

Sam PAUL

Thesis_SamriddhiPaul (1) (1)

 M.TECH. THESIS 2

Document Details

Submission ID

trn:oid:::27535:145033020

Submission Date

Jul 6, 2026, 11:00 PM GMT+5:30

Download Date

Jul 6, 2026, 11:17 PM GMT+5:30

File Name

Thesis_SamriddhiPaul (1) (1).pdf

File Size

2.1 MB

48 Pages

16,135 Words

90,893 Characters

8% Overall Similarity

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

Filtered from the Report

- Bibliography
- Quoted Text
- Small Matches (less than 8 words)

Exclusions

- 14 Excluded Matches

Match Groups

-  **126 Not Cited or Quoted 8%**
Matches with neither in-text citation nor quotation marks
-  **5 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

- 5%  Internet sources
- 4%  Publications
- 6%  Submitted works (Student Papers)

Match Groups

- 126 Not Cited or Quoted 8%**
Matches with neither in-text citation nor quotation marks
- 5 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

- 5% Internet sources
- 4% Publications
- 6% 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	The University of the West of Scotland on 2026-04-23	<1%
2	Internet	arxiv.org	<1%
3	Internet	ijifr.org	<1%
4	Internet	mdpi-res.com	<1%
5	Internet	idr.nitk.ac.in	<1%
6	Publication	Jamuna S Murthy, G M Siddesh. "Introduction to Generative AI - Theoretical Foun...	<1%
7	Student papers	Sim University on 2026-04-17	<1%
8	Student papers	Chitkara University on 2026-05-09	<1%
9	Student papers	Istanbul Aydin University on 2026-04-10	<1%
10	Student papers	University of Northumbria at Newcastle on 2026-05-11	<1%

11	Internet	library.oapen.org	<1%
12	Internet	link.springer.com	<1%
13	Publication	Areejit Phukan, Manoj Kalava, Vittal Prabhu. "Complexity metrics for manufacturi...	<1%
14	Student papers	University of Hertfordshire on 2019-08-31	<1%
15	Internet	pmc.ncbi.nlm.nih.gov	<1%
16	Internet	www.coursehero.com	<1%
17	Internet	www.mdpi.com	<1%
18	Publication	Poonam Nandal, Tapas Kumar, Meeta Singh, Mamta Dahiya. "Progressive Compu...	<1%
19	Publication	Qiong Gu, Li Zhu, Zhihua Cai. "Chapter 53 Evaluation Measures of the Classificatio...	<1%
20	Student papers	Strathmore University (Main Account) on 2026-06-08	<1%
21	Student papers	Higher Education Commission Pakistan on 2026-01-12	<1%
22	Publication	Sukhpreet Kaur, Amanpreet Kaur, Manish Kumar. "Recent Advances in Computati...	<1%
23	Student papers	Taylor's Education Group on 2024-06-08	<1%
24	Internet	doctorpenguin.com	<1%

25	Internet	ijsrset.com	<1%
26	Internet	lawlifted.com	<1%
27	Publication	Ilkka Juuso, Ilpo Pöyhönen. "Medical-Grade Software Development - How to Build...	<1%
28	Publication	Jianzhong Zhou, Yahui Shan, Jie Liu, Yanhe Xu, Yang Zheng. "Degradation Tenden...	<1%
29	Student papers	Kazakh-British Technical University on 2026-05-29	<1%
30	Student papers	University of Greenwich on 2026-04-15	<1%
31	Internet	escholarship.org	<1%
32	Publication	XiaoLi Zhang. "A grid-based ACO algorithm for parameters optimization in suppo...	<1%
33	Internet	oaji.net	<1%
34	Internet	openaccess.sgul.ac.uk	<1%
35	Internet	wes.copernicus.org	<1%
36	Student papers	University of East London on 2026-05-05	<1%
37	Internet	coek.info	<1%
38	Internet	indiafreenotes.com	<1%

39	Internet	med.stanford.edu	<1%
40	Internet	osuva.uwasa.fi	<1%
41	Internet	oulurepo oulu.fi	<1%
42	Internet	projekter.aau.dk	<1%
43	Publication	Communications in Computer and Information Science, 2014.	<1%
44	Student papers	University Metropolitan Tirana on 2026-06-22	<1%
45	Publication	"Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries", Spr...	<1%
46	Student papers	City University of Seattle on 2025-06-11	<1%
47	Student papers	Liverpool John Moores University on 2026-03-30	<1%
48	Student papers	Universiti Sains Islam Malaysia on 2026-06-26	<1%
49	Student papers	University of Melbourne on 2021-06-12	<1%
50	Internet	acikerisim.karabuk.edu.tr:8080	<1%
51	Internet	campus.datacamp.com	<1%
52	Internet	irjiet.com	<1%

