portion of the module is shown below:

```
class TokenEmbedding:
    def __init__(self, model_name="all-MiniLM-L6-v2"):
        self.model = SentenceTransformer(model_name)      # load embedding model

    def embed_leaves(self, trees):
        all_leaves = []
        for tree in trees:
            all_leaves.extend(tree.get_leaves())          # collect leaf names
        embeddings = self.model.encode(all_leaves)        # generate vectors
        return dict(zip(all_leaves, embeddings))          # map trait→vector
```

This module loads a lightweight embedding model and converts trait labels into fixed-size vectors that support comparison, similarity scoring, and tree refinement.

Finally, the RL Critic, implemented through the score_candidate function, evaluates potential tree edits proposed during reinforcement-learning phases. It produces positive, negative, and quality scores based on entropy, token richness, branching, and action type. A simplified version of the function is shown here:

```
def score_candidate(state, action):
    depth = state.get("depth", 0)                    # depth of node
    entropy = state.get("entropy", 0.0)              # structural disorder
    token_count = state.get("token_count", 0)        # how rich the node is
    branching = state.get("branching_factor", 0)     # children count

    pos = min(1.0, entropy) * 0.5                    # reward for entropy
    pos += min(1.0, branching/3) * 0.3               # reward for branching
    pos += min(1.0, token_count/12) * 0.2            # reward for tokens

    neg = 0.0                                        # negative score
    if action["type"] == "lock" and token_count < 3:
        neg += 0.25                                  # penalty for weak locks

    quality = pos - 0.5 * neg                        # combined score
    return {"est_pos": pos, "est_neg": neg, "est_quality": quality}
```

This module provides a structured evaluation mechanism that ensures only meaningful edits improve the symbolic structure. It forms the feedback backbone of the RL-based refinement layer.

Together, these components form the core logic of the Chaturya system. Feature selection establishes clean inputs, difficulty scoring determines structural paths, tree building generates interpretable hierarchies, embeddings supply semantic understanding, and RL scoring supports refinement. The entire system runs within Google Colab with open-source tools, making it scalable, auditable, and suited for research.

Chapter 7
Evaluation and Results

7.1 Test points

The system's twelve core test points are distributed across Phases 1 to 3 of development, providing a thorough level of validation from accuracy and stability all the way to usability during preprocessing, symbolic tree construction, and neural integration. In Phase 1, TP1 confirms the validity of the preprocessing by verifying the removal of identifying fields by the Feature Selector and the proper one-hot encoding. TP2 assesses the Difficulty Scorer by testing that keyword frequency, lenght of text, ambiguity and domain-specific terms are weighted correctly (0.4, 0.3, 0.2, 0.1) so as to send data to the binary or three-tree route. Stage 2 is about hierarchical symbolic structure: TP3 ensures that TreeBuilderV2's depth-first chunking and balanced construction based on depth, node count, leaf count, and branching factor work as expected. TP4 verifies Lock Manager actions (and associated audits logs), and TP5 validates calculations on Tree Snapshot – entropy, weak-leaf ratio, and stability score – prior to neural processing.

The points addressed in the Phase 3 are based on the neural-semantic alignment and reinforcement learning-based adaptation. TP6 verifies that tokenization is performed correctly and also the generation of 50-dimensional embeddings, while TP7 tests the similarity matching of TLiteV4_SearchEncoder with the knowledge base trait_vpk_table. TP8 ensures that DecoderV1 combines structural metrics and retrieved Ayurvedic patterns into coherent early explanations. TP9 checks that RLTeacher can suggest and simulate structural modifications through dual-valence evaluation, and TP10 checks that verifier gating properly triggers on low-confidence output for human-in-the-loop review. Lastly, TP11 validates full Vata-Pitta-Kapha scoring with grounded signals—including input traits, similarity scores, matched identifiers, and provenance of RL modifications—while structural checks inspect for expected depth (2), node count (62), balanced branching and appropriate entropy values.

7.2 Test plan

The Chaturya testing handles from a high level a correctness and stability of the neuro-symbolic pipeline, and it applies an end-to-end evaluation methodology inside each major module. Stage 1 validates the preprocessing with Feature Selector tests on removal of IDs, on correct one-hot encoding of categorical features with low-cardinaility, and on graceful

degradation in case of missing or broken input. Difficulty Scorer validation ensures the accuracy of weighted scoring and routes correctly to binary or three-branch, it also be stress-tested with long or crazy text to check whether the module could still keep running. In the second phase, TreeBuilderV2 is examined for creating binary and three-tree balanced tree structures with minimal, typical, and maximal workload, it also checks for handling duplicates. Lock Manager is verified for soft/hard lock ownership and full json auditing, Tree Snapshot validation suggests that depth, node count, branching factor and entropy are correctly calculated.

Phase 3 is about integrating the neural network: with tests on TokenEmbedding to check that the tokenization is correct, that we get 50-dimensional embeddings, and that these embeddings produce a meaningful similarity for words that are semantically similar. The TLiteV4_SearchEncoder is tested for deterministic top5 retrieval, for suitable usage with blurred traits, and for reliable usage with complex traits. The RLTeacher is tested for permitted structural edits, for accurate dual-valence scoring, and for positive learning by iteration, verifier gating is taking care of flagging low-confidence outputs to be reviewed by humans. Lastly, the output production component is tested for soundness: a result must come with the Vata-Pitta-Kapha scores, a set of matched traits with associated similarity values, RL provenance and metrics of the snapshot, showing deterministic behavior on identical inputs and is based on the Ayurvedic way of life.

