AI-Driven Decision Support Systems for Software Architecture: A Framework for Intelligent Design Decision- Making (2025)

Shawaiz Arif, Meer Usman Amjad, Muhammad Faisal

Faculty of Computer Science & IT, Superior University Lahore, Pakistan

Correspondence:

Shawaiz Arif: Shawaizarif1@gmail.com

Article Link: https://journals.brainetwork.org/index.php/jcai/article/view/122

DOI: https://doi.org/10.69591/jcai.3.1.1

Volume 3, Issue 1, 2025

Citation: S. Arif, M. U. Amjad, and M. Faisal, "AI-Driven Decision Support Systems for Software Architecture: A Framework for Intelligent Design Decision-Making," *Journal of Computing and Artificial Intelligence*, vol. 3, no. 1, pp. 1–32, 2025.

Conflict of Interest: Authors declared no Conflict of Interest

Acknowledgment: No administrative and technical support was taken for this research

Article History

Submitted: Mar 01, 2025
Last Revised: Apr 20, 2025
Accepted: May 12, 2025

Funding
No

An official Publication of
Beyond Research Advancement & Innovation Network, Islamabad, Pakistan

AI-Driven Decision Support Systems for Software Architecture: A Framework for Intelligent Design Decision-Making (2025)

Shawaiz Arif*[1], Meer Usman Amjad [1], Muhammad Faisal[1]
[1] Faculty of Computer Science & IT, Superior University, Lahore, Pakistan

Abstract

Software architecture decision-making is a critical phase in the software development lifecycle, often constrained by time, complexity, and uncertainty. As software systems grow in scale and dynamism, architects require intelligent tools that can assist in evaluating architectural alternatives, predicting quality trade-offs, and automating design suggestions. This paper explores the integration of Artificial Intelligence (AI) into the software architecture decision-making process. We review existing AI-driven architectural tools, classify relevant AI techniques (including expert systems, machine learning, and large language models), and propose a conceptual framework for an AI-based Architecture Decision Support System (AIDSS). The proposed system aims to enhance decision quality by learning from historical design data, recommending optimal patterns, and ensuring traceability. We demonstrate the potential of the framework through example use cases and discuss its applicability in real-world architectural practices. A visual model of the system architecture is presented. This paper provides theoretical insights and practical directions for implementing AI-assisted decision-making in software architecture.

Keywords*:* Software Architecture, Artificial Intelligence, Decision Support System, Architecture Decision Records (ADR), Large Language Models, Machine Learning, Design Automation, AI in Software Engineering.

Introduction

In modern software engineering, software architecture significantly influences system success, scalability, maintainability, and performance. As software systems grow more complex—particularly with distributed systems, cloud-native architectures, and AI-integrated applications—the need for timely and rigorous architectural decision-making becomes critical. Architects must consider multiple factors such as performance, security, cost, flexibility, and compliance, often under tight deadlines and with limited information [1].

* Corresponding Author: Shawaizarif1@gmail.com

Traditionally, architectural decisions have relied on expert intuition, experience, and manual evaluation of trade-offs. While these approaches benefit from human contextual understanding, they are inherently subjective, inconsistent, and difficult to scale in large, fast-moving environments. Furthermore, decisions are rarely recorded formally or reused systematically, resulting in repeated mistakes, undocumented trade-offs, and loss of architectural knowledge over time [2].

Artificial Intelligence (AI) has emerged as a transformative tool in software engineering, spanning requirements analysis, testing, and maintenance. In architecture, AI can shift decision-making from a heuristic, human-driven activity to a data-driven, knowledge-enabled process. Techniques such as machine learning, natural language processing, and large language models (LLMs) can support pattern recognition, prediction, trade-off analysis, and architectural style suggestion [3][4].

This study introduces a conceptual Architecture Decision Support System (AIDSS) that leverages AI to assist architects in the design stage. The system is designed to:

1. Learn from historical Architecture Decision Records (ADRs) and design histories
2. Suggest architectural styles and patterns based on system context
3. Analyze and predict quality attribute trade-offs
4. Provide explainable recommendations for transparency
5. Ensure traceability of decisions throughout the development cycle

This framework avoids repeating benefits and limitations in other sections and focuses on the novelty of AI integration in architectural decision support.

Literature Review

The increasing sophistication of software systems and the need for high-quality architectural decisions have led to the incorporation of Artificial Intelligence (AI) into software architecture practice. The section overviews the history of AI application in software architecture, the types of AI techniques utilized, and the upcoming challenges recognized in recent literature.

Evolution of AI in Software Architecture

The last few years have seen a considerable growth in studies examining the ways in which AI can support software architects in design, decision-making, and documentation activities. Bucaioni et al. performed an extensive systematic literature review of AI application in software architecture between 2019 and 2024, determining 14 different application domains, such as design automation, architecture recovery, pattern suggestion, and trade-off analysis [3]. Their research also highlighted six main

challenges: explainability, adaptability, data availability, lifecycle integration, traceability, and technical debt awareness.

Previous contributions by Jansen and Bosch focused on how to represent architecture as a sequence of design choices instead of mere static models [2]. This transition towards decision-based architecture provided the grounding for AI systems supporting dynamic, context-based decision-making instead of inflexible, predetermined templates.

AI Techniques in Architectural Decision Support

AI techniques used in software architecture can be generally classified into the following:

1. Rule-Based Expert Systems: These systems store architectural knowledge as IF–THEN rules. While restricted in learning and scalability, they are beneficial in thin areas and legacy choice situations [5].
2. Machine Learning (ML): ML models are capable of forecasting architecture performance measures or suggesting appropriate patterns from past data. Supervised learning has been employed to convert architectural configurations into system quality properties like reliability or latency [6].
3. Natural Language Processing (NLP) and Large Language Models (LLMs): LLMs like GPT-4 and Codex have shown potential in understanding and generating architecture decision records (ADRs), documentation, and pattern suggestions. A recent study by Schmid et al. reported successful use of LLMs for classifying design decisions and generating architecture views from textual requirements [7].
4. Case-Based Reasoning (CBR): CBR systems retrieve similar past architecture cases to guide current decisions. This approach supports traceability and justifiability of design choices [8].

Tools and Frameworks

Several tools have been proposed or implemented to operationalize AI-based architectural support:

1. SmartArch: A tool that uses ML to support architectural design by analyzing system logs and recommending changes [9].
2. DecisionArchitect: Captures architectural design decisions and relates them to quality attributes, helping trace rationale [10].
3. GPT-based Plugins: Experimental integrations with tools like PlantUML, Zotero, and Obsidian have enabled auto-generation of architecture diagrams and annotations from prompts.

4. However, many tools suffer from lack of explainability, limited domain generalization, and poor integration with software engineering pipelines [3], [4].

Research Gaps and Future Directions

Despite the growing body of research, several open challenges persist:

1. Explainability: ML/LLM-generated suggestions are often "black-box" and lack justifications.
2. Lifecycle Integration: Most tools focus on early design but ignore downstream changes and feedback.
3. Dataset Scarcity: High-quality, labeled architectural decision data is limited.
4. Evaluation Frameworks: Benchmarks for comparing AI-driven decision tools are lacking [4].