53	Internet	www.cellbiotech.com	<1%
54	Student papers	Higher Education Commission Pakistan on 2023-10-19	<1%
55	Publication	Leon, Javier. "Forecasting Imported Fruit Prices in the United States Using Neural..."	<1%
56	Publication	Liu, Xuanzhang. "Incentive Mechanism Design in Blockchain-Based Federated Lea..."	<1%
57	Student papers	M S Ramaiah University of Applied Sciences on 2026-05-21	<1%
58	Student papers	National Institute of Technology, Warangal, Telangana, India on 2026-04-27	<1%
59	Student papers	University of Hertfordshire on 2024-11-11	<1%
60	Student papers	University of Hertfordshire on 2025-06-24	<1%
61	Internet	docplayer.net	<1%
62	Internet	iopscience.iop.org	<1%
63	Internet	openscience.ink	<1%
64	Internet	proceedings.icisna.org	<1%
65	Internet	riunet.upv.es	<1%
66	Internet	www.mphasis.com	<1%

67	Student papers	Aligarh Muslim University, Aligarh on 2026-05-09	<1%
68	Publication	Amie Taal. "The GDPR Challenge - Privacy, Technology, and Compliance in an Age ...	<1%
69	Student papers	Coventry University on 2019-03-05	<1%
70	Student papers	ICTS on 2026-06-20	<1%
71	Student papers	TU Delft on 2026-07-06	<1%
72	Student papers	Thesis 2026 on 2026-06-05	<1%
73	Student papers	University of Hertfordshire on 2025-09-04	<1%
74	Student papers	Vels Institute of Science, Technology & Advanced Studies (VISTAS) on 2026-06-27	<1%
75	Internet	fg-sre.gi.de	<1%
76	Internet	learn.lajevardi.id.ir	<1%
77	Internet	pure.uva.nl	<1%
78	Internet	raw.githubusercontent.com	<1%
79	Internet	studenttheses.uu.nl	<1%
80	Internet	www.dafx.de	<1%

81	Internet	www.gcu.edu	<1%
82	Internet	www.ijiemr.org	<1%
83	Internet	www.osti.gov	<1%
84	Publication	"Computer Vision – ECCV 2016 Workshops", Springer Nature, 2016	<1%
85	Publication	"Software Project Management for Distributed Computing", Springer Science and...	<1%
86	Publication	A. Punitha, S. Syedakbar, S. Jeyasudha. "Intelligent and Sustainable Systems: AI, G...	<1%
87	Student papers	DSI HEC on 2026-06-14	<1%
88	Publication	Debbie Hahs-Vaughn, Richard Lomax. "An Introduction to Statistical Concepts - T...	<1%
89	Publication	Dinesh Goyal, Payal Bansal, Ritam Dutta, Madhav Sharma. "Intelligent Systems U...	<1%
90	Student papers	Dublin Business School on 2026-01-07	<1%
91	Student papers	Georgia Institute of Technology Main Campus on 2020-09-21	<1%
92	Publication	Haixiang Zang, Ling Liu, Li Sun, Lilin Cheng, Zhinong Wei, Guoqiang Sun. "Short-t...	<1%
93	Student papers	Khalifa University of Science Technology and Research on 2025-05-02	<1%
94	Student papers	Leeds Beckett University on 2026-04-30	<1%

95	Student papers	Multimedia University on 2026-06-24	<1%
96	Student papers	National College of Ireland on 2024-08-12	<1%
97	Student papers	Nottingham Trent University on 2020-09-03	<1%
98	Student papers	Robert Kennedy College on 2026-07-04	<1%
99	Publication	Sushant Kumar Pandey, Ravi Bhushan Mishra, Anil Kumar Tripathi. "BPDET: An ef...	<1%
100	Student papers	TU Delft on 2026-07-06	<1%
101	Student papers	Universiti Teknikal Malaysia Melaka - Library on 2026-06-19	<1%
102	Student papers	University of Edinburgh on 2024-08-15	<1%
103	Student papers	University of New South Wales on 2023-04-28	<1%
104	Internet	dergipark.org.tr	<1%
105	Internet	ebin.pub	<1%
106	Internet	ijsra.net	<1%
107	Internet	thesis.unipd.it	<1%
108	Internet	www.hindawi.com	<1%