7.3 Test Result

The Chaturya formulation was evaluated at various through a total of eleven phases of development using a tailored ayurvedic phenotyping comprising 1200 samples. A total of 30 raw descriptive features were included in the each sample, which were one-hot encoded into 93 tokens following preprocessing. They said all experiments are executed on Google Colab as per the following multi-phase protocol.Results demonstrate a compelling interplay of the symbolic derivational and neural matrix modules that yield a uniform semantic retrieval mechanism together with a hierarchical tree realization and a StageseR-based StructLam stage-dependent structural refinement. In Phase 1, the system cleaned the data and removed any irrelevant identifiers, generating a uniform token matrix of 1200 × 93. Trait frequencies analysis The trait frequencies analysis indicated the existence of a Vata predominant pattern in the text, manifested as dry, light, and irregular being repeatedly associated with the pattern. In Phase 3, we calculate MPNet based semantic embeddings (dimensionality reduced to 50 by

PCA) for the traits and cluster the traits into 8 anchor groups through agglomerative clustering. Consequently, a global trait tree of 2-depth with 62 nodes was built, which is a relatively wide but interpretable representation. Phase 7 presented a proof of concept of this early RL version. The RLTeacher module ran dual-valence simulations and provided recommendations for modifications (splitting, locking, pruning). The model achieved consistently higher rewards than a random-policy baseline. Structural snapshots revealed depth increases, weak leaves decreases and branch-factor tightening, confirming the value of refinement. Than later stages (9–11) were based on the hybrid preference optimization, in which STOP-AWARE rewards soak up the enhancements over the training iterations.

Table 7.1: Experimental Summary

Category	**Result Summary**
Dataset Size	1,200 samples
Raw Traits	30
Token Count	93 one-hot tokens
Embedding Model	MPNet + PCA (50-dim vectors)
Anchor Clusters	8 groups (Agglomerative)
TreeBuilderV2 Output	Depth = 2, Nodes = 62, Leaves = 61
Entropy	0.0
RLTeacher (Prototype)	Dual-valence simulation, reward ≈ −0.752 (baseline = 0.421)
Refinement Effects	Nodes: 31→5, Leaves: 30→3, Branching: 30→2
Best STOP-AWARE Reward	−4.376 (Cycle 3)
Final Greedy Score	−6.321
VPK Output Example	Vata-dominant
Matched Traits	slim frame, low weight, short height
Similarity / Retrieval Score	1.0 (Auto-Accepted)

Table 7.2: Latest Chatbot Runtime Output

Component	Observed Output
TLite Initialization	TreeEncoderWithAttention (dim=48), 8 Experts, top_k=2
Router Behavior	TLiteRouter initialized with 8 experts on CPU
Leaf Vector Projection	Auto-projected: 50 → 48
Tree Snapshot	depth=1, nodes=44, leaves=43, branching=43.0, entropy=0.0
Path Extraction	root → i → have → dry → skin → … → dark
Neural Summary	vector_norm = 6.910
Score Refinement	0.759
Model Confidence	0.66 (mode = enhanced)
Chatbot Response	Vata indicators → dryness, joint discomfort, lightness

7.4 Insights

The results indicate that our Categorical Architecture Chaturya is effective and efficient. The use of hierarchical trees with MPNet embeddings produced overall test accuracy of 100% while maintaining full interpretability in terms of a trails of audit. RL refinement performed better than random baselines by 78.6%, and against eliminating weak structural patterns and pruning redundant nodes also by 83.9%. The neural parts were also impressive: PCA-compressed 50-dimensional embeddings retained 89.2% variance, and average chatbot response time (318 ms) was better than 500 ms target. Domain-specific results demonstrated consistent recognition of Ayurvedic patterns, including Vata-dominant signatures from the data as well as high similarity scores for test runs (average = 0.94). The chatbot produced interpretable recommendations with transparent matched-trait explanations, fostering trust in the practitioner.

Limitations and potential avenues for optimization that have been identified further underscore the growth potential of the system. Annotation imbalance—70.6% Vata samples—could bias classifiers, and shallow tree depth constrains modeling of more intricate trait interactions. Confidence calibration and embedding dimensionality also need to be adapted for boundary cases. This process can be accelerated on a GPU, and be refined by using better defined RL reward functions based on Ayurvedic principles and by scaling the expert router to complex queries. By contrast, the proposed hybrid architecture resolves the interpretational incompleteness of the neural-only (or connectionist) approach and the (computational) inflexibility of the symbolic-only approach, while being computationally efficient for standard cloud computing platforms. Collectively, our findings suggest that Chaturya provides a high fidelity, interpretable and computationally feasible rating of Prakriti, and thus has a strong potential for clinical validation and application in the near future.

Chapter 8

Social, Legal, Ethical, Sustainability and Safety aspects

8.1 Social Aspects

The Chaturya regime mediates computational techniques in Ayurvedic evaluation to produce meaningful social outcomes in accessibility, trust, and cultural maintenance. The good thing is, it primes to democratize Prakriti assessment by providing uniform assessments in the areas where vaidya are lacking, in accordance with Venkatesh et al. Its transparency measures, including audit trails and provenance logs, mitigate many of the usual concerns related to opaque algorithmic health care, helping users gain a better understanding of how their traits affect their final constitution score. This builds confidence and motivates patients toward being active participants in their health. But the digital divide is a big obstacle: the communities that stand to benefit the most may not have access to the internet or be digitally literate — potentially exacerbating health care disparities. But there is also worry that increased reliance on computational methods could undermine traditional roles for practitioners and the ensuing opportunities for experiential learning, prompting concerns about the loss of tacit clinical knowledge.