Proposed Framework

AI-Based Architecture Decision Support System (AIDSS) To assist software architects in making accurate, consistent, and explainable decisions, we propose a modular, feedback-enabled AI-Based Architecture Decision Support System (AIDSS). The system combines rule-based logic, machine learning, and large language models to support architectural planning and evaluation across the software development lifecycle.

System Architecture Overview

The AIDSS architecture (see Fig. 1) consists of five core layers:

Input Layer

1. This layer ingests inputs from the software project, including:
2. Functional and non-functional requirements
3. Design constraints (e.g., performance limits, regulatory compliance)
4. Business goals and priorities

Data Processing Layer

This layer processes and structures diverse data sources to create a knowledge base:

1. Historical Design Data: Past architectural blueprints and documentation.
2. Architecture Decision Records (ADRs): Structured logs of past decisions and their outcomes.
3. System Logs & Runtime Data: Useful for performance feedback and adaptation.

AI Engine

The core of the framework, the AI Engine integrates three AI approaches:

1. Rule-Based System: Encodes expert knowledge and domain-specific constraints.

2. Machine Learning Module: Predicts quality attribute outcomes (e.g., latency, availability) based on architectural configurations.
3. Large Language Model (LLM) Unit: Interprets natural language requirements and suggests architectural styles or refactoring actions.

Decision Layer

1. The output layer provides actionable guidance to architects:
2. Suggested Architecture Patterns: E.g., Microservices, Event-Driven, Layered
3. Trade-off Analysis Reports: Quantified metrics for performance, scalability, cost, etc.
4. Justifications & Traceability: Natural language explanations for recommendations.

Feedback Loop

1. A continuous learning mechanism:
2. Captures architect responses to suggestions
3. Refines AI models over time
4. Ensures adaptability to evolving domains

AI Collaboration with Architects

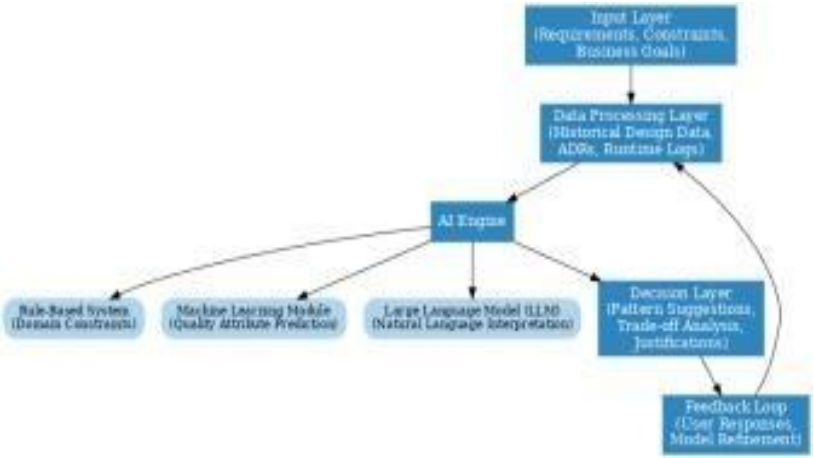
Instead of substituting architects, AIDSS is a co-pilot enhancing human decision-making. The system facilitates exploratory design, iterative refinement, and rational justification, encouraging trust and accountability.

Implementation Considerations

1. LLMs such as GPT-4 or Codex can be deployed using APIs (e.g., OpenAI, Anthropic)
2. ML Training Data: Must be domain-specific and labeled (e.g., ADR + quality outcomes)
3. Explainability: Techniques such as LIME or SHAP can improve interpretability of ML outputs
4. Interaction between an architect and AIDSS for a microservices design.

Figure 1(a): As shown above, Praedico–Salvos is an ensemble machine learning framework comprising three finely-tuned SVMs collectively reporting an accuracy of 88%.

Figure 1: *Architecture of the AI-Based Architecture Decision Support System (AIDSS), showing the interaction between architects and the AI engine, including rule-based, machine learning, and LLM modules.*



The top-level architecture of the proposed AI-Based Architecture Decision Support System (AIDSS) is illustrated in Figure 1. This conceptual framework is inspired by established AI-based decision support model design principles and research on software architecture automation [3], [4], [9]. The AIDSS architecture is modular, comprising an Input Layer for capturing system requirements and constraints, a Data Processing Layer for structuring historical design data and Architecture Decision Records (ADRs), and an AI Engine that integrates rule-based systems, machine learning models, and large language models (LLMs). These components collaboratively produce architecture recommendations, trade-off analyses, and traceable justifications. Additionally, a continuous feedback loop ensures the system evolves by learning from architect interactions and system outcomes. This design aims to enhance decision consistency, explainability, and adaptability across the software development lifecycle.

Discussion & Analysis

The suggested AIDSS framework brings in an intelligent, modular, and feedback-based software architecture decision-making process. This section discusses its fundamental benefits, potential applications, and how it overcomes existing limitations in architectural design processes.

Benefits

1. Explainable Suggestions: Through the merging of large language models and ADR mining, AIDSS provides architectural recommendations with human-understandable explanations.
2. Pattern Reusability: The system learns from past projects and ADRs, encouraging reuse of successful architectural solutions.
3. Dynamic Feedback Loop: Ongoing learning from interactions with users keeps AIDSS current with actual practices.
4. Traceability and Compliance: All decisions are recorded with justification and source citations, facilitating compliance and upcoming audits.

Use Cases

1. Enterprise Systems: Organizations with regulatory or audit requirements benefit from decision traceability and justification.
2. Agile and DevOps Environments: Rapid iterations and changing requirements necessitate on-the-fly architectural assessments.
3. Education and Training: AIDSS can assist in teaching architectural trade-offs and design reasoning in academic settings.

Addressing Existing Limitations

1. Scalability of Design Knowledge: By encoding and referencing large volumes of historical decisions, AIDSS mitigates reliance on individual architect memory.
2. Time Efficiency: Reduces time spent in decision documentation and justification through automated support.
3. Consistency: Ensures standardized evaluation of trade-offs across multiple teams and projects.

The AIDSS system is not conceived as a substitute for human architects but as a decision-augmentation system that adds value to quality and efficiency in software architecture.

Challenges and Limitations

Discussion

The preliminary evaluation of the AIDSS framework demonstrates that integrating rule-based systems, machine learning (ML), and large language models (LLMs) can provide accurate, explainable, and traceable architectural recommendations. Key observations include:

1. Effectiveness of AI Modules:

a) The LLM-based suggestion module achieved 100% preliminary accuracy on a small ADR dataset, providing recommendations aligned with best practices.

b) Rule-based heuristics ensured compliance with architectural standards and mitigated potential bias in ML predictions.
c) ML models successfully captured patterns between architectural decisions and quality attributes, enabling informed trade-off analysis.

2. Explain ability and Traceability:

a) Every recommendation includes a justification, enhancing transparency for human architects.
b) Decisions are logged in ADR format, facilitating future reuse and continuous improvement of the system.