109

Internet

www.llamaindex.ai

<1%

110

Internet

www.mechanics-industry.org

<1%

A HYBRID CNN-LSTM FRAMEWORK FOR SOFTWARE DEFECT PREDICTION IN SAFETY-CRITICAL HEALTHCARE SYSTEMS WITH EXPLAINABILITY AND REGULATORY COMPLIANCE

Samriddhi Paul

ABSTRACT

In safety-critical medical settings, software must work correctly, follow rules, yet protect patients above all. Spotting faulty code early helps lower risks while cutting testing expenses. Instead of Support Vector Machines or Random Forest, methods like Naïve Bayes once set early standards for predicting bugs in programs. Yet these older techniques usually miss curved relationships and time-based changes within health data logs. Lately, deeper networks - including Convolutional Neural Networks - show stronger results by uncovering layered traits across defect flows. Patterns that shift over time get captured better using Long Short-Term Memory units inside prediction setups. When reversed sequences join them, via Bidirectional LSTM variants, detection improves further still. Combining both types into merged CNN-LSTM frameworks reveals hidden links regular tools overlook entirely. Hybrid setups, especially those tracking time patterns, tend to beat older methods when it comes to precision, catching true cases, and overall scoring - just as importantly, they miss fewer actual problems, which matters greatly in patient care settings. Starting from this point, the current study applies a combined CNN-LSTM structure using data drawn from NASA PROMISE archives alongside real-world medical records. Instead of ignoring uneven class sizes, SMOTE adjusts the input flow, helping balance representation across groups; results land at 0.94 for correctness and detection ability, with average deviation measured at 0.12. Moving past numbers alone, the system embeds audit-ready trace logs built around IEC 62304 rules, pulling research ideas into usable product territory. Not only does the project survey existing knowledge carefully, it delivers working code, showing how neural networks might improve bug forecasting within tightly governed health tech environments.

Chapter 1

Introduction

1.1 Overview

Most health systems now rely heavily on software, a shift that has accelerated over twenty years. Systems like electronic records (HER) or image storage tools run daily operations. Devices such as heart monitors, insulin delivery units, IV controls, and robotic surgery aids depend on code. When these programs fail, consequences go beyond downtime - harm to patients becomes possible. Mistakes may stall hospital routines while inviting legal scrutiny. Safety rules differ from typical business software demands. One global norm, IEC 62304, sets clear stages for developing medical software. Regulators in the United States outline similar expectations through FDA guidelines for software classified as medical devices. Each framework requires structured planning, hazard assessment, proof of testing, and ongoing monitoring after release.

With these challenges in mind, predicting software defects becomes a practical way to improve quality by spotting risky parts of code during early stages, like design or programming. Instead of waiting until late testing rounds - methods known for high expense and slow results - this approach uses past data such as version history, changes in code structure, and archived project records to anticipate trouble spots. Spotting issues sooner allows teams to direct reviews where they matter most, adjust test plans with precision, and manage risks without wasting resources - a benefit especially clear in medical software governed by strict rules.

Older methods in software defect prediction used stats tools like logistic regression alongside older classification techniques - **Support Vector Machines, Random Forests, Naive Bayes, and Decision Trees**. Cyclomatic complexity, lines of code, Halstead volume, inheritance depth, and coupling served as key inputs for these systems. Though results were acceptable on standard sets like NASA PROMISE, weaknesses appear when dealing with complex, layered, time-based traits common in medical software projects. Sequential changes over time, shifting edit behaviors, mixed clinical data formats, plus rare fault cases, make classic setups struggle. Because defects occur infrequently, models often fail to detect real patterns across evolving codebases.

One way to tackle several existing challenges involves deep learning, which relies on layered neural structures to learn useful representations and track dependencies over time. Instead of handling features manually, convolutional neural networks automatically detect localized patterns in data such as code measurements or visual indicators of defects. When it comes to analyzing ordered information - like changes across software versions - models like **long short-term memory units and gated recurrent units** show strong capability due to their design for sequence tasks. By moving in two directions along a sequence, bidirectional variants capture context from past and future states simultaneously. Combining convolution-based feature detection

with recurrence-based timing analysis has led to better results than either method alone, according to various experiments reported in recent research [1][3][4].