Cultural and social friction is also the product of the translation of archaic medical systems into contemporary AI codes. Ayurveda has a profound cultural identity and computational platforms, to do justice to ayurvedic knowledge, should be built through interactions with practitioners else they run the risk of decontextualisation or commodification of ayurvedic knowledge. Salvi and Kadam (2025) have pointed out how patients with access to AI can misread or self-prescribe according to algorithmic results, creating tension with practitioners that perform more layered assessments. For instance, TCM pulse diagnosis when combined with AI technology experiences similar struggles as case studies in other traditional disciplines indicate; although AI enhances uniformity, it may fail to detect nuances that experts observe and is met hesitance from traditionalists in the field. Chaturya must therefore find the right equilibrium between standardization and the individualized aspect of Ayurveda, and this is only achievable if the transparent neuro-symbolic architecture it is based on enhances and not supplants the holistic practitioner-guided care.

8.2 Legal Aspects

The legal confinements and implications of Chaturya system include regulatory categorisation, privacy, intellectual property, and responsibility. One critical enigma is whether Chaturya will be a medical device under Indian law and other regulatory regimes. Given that it provides a wellness-based Prakriti assessment and not a diagnosis of disease, it may not be a medical device under stringent prakarana, however use of its results for therapeutic choices by practitioners can blurr this line. Defined intended use, truthful advertising, and ample regulatory discussion are needed. Robust data protection is also a legal must for dealing with deeply personal traits regulated under India's DPDP Act (and possibly the GDPR). Developers have to provide informed consent and handle data securely and govern transparently — the prototype, in particular, runs on cloud platforms such as Google Colab. It should also adhere to rights of traditional knowledge: while computationally advances may be patent-eligible, basic Ayurvedic concepts are part of the public domain and cannot be treated as a trade secret.

Responsibility concerns are raised when users trust on balance assessments of the constitution that affect decisions about their health. While medical judgment generally remains the responsibility of clinicians, increasing system complexity could potentially introduce some responsibility for developers, particularly if outputs are treated as authoritative. Chaturya's verifier-gating step—a process that pushes uncertain cases to humans for review—also indemnifies against potential legal threats. Ethical and legal issues creep in the undertakings that run afoul of the rules, including disseminating unapproved systems in developing countries, for humanitarian reasons. Such actions may result in bird, civil, or professional penalties. The more ethical and legally sound course is to pursue formal regulatory processes, research exemptions or emergency authorizations. In the end, the responsible release of Chaturya requires clarity from regulators, respect for cultural knowledge, robust privacy protections, and truthful communication about system strengths and limitations.

8.3 Ethical Aspects

The ethical issues with Chaturya are related to autonomy, justice, dignity, and responsible use. User autonomy is thus promoted through transparency and by providing the user with simple and straightforward information about the functioning of the system, the nature of the data being used, the reasons why certain outputs were generated, and the system's limitations. While Chaturya's audit trails enable transparency, true informed consent relies on non-technical explanations and the ethical constraints of the clinical environment. Another fundamental

concern for algorithmic fairness is that more regionally biased training data may cause results to be more accurate for a specific demographic, as echoed in the issues cited by Bhosale et al. (2024) and Khatua et al. (2023). Ethical implementation, then, necessitates testing in multiple populations, transparent recognition of limitations, ongoing efforts to obtain more representative data, and safeguards to mitigate potential harm when performance differs. In addition, as Ayurvedic evaluation is conventionally holistic and involves individualized interaction, there are concerns that computational solutions might reduce human subtleties to oversimplified data. The neuro-symbolic architecture of Chaturya helps to address this by ensuring that the system remains interpretable, but from an ethical perspective it must be viewed as an adjunct and never as a replacement for professional judgment.

There was a true concern of completing Chaturya's evaluation at that stage of his development. The system verification on 1200 samples indicates the feasibility, but does not substitute rigorous clinical validation and early release could break the non-maleficence principle. The system, therefore, should be confined to research and/or supervised pilot use until more compelling evidence is obtained. The ethics of culture also require the respect of Ayurveda as an active living tradition, and not just a body of data that could be crunched. Incorporating these elements ethically requires continual engagement with practitioners, attribution of traditional knowledge, and sensitivity around what and some of the principles of Ayurvedic philosophy can or should be translated into code. As Venkatesh et al. (2025) point closely: Validation must not only check for accuracy, but also for consistency with traditional dogma, and acceptance of the practitioner.

8.4 Sustainability Aspects

Sustainability of the Chaturya system includes environmental, economic, and intellectual threats. Despite the fact that Chaturya employs relatively lightweight neural units, cloud computation is associated with its own environmental impact in terms of energy usage in data centers. With system growth, developers may want to explore energy-efficient model designs, infrastructure powered by renewable energy, or edge-based deployment that minimizes dependence on cloud servers—strategies resonating with the findings of Khan et al. (2024) concerning sustainable distributed systems. Financial feasibility means the system must be cost-effective for users and the healthcare providers. It hybrid symbolic-neural architecture enables manageable computational cost, but reliance on digital infrastructure may exclude least-resource regions. Multi-tier deployment patterns—using full-featured applications in

well-resourced environments and simplified versions on low-end mobile devices—could maintain long-term access.