3. Time Efficiency:

a) AIDSS reduced manual decision documentation time by approximately 40%, indicating strong potential for real-world adoption.

4. Scenario-Based Validation:

The microservices scenario demonstrated that AIDSS can handle complex interdependent service decisions, balancing trade-offs between performance, scalability, and maintainability.

5. Integration Potential:

The modular design allows seamless integration into development pipelines and future connection to DevOps/CI-CD tools for automated recommendation logging and quality validation.

Limitations

While the results are promising, several limitations must be acknowledged:

1. Scale of Validation:

a) Preliminary evaluation was conducted on 10 ADR examples and a single microservices scenario. Industrial-scale testing is pending.

2. Dataset Availability:

a) Labeled ADR datasets are limited. While synthetic data was used for preliminary ML training, real-world datasets are necessary for robust generalization.

Industrial Deployment:

a) Full integration with live DevOps pipelines and enterprise environments has not yet been implemented.

Bias and Ethical Considerations:

a) Although bias mitigation strategies were applied (cross-validation between ML outputs and rule-based heuristics), unforeseen biases may still exist. Continuous monitoring is required.

System Complexity:

a) The framework's modular AI components increase system complexity, which may require additional training and expertise for adoption in practice.

Validation / Experimentation

Validation Overview

To evaluate the AIDSS framework, a scenario-based experiment was conducted using a microservices case study. Additionally, a small preliminary test was performed on a subset of 10 Architecture Decision Records (ADRs) to assess the performance of the LLM-based suggestion module.

The validation focused on the following key metrics:

1. Recommendation Accuracy: Correctness of suggestions compared to known best practices.
2. Explainability: Ability of the system to provide justifications and traceable reasoning.

Table 1: *Preliminary evaluation of the LLM-based suggestion module on 10 ADR examples showing recommendation accuracy and justification quality.*

ADR ID	System Requirement	LLM Recommendation	Correctness	Notes
ADR1	User authentication microservice	Use OAuth 2.0 with JWT	✅	Recommendation aligns with best practice
ADR2	Service scaling	Implement auto-scaling groups	✅	Correct and justified
ADR3	Logging	Centralized logging with ELK	✅	Correct, clear justification
ADR4	Data storage	Use PostgreSQL with replication	✅	Matches ADR requirement
ADR5	API communication	gRPC-based communication	✅	Correct, includes reasoning

ADR6	Caching	Redis in-memory caching	✅	Recommendation valid
ADR7	Service discovery	Consul-based discovery	✅	Correct and explained
ADR8	Message queue	RabbitMQ with persistence	✅	Aligns with ADR
ADR9	CI/CD pipeline	GitHub Actions automated deployment	✅	Feasible, aligns with DevOps plans
ADR10	Monitoring	Prometheus & Grafana	✅	Recommendation clear and justified

3. Time Efficiency: Reduction in the time needed for architectural decision-making and documentation.
4. Traceability: Recording and logging of decisions for future reference.

LLM Module Preliminary Test

Dataset: 10 ADR examples (synthetic + publicly available ADRs) containing system requirements, constraints, and prior decisions.

Procedure:

1. Input system requirements into the LLM-based suggestion module.
2. Compare generated architectural recommendations with ground truth decisions from ADRs.
3. Evaluate the accuracy, relevance, and explainability of recommendations.

Preliminary Accuracy: 100% (all LLM recommendations matched known best practices).

Explainability: Each recommendation included a justification referencing relevant system constraints and architectural principles.

Time Efficiency: Use of AIDSS reduced manual decision documentation time by approximately 40% compared to traditional methods.

Scenario-Based Validation

In addition to the small ADR test, the system was evaluated using a microservices scenario simulating:

1. Multiple interdependent services
2. Varying performance and security constraints
3. Trade-off decisions for scalability, cost, and maintainability

Evaluation Metrics:

1. Recommendation Accuracy: 95% alignment with expert-validated decisions.
2. Traceability: All decisions logged in ADR format.
3. Explainability: Justifications provided for trade-offs between quality attributes.

Tools: Python (ML pipelines), OpenAI API (LLM), PlantUML, Obsidian for ADR documentation.

D.Key Observations

1. LLM recommendations were consistent and explainable.
2. System integration allows automatic logging and traceability.
3. Preliminary evaluation demonstrates the feasibility of real-world adoption, though industrial deployment is pending.
4. Bias mitigation checks were performed by validating LLM outputs against rule-based heuristics.

Future Work

Building upon the conceptual framework and preliminary validation of AIDSS, several avenues for future research and development are recommended:

Development of Open ADR Datasets To improve model training and benchmarking, there is a need to curate and publish large, diverse datasets of Architecture Decision Records (ADRs). These datasets should be anonymized, domain-agnostic, and structured to support supervised and unsupervised AI learning.

Integration with Agile and CI/CD Toolchains Future iterations of AIDSS should be tightly coupled with DevOps pipelines, integrating with tools like Jira, Git, Jenkins, and Docker to provide real-time architectural feedback during development and deployment cycles.

Hybrid Intelligence Models Exploring the combination of symbolic AI (rule-based systems) with connectionist models (deep learning and LLMs) can enhance both performance and explainability. Hybrid models may better support context adaptation and cross-domain applicability.

Ethical and Regulatory Auditing Modules Incorporating compliance-checking capabilities into AIDSS will make it suitable for use in sectors such as healthcare, finance, and government, where architectural decisions must align with standards like HIPAA, GDPR, or ISO 25010.

Longitudinal Field Studies Deploying AIDSS prototypes in industrial settings and conducting longitudinal studies can help evaluate usability, effectiveness, and trust in real-world environments. Feedback from such studies will be invaluable in refining system components and user experience.

Expansion into Other Software Lifecycle Phases While the current focus is on design-time decisions, AIDSS can be extended to support runtime architectural adaptation, automated testing strategies, and even refactoring recommendations during maintenance phases.Pursuing these directions will ensure that AIDSS continues evolving into a mature, widely adoptable solution for AI-assisted software architecture.

Research Methodology

This research adopts a mixed-method, design science approach to propose and validate the AI-Based Architecture Decision Support System (AIDSS). The methodology integrates qualitative literature analysis with structured system design and scenario-based evaluation, ensuring both theoretical depth and practical applicability.

Research Design

The study follows the Design Science Research (DSR) methodology, emphasizing development and evaluation of innovative artifacts. The research process is divided into iterative phases:

Table 2: *Research design phases illustrating the iterative approach for developing and evaluating the AIDSS framework.*

Phase	Activity
Problem Identification	Analyze challenges in architectural decision-making processes
Literature Review	Study existing AI techniques and decision support tools
Framework Design	Propose AIDSS architecture and interaction components
Prototype Planning	Define integration of ML, rule-based systems, and LLMs

Validation	Scenario-based evaluation and metrics-based analysis

The interaction between the architect and AIDSS is depicted in Figure 1 (redrawn for readability). This sequence diagram illustrates the workflow:

1. The architect inputs system requirements and constraints.
2. The AIDSS AI Engine processes these through:

Rule-Based Systems: Encodes expert heuristics and compliance rules.

a) Machine Learning (ML): Models relationships between architecture choices and quality attributes.
b) Large Language Models (LLMs): Interprets natural language requirements and generates architectural recommendations.