The work adds to knowledge in three ways - through data, technique, and policy impact. Because it uses deep learning, progress happens in healthcare secure data processing; at the same time, hurdles around transparency and rules are tackled for real-world clinical tools. This structure uses SMOTE to correct uneven class distribution. Its results include accuracy at 0.94, AUC reaching 0.94, recall matching that level, alongside a mean absolute error of just 0.12 - surpassing earlier approaches. Performance gains stand clear when compared to referenced benchmarks.

Figure 1.1: Healthcare Software Ecosystem and the Role of Defect Prediction

Healthcare Software Domain	Quality Assurance Pathway	Outcome
EHR Systems Diagnostic Imaging Patient Monitoring Medical Devices (IEC 62304 / FDA SaMD)	Traditional Testing (Unit, Integration, System) + Software Defect Prediction (ML / Deep Learning)	Early defect detection Reduced testing cost Regulatory compliance Patient safety assurance
Key Metrics: LOC, Cyclomatic Complexity, Halstead Volume, Code Churn	DL Models: CNN, LSTM, GRU, BiLSTM, Hybrid CNN-LSTM (proposed)	AUC: 0.94 Recall: 0.94 MAE: 0.12 F1: 0.93

1.2 Motivation

Among today's healthcare software concerns, safety demands stand out because lives depend on reliable performance. Traditional methods fall short when applied to complex medical systems now in use. Deep learning could help, yet remains under-utilised in settings where regulations shape development. A shift toward transparent models is underway, driven by both clinical needs and oversight requirements.

Not every widely used method fits every domain equally well - take standard machine learning in health-related software prediction. These approaches often stumble when facing patterns typical in medical software data. Accuracy may look strong on paper, but catching actual defects? That measure lags behind where it needs to be, especially in contexts where missing an error can have serious consequences. Mislabeling a risky module as safe carries heavier risks than flagging a harmless one, yet most models do not account for such differences in outcome severity. Another hurdle emerges during preparation - older techniques depend heavily on hand-crafted features, lacking built-in capacity to track changes across time, like shifts in source code, update logs, or revised clinical guidelines.

One reason stands out when working under strict oversight: understanding how decisions emerge matters deeply. Notably, complex networks combining convolutional and sequential layers resist clear inspection by those tasked with safety checks. It follows then that regulators demand transparency - what inputs shaped outcomes, how rules form inside code, where results link to patient harm levels must all appear in records. Seen differently, tools like SHAP or LIME open hidden pathways; without

them, performance gains mean little if trust cannot grow alongside. Attention maps help too, revealing focus points much like human experts would note. Thus emerges a guiding idea - that clarity belongs at the core, built in before deployment, not patched later.

What drives this study forward ties closely to wider access to data, not just general benchmarks but also those tied to medical systems. From NASA PROMISE come several sets - AR1 through JM1, plus KC2, KC3, and MC1 - each offering uniform measurements alongside clear defect indicators. Alongside these sit materials focused on health: records pulled from PCORnet's electronic archives, image banks used in diagnosis, even logs tracking changes in mobile medical apps as in Fig 1.2 . With tools like TensorFlow and Keras now more refined, progress in software defect prediction within care environments becomes possible - not because it was time, but because conditions finally align.

Deep learning shows strong results elsewhere - matching human skill in reading medical scans, processing clinical text, tracking genetic data - yet its role in predicting flaws in health software stays limited. Scattered efforts appear across papers: convolutional networks spot anomalies from visual outputs [2]; sequence models forecast bugs after code changes in mobile systems [3]; blended designs run tests on heart-related records [1]. Still, none bring such methods together under one structure built for real-world rules.

Figure 1.2: Four Pillars of Research Motivation

Pillar 1 Safety-Criticality	Pillar 2 ML Limitations	Pillar 3 DL Potential Gap	Pillar 4 Explainability Need
IEC 62304 Class A/B/C FDA SaMD Guidance Patient harm from defects Zero-tolerance for false-negatives	Class imbalance bias Limited temporal modelling Manual feature engineering Poor recall in minority class	No unified DL+healthcare SDP Disparate architectures No compliance integration No SMOTE+CNN-LSTM pipeline	Black-box DL models FDA audit trail requirements SHAP / LIME / Attention needed Lifecycle traceability mandated

1.3 Problem Statement

In healthcare environments, software is used in safety-critical applications where software errors may endanger patient safety, clinical decisions, and compliance with regulation. With the increase in the complexity of medical devices, **Electronic Health Record Systems, and clinical decision support systems**, early detection of defective software components has become crucial as per regulations such as IEC 62304 and FDA guidelines for software as a medical device.