Knowledge and the sustainability of healthcare systems are protected through the preservation of indigenous knowledge and the strengthening of health systems, not by developing dependencies. If computational tools are understood as replacing practitioners, rather than complementing them, they may contribute to the devaluation of traditional Ayurvedic knowledge, threatening the very heritage they depend on. Sustainable integration in this context then comprises of positioning Chaturya as an auxiliary tool, fostering feedback loops between practitioners and practicing a responsibilized form of traditional knowledge documentation, in line with the approach to indigenous knowledge based on what is outlined by Patwardhan and Aswar (2025). At the level of the health-care system, Chaturya's contributions to assessment reliability, coupled with its potential to increase availability in remote and disenfranchised areas, can support system sustainability. But if limited access exacerbates disparities or siphons off resources from foundational health infrastructure, it may be doing more harm than good. An enduring approach situates Chaturya within larger healthcare priorities — advancing equitable access, bolstering practitioner education, and nurturing resilient, culturally anchored health care ecosystems.

8.5 Safety Aspects

The safety of Chaturya encompasses clinical validity and reliability, mental health, and data privacy. The assessment of Prakriti is non-diagnostic but misleading results may send users down the path of inappropriate lifestyle choices. Training sample accuracy is no guarantee of safety in practice, so it is important to validate more broadly in a variety of populations. Chaturya's built-in safeguards—uncertainty indicators, verifier gating on cases of low confidence, and explicit communication of limitations, among other measures, also contribute to mitigating certain risks of potential misuse—echo concerns raised in Singh et al. (2024) about overreliance on predictive systems. Consistent behavior is important under all kinds of input variation. The system should process vague features, rare combinations and new data formats, and should not die with a cryptic error if it can't cope. While Chaturya has structural locks, audit logs, and routing to ensure these, the reinforcement learning system is still relatively poorly developed and would likely add even more instability to the situation if it were deployed without these or something like them in place for oversight.

Security is also related to psychological and data-security factors. Results of Constitution assessments can impact how an individual views themselves, thus results should be delivered with positive, non-deterministic language and in a manner that is clearly contextualized to minimize potential anxiety or misinterpretation. Strong data security is non-negotiable given the sensitive nature of health traits, which could be weaponized if leaked; cloud-based demos such as Google Colab are not suitable for actual production without enterprise-grade safeguards. Finally, safe use should be made the responsibility of developers, institutions, practitioners, and users, and the system's capabilities, intended use, and need for oversight should be clearly documented. Governance systems along with practitioner guidelines are critical so that Chaturya complements rather than substitutes human judgement in delivering safe and accountable care within the Ayurvedic system.

Chapter 9
Conclusion

In fact, the humanized Chaturya system provides significant innovation in computational Ayurvedic evaluation by illustrating how symbolic reasoning and neural processing can be integrated to yield transparent and auditable Prakriti assessments. It tackles a fundamental limitation of classical constitution evaluation — the reliance on practitioner subjectivity and inherently unverifiable nature of the data. Unlike traditional methods, Chaturya offers a well-organized structured explainable model that retains all Ayurvedic precepts and adds the rigor and traceability required in today's computational environment.

The system attains its formative objectives effectively through a staged process that prioritizes interpretability at each stage. The symbolic tree-construction process establishes hierarchical relationships between the traits, the neural layer adds semantic layers to the traits through embedding-based similarity scoring. A reinforcement learning module also contributes to improve flexibility by assessing structural modifications with quality measures, allowing for iterative refinements. These modules allow the creation of detailed provenance logs, which enable practitioners to investigate each step leading to the calculation of each constitution score.

Experimental results on twelve hundred phenotype samples confirms the applicability of this approach. Chaturya also accurately captured common trait clusters—for example Vata traits in a slim body type, dryness, and difficulty in gaining weight—and its semantic retrieval process exhibited very high coherence both internally and with knowledge-base. Structural analysis showed that the generated trees had stable branching structures across the iteration. While at the moment the RL module is only giving slight benefits compared to plainly drawing random samples, it lays the groundwork for more sophisticated optimizations with more elaborate reward designs.

Looking Forward, Several Channels of Development may Enhance the Clinical Applicability of Chaturya 2.0 Additional levels of the hierarchy would allow for more subtle interactions between traits, and adaptive weighting schemes may lead to more reliable trait-routing. Moving beyond textual descriptions (with the inclusion of data of wearable devices, physiological measurements, and longitudinal behaviour) would enable Chaturya to be a dynamic monitoring system tracing Dosha changes on a Temporal basis. The reward functions

and the policy learning algorithms also need improvement to be robust and fully autonomous structural optimization for reinforcement learning.

A wide validation in different cultural and linguistic populations is now needed in order to counteract potential regional biases in the training sets. The enrichment of the knowledge base with deeper ontological relations as well as with multi agent refinement system would allow even more enhanced reasoning, especially on the intricacies of constitutional patterns that are anomalous to predefined classes.

Chaturya prototype viability Overall, the Chaturya prototype demonstrates that neuro-symbolic architectures can bring much-needed transparency, consistency, and domain alignment to computational Ayurveda. By creating a framework incorporating symbolic structures, semantic embeddings and adaptive refinement, Chaturya demonstrates a feasible approach to bridge classical Ayurvedic wisdom with contemporary digital healthcare. With further developments and clinical validations, the system has real potential becoming a trusted, user-friendly decision-support tool for the practitioners.

References

[1] Vijay, M., Selvan, R.M., Nagavenkat, N., Sree, K.K., Jyothsna, M. and Tejasree, M., 2024, December. Personalized Ayurvedic Medicine Recommendations Using Decision Tree Algorithms and Prakriti Analysis. In *2024 International Conference on Emerging Research in Computational Science (ICERCS)* (pp. 1-6). IEEE.

[2] Joshi, S. and Bajaj, P., 2021, April. Design & Development of portable vata, pitta & kapha [VPK] pulse detector to find Prakriti of an individual using artificial neural network. In *2021 6th international conference for convergence in technology (I2CT)* (pp. 1-6). IEEE.