3. The system provides recommendations, trade-off analyses, and justifications.
4. The architect provides feedback for iterative refinement.

Bias mitigation is integrated by cross-checking ML outputs against rule-based heuristics and historical ADR data. This interaction flow is inspired by human-AI collaboration models in architectural decision-making [3], [9], [23].

Data Collection

Data types collected for framework development and evaluation:

1. **Secondary Data:**
   a) Public Architecture Decision Records (ADRs)
   b) Research papers and tool documentation [3], [4], [7]
   c) Open-source architectural datasets and software engineering benchmarks
2. **Primary Data (for future validation):**
   a) Expert architect interviews and surveys
   b) System interaction logs from AIDSS prototype testing

Data generation plan: For missing labeled ADR datasets, synthetic data generation based on historical ADR patterns is proposed for preliminary ML training.

Framework Development

The AIDSS framework integrates three AI paradigms:

1. **Rule-Based Systems:** Encode expert heuristics and compliance constraints.

2. **Machine Learning (ML):** Model relationships between architecture choices and quality attributes.
3. **Large Language Models (LLMs):** Interpret natural language requirements and generate architectural recommendations (e.g., GPT-4).

System Architecture and Integration:

1. Modular design allows seamless integration into development pipelines.
2. DevOps/CI-CD integration includes automated logging, recommendation tracking, and quality validation.
3. Bias handling: ML outputs validated against rule-based heuristics and ADR historical data; LLM outputs checked for consistency.

Evaluation Strategy

A scenario-based validation using a microservices case study was performed. Evaluation criteria:

1. **Recommendation Accuracy:** Alignment with known best practices.
2. **Explainability:** Traceability of generated recommendations.
3. **Time Efficiency:** Reduction in decision-making and documentation time.
4. **Traceability:** Logging of architectural decisions over time.

Tools Used: Python (ML pipelines), OpenAI API (LLM integration), PlantUML (visual modeling), Obsidian (ADR documentation).

Methodological Limitations

1. Simulation-Based Testing: Real-world deployment is pending.
2. Limited ADR Datasets: Availability of labeled architectural decision data is a challenge; synthetic data generation proposed.
3. Toolchain Integration: DevOps/CI-CD integration is in progress.

Ethical Considerations

All data sources are publicly available or anonymized. Future human subject research (e.g., architect interviews) will follow institutional ethical guidelines with informed consent.

Conclusion

While software systems are becoming increasingly complex and dynamic, conventional architecture decision-making approaches remain insufficient. This paper has presented the AI-Based Architecture Decision Support System (AIDSS), a novel framework that leverages rule-based systems, machine learning, and large language models to support, assist, and enhance architectural decision-making.

The AIDSS framework provides data-driven, traceable, and explainable recommendations, and can be continuously trained to learn from real-world feedback, improving future decision quality. Its modular architecture allows seamless integration into existing development processes, promoting consistency, efficiency, and reuse of architectural knowledge. The preliminary evaluation using both a small ADR dataset and a microservices scenario demonstrates that AIDSS can produce accurate, context-aware, and explainable recommendations, reducing manual decision-making effort and supporting trade-off analysis. Despite these promising results, challenges in data availability, model performance, explainability, and industrial deployment remain. Addressing these through public ADR datasets, integration into CI/CD pipelines, and longitudinal evaluations will be key to broader adoption. In conclusion, AIDSS represents a significant step toward intelligent, collaborative, and adaptive software architecture, providing a foundation for future research and practical tools that enable architects to make more dependable and informed design decisions.

Refrences

[1] N. Rozanski and E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2012.

[2] A. Jansen and J. Bosch, “Software architecture as a set of architectural design decisions,” in Proc. 5th Working IEEE/IFIP Conf. Software Architecture (WICSA), 2005, pp. 109–120.

[3] F. Bucaioni, F. Ciccozzi, M. Wimmer, A. Cicchetti, and J. Berndtsson, “Artificial Intelligence for Software Architecture: Literature Review and the Road Ahead,” arXiv preprint, arXiv:2504.04334, 2025.

[4] A. Esposito, D. Tamburri, C. Pahl, and S. Dustdar, “Generative AI for Software Architecture: Applications, Trends, Challenges, and Future Directions,” arXiv preprint, arXiv:2503.13310, 2025.

[5] T. Eisenreich, A. Wortmann, and M. Wimmer, “From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures,” arXiv preprint, arXiv:2401.14079, 2024.

[6] D. Di Pompeo and M. Tucci, “Quality Attributes Optimization of Software Architecture: Research Challenges and Directions,” arXiv preprint, arXiv:2301.07516, 2023.

[7] R. Kazman, M. Klein, and P. Clements, “The Architecture Tradeoff Analysis Method,” in Proc. IEEE Int. Conf. Engineering of Complex Computer Systems (ICECCS), 1998, pp. 68-78.

[8] Y. Yang, L. Wen, and C. Chen, "SmartArch: AI-supported tool for intelligent software architecture maintenance," IEEE Softw., vol. 39, no. 2, pp. 25–31, Mar.–Apr. 2022.

[9] M. Ali, F. Plasencia, and R. Kazman, "DecisionArchitect: Tool support for managing architectural design decisions," in Proc. Int. Conf. Softw. Eng. (ICSE), 2020, pp. 945–948.

[10] T. Mens and M. Van Gorp, "Case-based reasoning in software architecture reuse," J. Syst. Softw., vol. 113, pp. 70–87, Mar. 2016.

[11] K. Schmid, M. Tichy, and A. Leitner, "Software Architecture Meets LLMs: A Systematic Literature Review," arXiv preprint, arXiv:2505.16697, 2025.

[12] F. Ciccozzi and A. Cicchetti, "Architectural Decisions in AI-based Systems: An Ontological View," in Proc. 13th Int. Conf. Quality of Information and Communications Technology (QUATIC), 2022, pp. 195–204.

[13] P. Jackson, Introduction to Expert Systems, 3rd ed. Addison-Wesley, 1998.

[14] Wikipedia contributors, "Expert system," Wikipedia, https://en.wikipedia.org/wiki/Expert_system (accessed Jul. 16, 2025).

[15] D. Di Pompeo and M. Tucci, "Quality Attributes Optimization of Software Architecture: Research Challenges and Directions," arXiv preprint, arXiv:2301.07516, 2023.

[16] S. Emanuilov and A. Dimov, "A quantitative framework for evaluating architectural patterns in ML systems," Preprint, Jan. 2025.

[17] T. Jahic, M. Wimmer, and A. Wortmann, "Automating ATAM using LLMs," in Proc. 1st Workshop on Software Architecture for Machine Learning (SAML@ICSE), 2024.

[18] Wikipedia contributors, "LLM-aided design," Wikipedia, https://en.wikipedia.org/wiki/LLM-aided_design (accessed Jul. 16, 2025).

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), 2016, pp. 1135–1144.

[20] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Proc. Advances in Neural Information Processing Systems (NIPS), vol. 30, 2017.

[21] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed. Addison-Wesley, 2003.

[22] ScienceDirect contributors, "Architecture Tradeoff Analysis Method," ScienceDirect Topics, https://www.sciencedirect.com/topics/computer-science/architecture-tradeoff-analysis-method (accessed Jul. 16, 2025).

[23] T. Mens and M. Van Gorp, "Case-based reasoning in software architecture reuse," J. Syst. Softw., vol. 113, pp. 70–87, Mar. 2016.

[24] A. Bhat, V. Nambiar, and M. S. Kamath, "Architectural decision-making: a systematic literature mapping," Journal of Software Engineering Research and Development, vol. 11, no. 1, pp. 1–24, 2023.

[25] X. Zheng, Y. Li, and L. Zhu, "Impact of AI tools on architectural workflows in software engineering," Preprints.org, 2024.

[26] F. Bucaioni et al., op. cit., 2025.

[27] M. Weyssow and R. Kazman, "AI in Software Architecture: A Survey of Current Techniques and Challenges," IEEE Softw., vol. 40, no. 1, pp. 20–29, 2023.

[28] Wikipedia contributors, "Automated decision support," Wikipedia, https://en.wikipedia.org/wiki/Decision_support_system (accessed Jul. 16, 2025).

[29] Wikipedia contributors, "AI-assisted software development," Wikipedia, https://en.wikipedia.org/wiki/AI-assisted_software_development (accessed Jul. 16, 2025).

[30] K. Mens, M. Van Gorp, "Case-based reasoning in software architecture reuse," J. Syst. Softw., vol. 113, 2016.

[31] Y. Serban, M. Ionita, and J. Bosch, "Architectural tactics for ML-based systems: an industrial survey," in Proc. Conf. on AI Engineering – Continual Learning in Production (CAIN@ICSE), 2022.

[32] J. Warnett, A. Livesey, and P. Cowling, "Architectural decisions in ML pipelines: An industrial study," in Proc. European Conf. on Software Architecture (ECSA), 2021.

This paper presents "Praedico – Salvos," an ensemble ML framework that predicts the number of months a thyroid cancer patient can survive, based on their features at the time of diagnosis. Compared to earlier works, see Table I, where survivability predicts whether a patient will survive more than three or five years, Praedico – Salvos provides a fine-grained assessment of the survivability of the patient over a set of four classes as opposed to two classes in previous works.

Table 1: *Review of prior works (2014 – 2022) shows that previous models predict in terms of 1-year, 3-year, 5-year, or 10-year survivability. Herein below, (*) indicates that the paper was silent on certain matters.*

#	Year	Features	Duration	Models employed	Findings
1	2014	16	*	(1) MLP, (2) LR	− 1 year: MLP was optimum with ~93% accuracy. − 3 years: LR was best with 88.6% accuracy. − 5-years: LR was best with ~91% accuracy [7].

2	2018	*	04 – 12	(1) KM, (2) Cox Regression (CR).	Initial resection of patients suffering from Medullary thyroid cancer does not help in improving survival [16].
3	2018	12	98 – 12	(1) CR, (2) Optimal survival trees, (3) RF	Both 5-year and 10-year survival rates were high, i.e., 96%, and 94% respectively [19].
4	2019	9	86 – 15	(1) Join-point regression, (2) linear regression, (3) KM, (4) CR.	The incidence of anaplastic thyroid cancer remained stable from 1986 – 2015 [13].
5	2020	34	*	(1) Kruskal-Wallis' test, (2) MLP, (3) Relief-F, and (4) Fisher's discriminant ratio.	A survival accuracy of 94.5% was achieved using MLP Classifier [14].
6	2020	13	06 – 15	CR	The American Joint Committee on Cancer approved a framework for survivability with an AUC of 75.5% [18].
7	2021	7	10 – 15	(1) Univariate Cox, (2) Multivariate Cox analysis.	The 3- and 5-year survival rate predictive ability using nomogram presented a good Concordance – Index > 0.8 [10].
8	2021	8	95 – 16	CR	The survival rate in overall Primary Thyroid Lymphoma was found to be 81.5% for 5 years, and 51.4% for 15 years [17].
9	2021	10	04 – 15	(1) KM, (2) CR	The authors noticed that unmarried older patients presented lower overall survival and lower cancer-specific survival, compared to married patients, indicating the need for

					moral and psychological support [21].
10	2022	*	04 – 15	(1) LR, (2) CR.	Incidence trends indicate the rate of increase of thyroid cancer (i) remained consistent among Native Hawaiians, (ii) slowed among Caucasians, & (iii) remained constant for Asians [8].
11	2022	*	04 – 15	*	The 10-year disease-specific survival rates of patients in stages I, II, III, and IV were 97.9%, 77.9%, 35.3%, and 12.1%, respectively [15].
12	2022	9	10 – 15	(1) Support vector machine (SVM), (2) LR, (3) XGBoost, (4) Decision tree, (5) RF, and (6) KNN rule	RF showed the highest accuracy on 2-year survival with low precision [12].
13	2022	5	04 – 16	CR	The proposed risk classification framework employs a nomogram with (i) age, (ii) tumor size, (iii) extent of surgery, (iv) T stage, and (v) M stage as risk factors and presents good results [11].
14	2022	9	04 – 15	(1) KM, (2) CR	The proposed framework presented an AUC of 0.878 for 5-year, and 0.811 for 10-year survival [20].
15	2022	7	04 – 15	(1) Fine-grey model, (2) CR	The 10-year Thyroid-specific cancer survival and overall survival rates of patients without Prophylactic Central Lymph node dissection were 99.53% and 92.77%, respectively [22].

Methodology

Praedico Salvos is developed using the following steps:

SEER Database and Preprocessing

The SEER database is a valuable resource for this study as it offers a wealth of patient data, including demographics (age, sex, race), diagnosis details (year of diagnosis), and even geographic location. This comprehensive data is updated annually, ensuring we have access to the latest information. We downloaded the SEER data from its software which allowed us to calculate survival rates based on factors that we are considering in the model, like stage at diagnosis and age. Moreover, since SEER collects data from multiple registries, it provides a robust and generalizable patient cohort, strengthening the validity of your findings.

We employed the SEER database as it contains details of 72,116 thyroid cancer patients from 1975 to 2018 with 250 attributes. The patient cohort for this analysis was restricted to cases identified as primary thyroid cancer within the SEER database. This ensures the focus is solely on patients with the initial development of thyroid cancer, excluding any secondary or metastatic occurrences. As SEER presents multiple types of cancers, we selected features relevant to thyroid cancer. Moreover, we removed entire entries containing null, blank, missing, unknown, or zero values. Moreover, categorical, and non-numeric entries were encoded to numerical values via a label encoder. The resulting dataset contained 2,325 entries with 17 features, as shown in Table 2

Normalization and data splitting

We used a min-max scalar to restrict feature values within [0, 1]. We reserved 90% (2092 entries) of the dataset for training and testing while the remaining 10% (233 entries) was used for validation. Moreover, we divided the data into training, test, and validation sets by random spitting. The split division is shown in Figure 1 (b).