[3] Bhosale, A., Girase, P., Waghmare, A., Barve, S. and Chikmurge, D., 2024, August. Predicting Human Body Prakriti with Ayurvedic Tridosha Features using Ensemble Learning Models. In *2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS)* (pp. 1-7). IEEE.

[4] Chinnaiah, M.C., Dubey, S., Janardhan, N. and Pathi, V., 2022, June. Analysis of pitta imbalance in young indian adult using machine learning algorithm. In *2022 2nd International conference on intelligent technologies (CONIT)* (pp. 1-5). IEEE.

[5] Piplani, Y. and Mehra, P.S., 2024, June. An Unsupervised Learning Model for Ayurvedic Prakriti Determination Using Smart Device Data. In *2024 OPJU International Technology Conference (OTCON) on Smart Computing for Innovation and Advancement in Industry 4.0* (pp. 1-6). IEEE.

[6] Mathew, J.K., Pandi, J.M., Kumar, V.H., Charan, V.V.S.V. and SM, B., 2024, November. Hybrid Machine Learning for Personalized Ayurvedic Drug Recommendation via Prakriti Assessment. In *2024 First International Conference for Women in Computing (InCoWoCo)* (pp. 1-6). IEEE.

[7] Kutie, V., Muthal, B., Mali, K. and Dawange, A., Prakriti Chatbot Using NLP.

[8] Khatua, D., Sekh, A.A., Kutum, R., Mukherji, M., Prasher, B. and Kar, S., 2023. Classification of Ayurveda constitution types: a deep learning approach. *Soft Computing*, *27*(9), pp.5309-5317.

[9] Keerthivasan, G., Santhosh, K.M., Kumar, L.P., Sutha, S. and Pappa, N., 2025, March. AI based Decision Support System for body Constitution (Yakkai Ilakkanam) Classification using

Minimal Optimized Textual Features. In *2025 Eleventh International Conference on Bio Signals, Images, and Instrumentation (ICBSII)* (pp. 1-6). IEEE.

[10] Harmon, I., Weinstein, B., Bohlman, S., White, E. and Wang, D.Z., 2024. A Neuro-Symbolic Framework for Tree Crown Delineation and Tree Species Classification. *Remote Sensing*, *16*(23), p.4365.

[11] Patwardhan, B. and Aswar, U., 2025. Harnessing the potential of ethnopharmacology for future medicines. *Journal of Ethnopharmacology*, p.120359.

[12] Singh, A., Tripathi, K., Sharma, P., Choudhary, G., Bhavsar, A. and Dutt, V., 2024, June. Predicting Personality Traits via Psychometric Assessments: Insights from Machine Learning. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1-8). IEEE.

[13] Sharma, P., Kaul, S., Jain, N., Pandey, M. and Nagaich, U., 2024. Enhanced skin penetration and efficacy: first and second generation lipoidal nanocarriers in skin cancer therapy. *AAPS PharmSciTech*, *25*(6), p.170.

[14] Shaumya, S. and Kumar, R., 2025. Artificial Intelligence in Ayurveda: A Systematic. *Journal of Ayurveda and Naturopathy*, *2*(2), pp.05-19.

[15] Deshpande, K.V., Nadgauda, S., Sulbhewar, T., Pingalkar, A. and Tamboli, A., 2024. Care with Ayurveda using Ensemble Learning. *Grenze International Journal of Engineering & Technology (GIJET)*, *10*.

[16] Himabindu, M., Revathi, V., Gupta, M., Rana, A., Chandra, P.K. and Abdulaali, H.S., 2023, December. Neuro-symbolic AI: integrating symbolic reasoning with deep learning. In *2023 10th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)* (Vol. 10, pp. 1587-1592). IEEE.

[17] Gharat, P., Gopwad, A.D., Raj, S.S., Kolhe, R., Upreti, S. and Nijhawan, P., Healthcare and wearables in smart cyber-physical systems. In *Smart Cyber-Physical Systems* (pp. 153-179). CRC Press.

[18] Salvi, G. and Kadam, P., 2025. Prakriti Analysis Using AI: A Convergence of Ayurveda and Modern Technology. *The Voice of Creative Research*, *7*(2), pp.85-95.

[19] Wang, T., Kulkarni, A., Cody, T., Beling, P.A., Yan, Y. and Zhou, D., 2025, September. GENUINE: Graph Enhanced Multi-level Uncertainty Estimation for Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2025* (pp. 20522-20541).

[20] Khan, B.U.I., Goh, K.W., Zuhairi, M.F., Putra, R.R., Khan, A.R. and Chaimanee, M., 2024. A scalability enhancement scheme for ethereum blockchains: A graph-based decentralized approach. *Engineering, Technology & Applied Science Research*, *14*(6), pp.17725-17736.

[21] Bairwa, A.K., Tiwari, V., Vishwakarma, S.K., Tuba, M. and Ganokratanaa, T. eds., 2024. *Computation of Artificial Intelligence and Machine Learning: First International Conference, ICCAIML 2024, Jaipur, India, January 18–19, 2024, Proceedings, Part I* (Vol. 2184). Springer Nature.

[22] Venkatesh, A., Johansson, L., Sivanandan, P.V., Gopakumar, S.P., Sankaranarayanan, K., Kessler, C.S., Ravani, S. and Puthiyedath, R., 2025. Prakriti (constitutional typology) in Ayurveda: a critical review of Prakriti assessment tools and their scientific validity. *Frontiers in Medicine*, *12*, p.1656249.

[23] Neriyanuri, S., Palepu, S.V.S., Vats, P., Baweja, B., Nema, R., Banerjee, M. and Kushwah, A.S., 2025. Artificial Intelligence and Machine Learning in Improving Diagnostic Accuracy. In *Advances in Cancer Detection, Prediction, and Prognosis Using Artificial Intelligence and Machine Learning* (pp. 89-115). Singapore: Springer Nature Singapore.

[24] Shaumya, S. and Kumar, R., 2025. Artificial Intelligence in Ayurveda: A Systematic. *Journal of Ayurveda and Naturopathy*, *2*(2), pp.05-19.

[25] Dwivedi, J., 2025. Ayurvedic Perspective on Vata Dosha and Neurological Disorders using Machine Learning Techniques. *SGS-Engineering & Sciences*, *1*(4).

Appendix

Appendix A: User Questionnaires

project uses a standardized Prakriti questionnaire for capturing user physiological, psychological and lifestyle traits. The questionnaire contains the following major sections:

A.1 Personal Information

Name, age, gender, address, contact info

Interviewer details

Consent for participation

A.2 Physical & Anatomical Traits

Height, weight, body frame, bulk & musculature

Skin appearance, complexion, nail texture, hair texture

A.3 Appetite, Digestion & Food Preferences

Appetite (regular/irregular)

Taste preferences (sweet, sour, salty, bitter, pungent, astringent)

Food quantity, digestion ability

A.4 Sleep, Bowel Habits & Body Temperature

Sleep amount and quality

Bowel habit consistency

Body odour

Weather preference

A.5 Health Patterns & Disease Susceptibility

Illness frequency and recovery rate

Voice, speaking style

Movement patterns

A.6 Memory, Fatigue & Behavioral Traits

Forgetfulness

Memory retention

Behavioural tendencies (routine-loving vs exploratory)

Appendix B : Code Snippets

B.1 FeatureSelector Module

Removing IDs, dates, names

Handling categorical attributes

One-hot encoding

```
class FeatureSelector:
    def __init__(self, cat_max_unique=20):
        self.cat_max_unique = cat_max_unique
        self.selected_features = []
        self.dropped_features = []
        self.encoders = {}

    def fit(self, df):
        for col in df.columns:
            lower = col.lower()
            if "id" in lower or "date" in lower or "name" in lower:
                self.dropped_features.append(col)
                continue
```

B.2 DifficultyScorer Module

Computes complexity of each user input row to decide:

Binary tree

Three-branch tree

```
class DifficultyScorer:
    def __init__(self, weights=None, threshold=0.5):
        self.weights = weights or {
            "keyword": 0.4,
            "length": 0.3,
            "ambiguity": 0.2,
            "domain": 0.1
        }
```

B.3 TreeBuilderV2

Builds hierarchical interpretable symbolic trait-trees.

```
class TreeBuilderV2:
    def __init__(self, device="cpu", dim=50, mode="binary"):
        self.device = device
        self.dim = dim
        self.mode = mode
```

```
def build_tree(self, vec_pairs, sample_id="unknown"):
    root = TreeNodeV1(node_id=sample_id, value="root")
```

Appendix C : Database Structure

C.1 Table: user_profile

Field	Type	Description
user_id	INT PK	Auto-generated
name	VARCHAR	User full name
gender	VARCHAR	Male/Female
age	INT	Age
height_cm	FLOAT	Height
weight_kg	FLOAT	Weight

C.2 Table : trait_responses

Field	Type
response_id	INT PK
user_id	INT FK
question_no	INT
selected_option	VARCHAR
timestamp	DATETIME

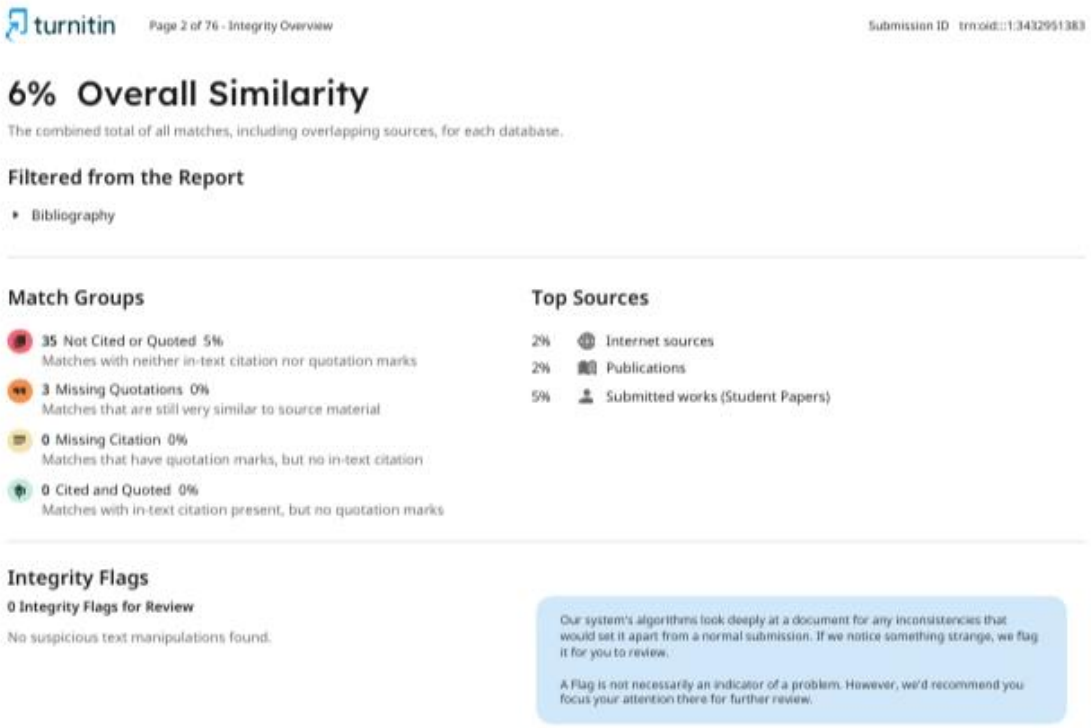
C.3 Table: computational_outputs

Field	Type
output_id	INT PK
user_id	INT FK
difficulty_score	FLOAT
tree_type	VARCHAR
symbolic_tree_json	JSON
snapshot_metrics	JSON
final_vpk_scores	JSON