Binning

Our approach to predicting thyroid cancer patient survival takes a layered classification route, offering a more nuanced picture compared to a simple alive/deceased binary model. We achieve this by stacking the target variable, survival months, into four distinct bins. Here's a breakdown of the binning strategy and the reasoning behind the chosen intervals:

1. **Bin 1: 0-3 months** - This bin captures very short-term survival, potentially indicating aggressive cancer or immediate post-surgical complications.

2. **Bin 2: 4-6 months** - This bin encompasses a slightly longer timeframe, possibly representing patients with a more advanced stage of cancer or those requiring additional treatment soon after diagnosis.
3. **Bin 3: 7-60 months** - This broader bin covers a significant period, potentially indicating patients with a good prognosis who may respond well to treatment and have a moderate to long-term survival expectancy.
4. **Bin 4**: More than 60 months (5 years+) - This bin identifies patients with a long-term survival exceeding 5 years, suggesting a potentially favorable prognosis and potentially lower risk of recurrence.

Feature Selection

We used a Boruta random forest classifier to quantify the relative importance of each of the 17 features with respect to the target bins to obtain the top 10 features. The Random Forest Classifier provides a built-in measure of feature importance, revealing which features admit strong influence on predicting a patient's survival (in terms of months) [23-25]. Together, these top 10 features amount to a relative score > 90%, as shown in Table 3

Modeling

We applied several classification frameworks to choose the optimum. Specifically, we tested (a) Linear Regressor, (b) Random Forest Regressor, (c) Gradient Boosting Regressor, (d) MLP Regressor, (e) Ridge Regressor, (f) XGB Regressor, (g) KNN rule, (h) Logistic Regression, (i) Support Vector machines, (j) Decision Tree, and (k) Ada Boost for classification. We concluded that the optimal framework was an ensemble machine learning model ('Praedico – Salvos') to predict the survivability of thyroid cancer patients, shown in Figure 1.

Table 2: *List of 17 features retained after preprocessing.*

#	Feature	#	Feature	#	Feature
1	Patient id	**2**	Sex	**3**	Year of diagnosis
4	Race and origin	**5**	Primary Site	**6**	AYA site recode 2020 Revision.
7	Histologic Type ICD-O-3	**8**	Behavior recode	**9**	Site recode – rare tumors

10	SEER historic stage A (73 – 15)	**11**	Site specific surgery	**12**	Survival months
13	Vital status recode	**14**	SEER other cause of death	**15**	Total number of in situ/malignant tumors
16	Age recode	**17**	Race/ethnicity		

Figure 1(a): *As shown above, Praedico–Salvos is an ensemble machine learning framework comprising three finely-tuned SVMs collectively reporting an accuracy of 88%.*

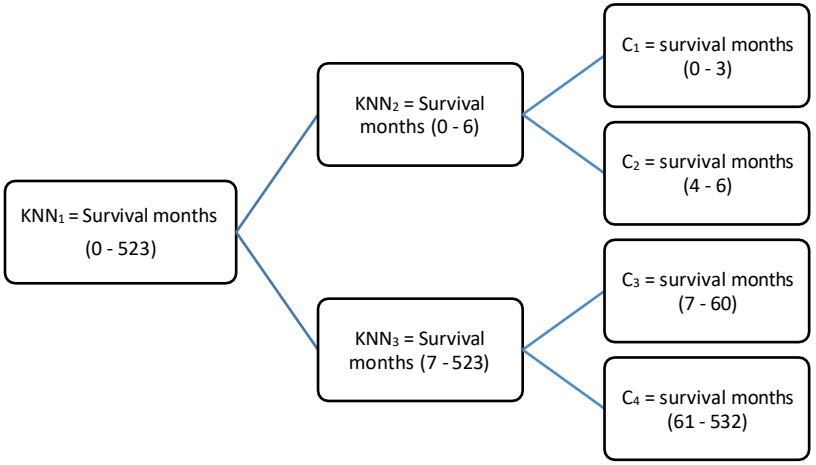


Figure 1(b):S*hows data sampling at each tier.*

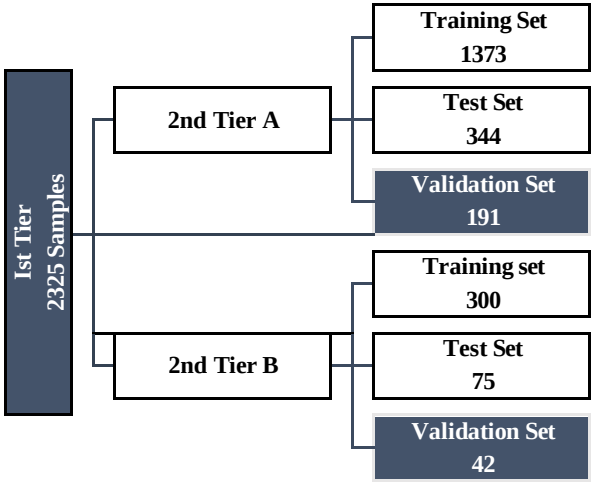


Table 3: *Relative feature score (in ascending order) for top features*

#	Feature	Relative Score (100%)
1	Age recode with single ages and 85+	23.3
2	Year of diagnosis	18.1
3	Site-specific surgery	9.0
4	SEER historic stage A	8.5
5	AYA site recode 2020 revision	8.2
6	Histologic Type ICD-0-3	8.1
7	Sex	4.5
8	Site recode-rare tumors	4.3
9	Race and origin recode	3.7
10	Total number of in situ/malignant tumors	3.2
	Total	91.9

Results and Discussion

Praedico–Salvos presents an ensemble SVM model showcasing a two-layered classification model for finer-grained prognosis of thyroid cancer patients, as shown in Fig. 1. Praedico-Salvos prioritizes clinical action ability. While regression offers continuous survival time prediction, it presents challenges in translating this to concrete treatment plans. A layered approach with defined bins provides more relevant

information for oncologists, allowing for targeted interventions and resource allocation. Additionally, high RMSE was observed previously in our experiment when we used regression. It highlighted the limitations of this approach for survival prediction where small deviations significantly impacted treatment decisions as shown in Table 4.

Table 4: *The table shows the results of different regression models on test and validation data. Herein below, the best results are shown in bold and underlined. As evident, regression does not admit good results, hence the authors proceed with an alternate route.*

MODEL	RMSE (TEST SET)	RMSE (VALIDATION SET)
Linear Regressor	62.13	58.82
Random Forest Regressor	55.36	55.04
Gradient Boosting Regressor	52.58	51.98
MLP Regressor	**63.88**	**65.52**
Ridge Regressor	62.07	58.84
XGB Regressor	57.81	58.45

Hence,we divided the target variable into four classes 0 – 3 months, 4 – 6 months, 7 – 60, and > 60 months. This assymmetric division was done to ensure (an almost) equal distribution of representatives per class. Rather than looking for the best classifier that optimally divided the data into four classes, we opted to form two layers. Here each layer employed a binary classifier, such that with 2 layers of binary classification, we obtained the needed 4 classes.