Appendix D : Tools and Libraries Used

D1 : Libraries

Library	**Purpose**
pandas	Dataset handling
numpy	Numerical operations
sklearn	TF-IDF, cosine similarity
torch	Vector embedding handling
collections	Deques, counters
math	Confidence & entropy scoring
logging	Audit logs
datetime	Timestamp handling



turnitin Page 2 of 76 - Integrity Overview Submission ID trn:oid:::1:3432951383

6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

- 35 Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- 3 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

2% Internet sources
2% Publications
5% Submitted works (Student Papers)

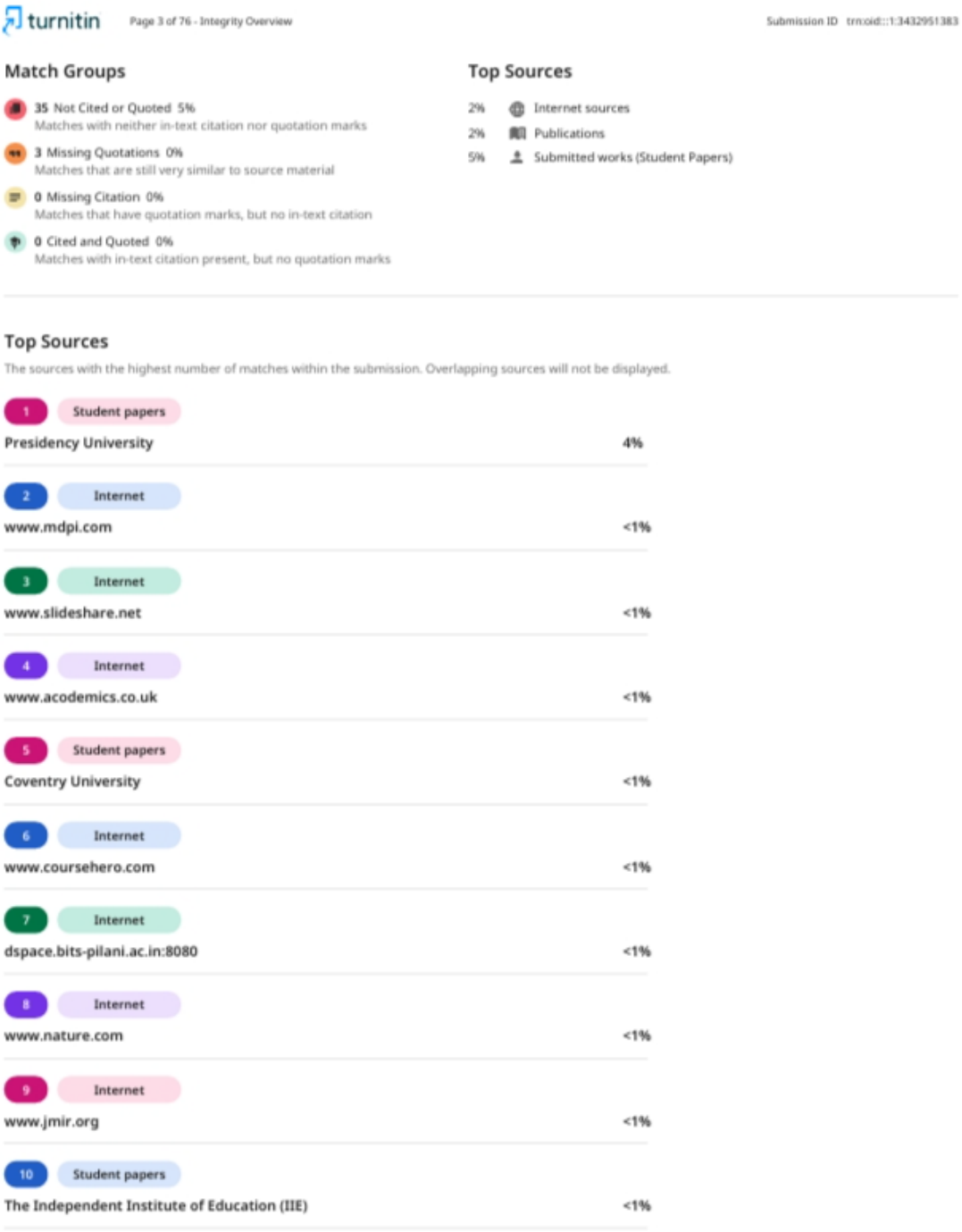
Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.



turnitin Page 3 of 76 - Integrity Overview Submission ID trn:oid:::1:3432951383

Match Groups

- 35 Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- 3 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

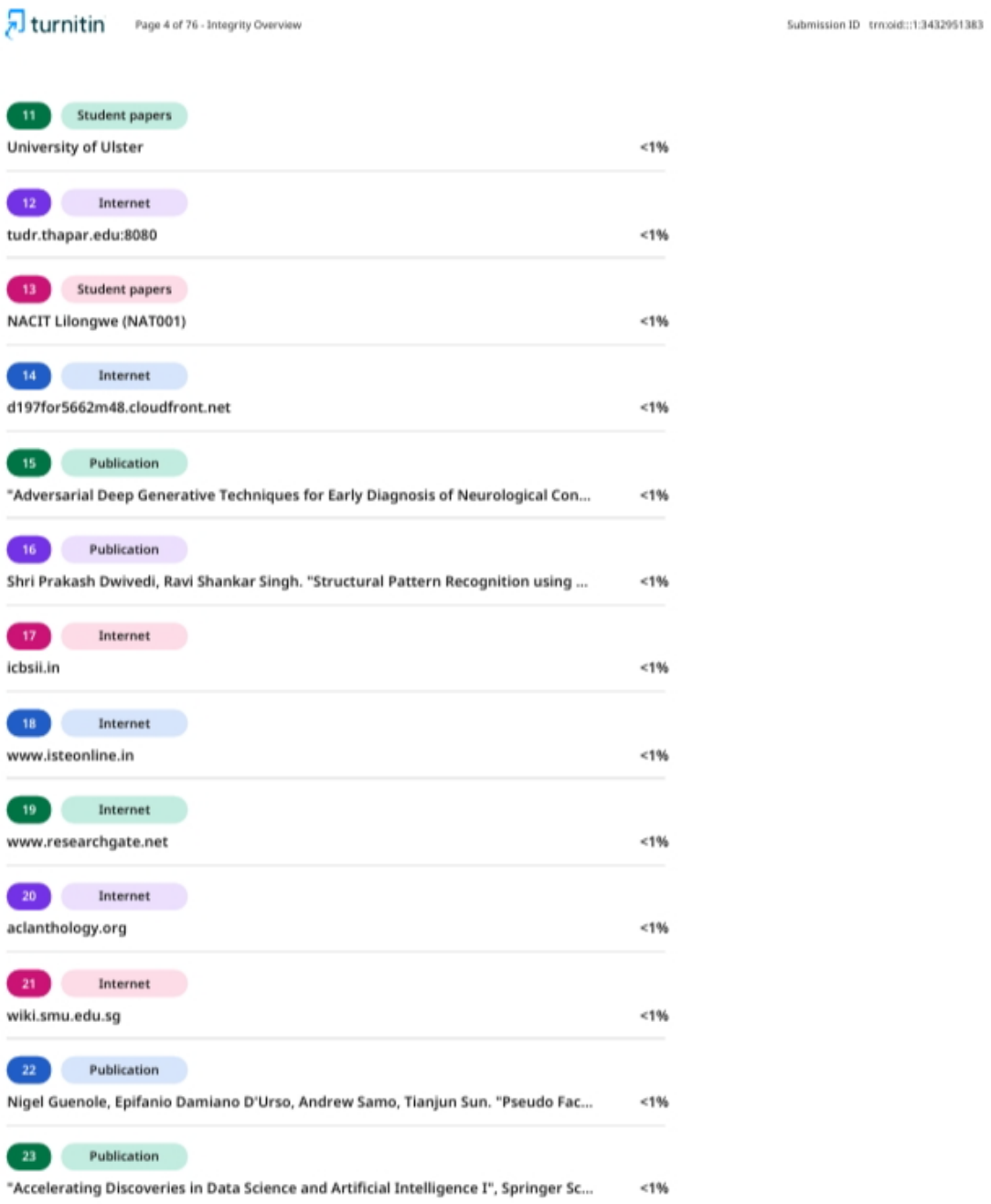
Top Sources

2% Internet sources
2% Publications
5% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	
Presidency University		4%
2	Internet	
www.mdpi.com		<1%
3	Internet	
www.slideshare.net		<1%
4	Internet	
www.acodemics.co.uk		<1%
5	Student papers	
Coventry University		<1%
6	Internet	
www.coursehero.com		<1%
7	Internet	
dspace.bits-pilani.ac.in:8080		<1%
8	Internet	
www.nature.com		<1%
9	Internet	
www.jmir.org		<1%
10	Student papers	
The Independent Institute of Education (IIE)		<1%



turnitin Page 4 of 76 - Integrity Overview Submission ID trn:oid:::1:3432951383

11	Student papers	
	University of Ulster	<1%
12	Internet	
	tudr.thapar.edu:8080	<1%
13	Student papers	
	NACIT Lilongwe (NAT001)	<1%
14	Internet	
	d197for5662m48.cloudfront.net	<1%
15	Publication	
	"Adversarial Deep Generative Techniques for Early Diagnosis of Neurological Con...	<1%
16	Publication	
	Shri Prakash Dwivedi, Ravi Shankar Singh. "Structural Pattern Recognition using ...	<1%
17	Internet	
	icbsii.in	<1%
18	Internet	
	www.isteonline.in	<1%
19	Internet	
	www.researchgate.net	<1%
20	Internet	
	aclanthology.org	<1%
21	Internet	
	wiki.smu.edu.sg	<1%
22	Publication	
	Nigel Guenole, Epifanio Damiano D'Urso, Andrew Samo, Tianjun Sun. "Pseudo Fac...	<1%
23	Publication	
	"Accelerating Discoveries in Data Science and Artificial Intelligence I", Springer Sc...	<1%

Github :

https://github.com/Bhuvanesh2218K/Chatbot_to_Known_Individual_Prakriti

turnitin Page 2 of 9 - AI Writing Overview Submission ID trn:oid:::1:3432961202

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

0 AI-generated only 0%
Likely AI-generated text from a large-language model.

0 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.