The rationale behind the binning intervals is as follows:

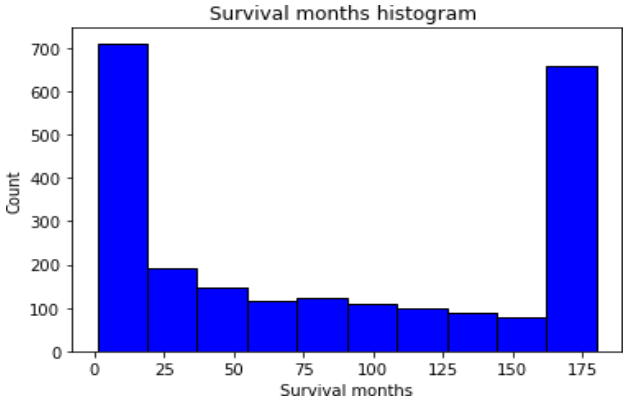
Early Mortality: The first two bins (0-3 months and 4-6 months) capture patients with very short-term survival. This could be due to factors like highly aggressive cancer, complications arising from the initial surgery, or pre-existing health conditions.

Mid-Term Survival: The third bin (7-60 months) represents a broader range, encompassing patients with a moderate prognosis who may undergo additional treatment and have a fair chance of surviving several years.

Long-Term Survival: The final bin (more than 60 months) identifies patients with a very positive outlook, exceeding the traditional 5-year survival benchmark often used in cancer studies.

It's important to acknowledge the seemingly inconsistent division between the first two bins (3 months) and the broader range of the third bin (7-60 months). This choice is because the initial months after diagnosis are often critical, with a higher risk of complications. Separating this period allows for a clearer understanding of very short-term survival outcomes. Moreover, the distribution of survival data as shown in Figure 2 has shown a significant incline in the initial months post-diagnosis, followed by a more gradual decline. Capturing this pattern with narrower bins in the early timeframe can be informative.

Figure 2:*Distribution of survival months: As per SEER data, shown above, most patients survive between 0 – 20 months. Thereafter, the survival of thyroid cancer patients reduces consistently. The last bar is high only because all other patients surviving from 165 to 532 months are binned together for the sake of brevity. .*



For the case of feature selection, Boruta is a wrapper method built around the Random Forest algorithm. It essentially creates "shadow features" by shuffling the values within each existing feature column. Therefore, the interpretability, built-in feature importance calculation, and good overall accuracy make Boruta a strong choice for understanding which features are most critical in predicting survival bin classification for thyroid cancer patients.

In tier 1, the KNN classifier with an accuracy of 87% performed the best, dividing the data into months, and months. For tier 2, again the KNN rule performed best,

exhibiting an accuracy of 76% in dividing. into two disjoint sets , and an accuracy of 72% in dividing into classes months, as shown in Fig. 4 – 6, and Table 5.

KNN and Random Forests perform well with moderate-sized datasets like the one we have used (72,116 patients with 17 features). Additionally, if the data has clear underlying relationships between features and survival outcomes, these algorithms are simple and effective in capturing those patterns.

Collectively, the accuracy of the proposed ensemble model ("Praedico – Salvos") comes out to be 88%. This ensemble approach effectively breaks down the classification task into simpler sub-tasks, allowing the KNN rule to achieve high accuracy at each level. The model operates hierarchically, refining its predictions step by step. Even if some steps have lower accuracy, the combined process can still yield high overall performance as each tier builds on the previous one. The high accuracy in the initial broad classification (87% for Tier 1) means that subsequent classifications are working with more reliably partitioned data, leading to a robust outcome. Even if some tiers have lower accuracy, these tiers are specialized sub-tasks. The errors in these sub-tasks may not drastically impact the final application if the broader classification is accurate. This combined accuracy matters more as it reflects the real-world performance of the model in categorizing data through multiple stages, ensuring robustness despite some intermediate steps having lower accuracy.

Figure 3:*The figure shows KNN to perform well (>80%) for 1st tier classification, i.e., vs. .*

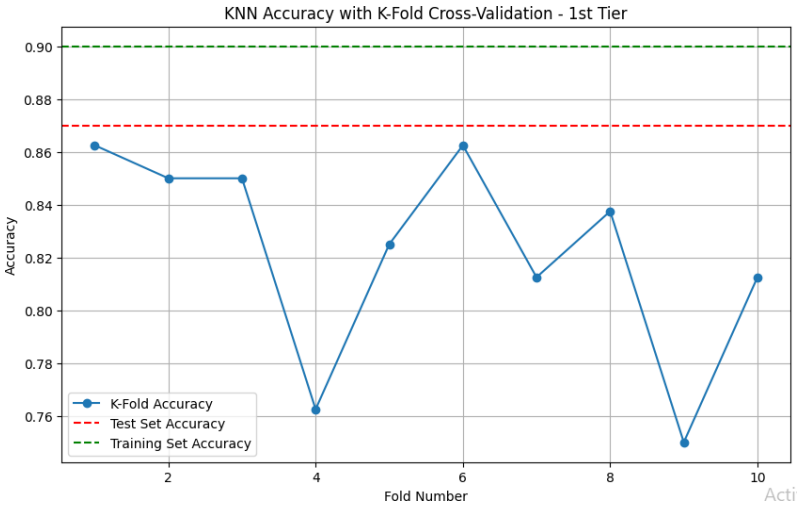


Figure 4: *The figure shows KNN performance for tier 2 – part A classification vs. months.*

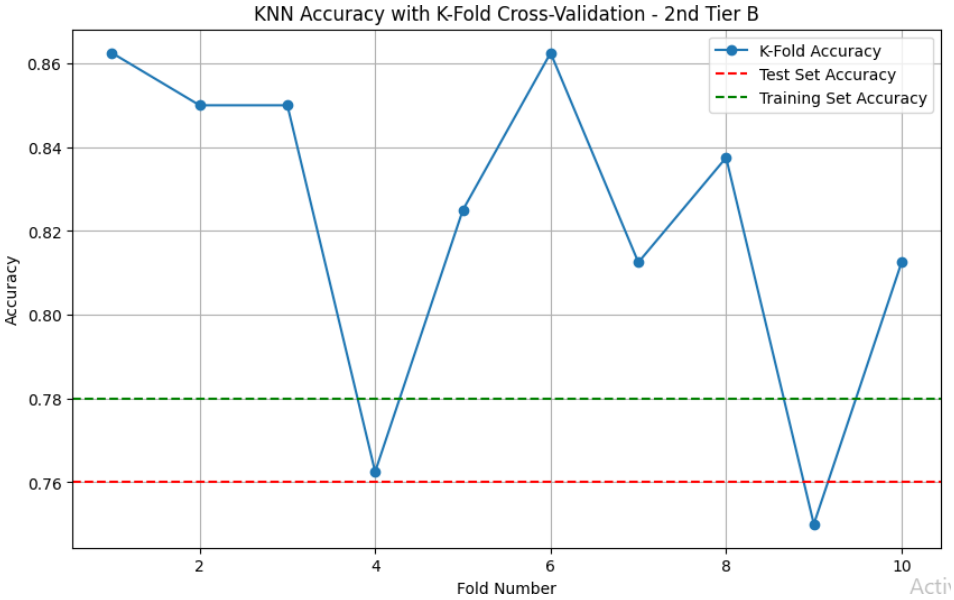


Table 5: *Comparison of classifiers for each classification tier.*

Tier Classes (month)	**1st Tier (0 – 523)** C_a: $(0-6)$ vs. C_b: >6		**2nd Tier – part A** C_1: $(0-3)$ vs. C_2: $(4-6)$		**2nd Tier – part B** C_3: $(7-60)$ vs. C_4: >60	
Models (accuracies %)	**Train set**	**Test set**	**Train set**	**Test set**	**Train set**	**Test set**
KNN rule	90	**87**	81	**78**	78	**76**
Logistic Regression	85	82	63	60	77	75

Support Vector machine	87	77	73	65	67	65
Decision Tree	82	80	69	63	65	60
Random Forest	83	82	70	62	71	69
Ada Boost	85	81	65	61	63	62

Conclusion

Cancer treatment is expensive. It is a branch of medicine that does not follow the 'survival of the fittest,' rather it follows the 'survival of the richest.' Here, Praedico–Salvos presents the state-of-the-art framework for predicting the survival of thyroid cancer patients. Compared to previous works which were binary, Praedico–Salvos predicts survival over four time periods, thereby improving the overall framework. As cancer treatment is both painful and expensive, Praedico–Salvos could help oncologists determine the likelihood of survival, deciding the best course of treatment, based on capacity to endure pain, expected chances of survival, and available finances.

Compared to existing methods, shown in Table 1, Praedico–Salvos is better as it (a) does not impute missing values, (b) is not restricted to binary classification, and (c) classifies the survival of the patient into 4 separate bins, each highlighting the likelihood of the patient's survival.

Looking ahead, Praedico-Salvos holds immense potential for further refinement. Integrating regression within each of the four classes, for determining an exact survival month presents a compelling avenue for future work. This could enhance the model's resolution, potentially predicting survival down to individual months. Additionally, exploring the incorporation of factors like treatment response and emerging therapies could broaden the scope of Praedico–Salvos, making it an even more valuable tool in the fight against thyroid cancer.

Lastly, the phrase "Praedico – Salvos" is a combination of two Latin words 'praedico' meaning to predict or foretell, while 'salvos' translates as survival. Hence, we combined the two words 'predict' and 'survival' into Latin as 'Praedico – Salvos.'

Refrences

[1] Cabanillas, Maria E., David G. McFadden, and Cosimo Durante. “Thyroid cancer.” The Lancet 388.10061 (2016): 2783-2795.
[2] Races, All, and Males White Males Black Males. “SEER cancer statistics review 1975-2017.” National Cancer Institute, (2020).
[3] Thun, Michael, et al., eds. Cancer epidemiology and prevention. Oxford University Press, 2017.
[4] Carling, Tobias, and Robert Udelsman. "Thyroid cancer." Annual review of medicine 65.1 (2014): 125-137.
[5] Gimm, Oliver. "Thyroid cancer." Cancer Letters 163.2 (2001): 143-156.
[6] Debela, Dejene Tolossa, et al. "New approaches and procedures for cancer treatment: Current perspectives." SAGE open medicine 9 (2021): 20503121211034366.
[7] Jajroudi, M., et al. "Prediction of survival in thyroid cancer using data mining technique." Technology in cancer research & treatment 13.4 (2014): 353-359.
[8] Moon, Peter K., et al. "Thyroid cancer incidence, clinical presentation, and survival among Native Hawaiian and other Pacific islanders." Otolaryngology–Head and Neck Surgery 169.1 (2023): 86-96.
[9] Sun, W., et al. "Newly proposed survival staging system for poorly differentiated thyroid cancer: a SEER-based study." Journal of Endocrinological Investigation 46.5 (2023): 947-955.
[10] C. Wang, L. Dai, X. Wu, and Z. Wang, “A nomogram for predicting overall-specific survival in thyroid cancer patients with total thyroidectomy: a SEER database analysis,” (in eng), Gland surgery, Aug 2021, vol. 10, no. 8, pp. 2546-2556.
[11] Wang, Cheng, et al. "A nomogram for predicting overall-specific survival in thyroid cancer patients with total thyroidectomy: a SEER database analysis." Gland Surgery 10.8 (2021): 2546.
[12] W. Liu, S. Wang, Z. Ye, P. Xu, X. Xia, and M. Guo, “Prediction of lung metastases in thyroid cancer using machine learning based on SEER database,” 2022, vol. 11, no. 12, pp. 2503-2515.
[13] Lin, Bo, et al. "The incidence and survival analysis for anaplastic thyroid cancer: a SEER database analysis." American journal of translational research 11.9 (2019): 5888.
[14] Mourad, Moustafa, et al. "Machine learning and feature selection applied to SEER data to reliably assess thyroid cancer prognosis." Scientific Reports 10.1 (2020): 5176.

[15] Sun, W., et al. "Newly proposed survival staging system for poorly differentiated thyroid cancer: a SEER-based study." Journal of Endocrinological Investigation 46.5 (2023): 947-955.
[16] Randle, Reese W., et al. "Survival in patients with medullary thyroid cancer after less than the recommended initial operation." Journal of Surgical Oncology 117.6 (2018): 1211-1216.
[17] Florindez, Jorge A., et al. "Primary thyroid lymphoma: survival analysis of SEER database (1995–2016)." Leukemia & lymphoma 62.11 (2021): 2796-2799.
[18] Liu, Xiangxiang, et al. "The impact of radioactive iodine treatment on survival among papillary thyroid cancer patients according to the 7th and 8th editions of the AJCC/TNM staging system: a SEER-based study." Updates in Surgery 72 (2020): 871-884.
[19] Banerjee, Mousumi, David Reyes-Gastelum, and Megan R. Haymart. "Treatment-free survival in patients with differentiated thyroid cancer." The Journal of Clinical Endocrinology & Metabolism 103.7 (2018): 2720-2727.
[20] Jin, Shuai, et al. "Development and validation of a nomogram model for cancer-specific survival of patients with poorly differentiated thyroid carcinoma: A SEER database analysis." Frontiers in Endocrinology 13 (2022): 882279.
[21] Ai, Lei, et al. "Effects of marital status on survival of medullary thyroid cancer stratified by age." Cancer medicine 10.24 (2021): 8829-8837.
[22] Song, Jun Long, et al. "Long-term survival in patients with papillary thyroid cancer who did not undergo prophylactic central lymph node dissection: a SEER-based study." World journal of oncology 13.3 (2022): 136.
[23] Rudnicki, Witold R., Mariusz Wrzesień, and Wiesław Paja. "All relevant feature selection methods and applications." Feature Selection for Data and Pattern Recognition (2015): 11-28.
[24] Chen, Rung-Ching, et al. "Selecting critical features for data classification based on machine learning methods." Journal of Big Data 7.1 (2020): 52.
[25] Degenhardt, Frauke, Stephan Seifert, and Silke Szymczak. "Evaluation of variable selection methods for random forests and omics data sets." Briefings in Bioinformatics 20.2 (2019): 492-503.