The conventional machine learning methods for predicting software defects such as SVMs, Random Forests, and Naïve Bayes have proved to perform well on traditional benchmark datasets, but none of these have been able to deal with the specific challenges of software defect prediction within the healthcare domain. These models have failed to identify the non-linear relations between features of data and do not understand long-range dependencies present in the data history of software versions. Moreover, they cannot handle skewed data where defective software components make

up only 5-20 percent of the whole dataset. In addition to that, current approaches ignore the critical issue of asymmetrical misclassification costs which is crucial in the case of healthcare software defects.

1.4 Proposed Solution

Should the issue outlined in Section 1.3 require addressing, one approach involves a unified deep learning structure intended for predicting defects in healthcare software. Built into its design are four interlinked parts working together without separation. First comes a process adjusting for unequal data distribution across classes. After that appears a forecasting mechanism combining convolutional and long short-term memory networks. This element connects directly to another layer revealing how predictions form internally. Lastly, included is a feature generating records necessary for adherence to legal standards. Descriptions follow for each piece just mentioned. Then follows an overview showing how everything operates as one sequence.

1.4.1 Class Imbalance Resolution by SMOTE

Class imbalances are solved at the level of the data pre-processing phase using the Synthetic Minority Over-sampling Technique (SMOTE). While random oversampling simply creates copies of existing defective instances, SMOTE creates synthetic defective instances based on interpolation of existing minority-class samples in the feature space. Specifically, for an instance x_i of the minority class and its k -nearest neighbour x_j , SMOTE creates synthetic instances as follows: $x_{\text{new}} = x_i + \lambda \cdot (x_j - x_i)$, where λ is a randomly chosen scalar belonging to $[0, 1]$. SMOTE was implemented as part of the training phase for all training folds in 10-fold cross-validation.

1.4.2 Hybrid CNN-LSTM Model

Our predictive model consists of a hybrid architecture involving two stages. First, a set of 1D CNN layers and an optional max pooling layer process the input metric vectors by learning filters along the feature axis, and second, LSTM layers process the results. During the application of 1D CNN layers with ReLU activation function, input vectors x are filtered using the following equation :

$$h_i = \sum_{m=0}^{M-1} w_m \cdot x_{i+m} + b \quad (1.1)$$

where b is a bias.

During the next step, the generated feature sequences are fed through LSTM layers, which aim to find temporal relationships both across features ordering and commit-level ordering, if possible. The gating structure of the LSTM cell, with its forget gate f_t , input gate i_t , output gate o_t , and cell state C_t , is formulated by the following equations:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t; \quad (1.2)$$

$$h_{\square} = o_{\square} \odot \tanh(C_{\square}) \quad (1.3)$$

with $\odot$ being an element-wise product operator. Thus, the neural network will be able to remember/reject relevant information for a long period without suffering from a vanishing gradient problem.

Training of the CNN-LSTM model takes place with Adam optimisation technique (with learning rate equals to 0.001) by minimising a **cross-entropy loss function** with **binary cross-entropy loss**, as well as with adding dropout layers for avoiding the overfitting issue. The sigmoid activation function at the final layer generates the defect probability value of $P(y=1|x)$. The model validation was performed using 10-fold cross-validation on various NASA PROMISE and healthcare datasets.

1.4.3 Explainability Layer

In order to meet the requirement of the regulators for transparency, the framework includes an explanation mechanism based on SHAP (SHapley Additive exPlanations) values. For each prediction generated by the model, SHAP provides the marginal contribution of the feature to the final output of the prediction, allowing clinical engineers and quality assurance auditors to determine which software metrics (cyclomatic complexity, number of lines of code, or code churn) were responsible for the defect identification. An attention mechanism is suggested for the LSTM module as well, resulting in saliency maps on the time axis showing the effect of certain time steps (revisions or commits) on the prediction.

1.4.4 Regulatory Compliance Documentation Module

The framework includes a documentation and logging module for creating documentation artefacts compliant with IEC 62304 for each defect prediction. Documentation artefacts generated from the logging and documentation module include: (i) dataset provenance (source, preprocessing, SMOTE parameter settings); (ii) model training logs (hyperparameters, fold-wise validation results, and convergence curves); (iii) prediction results for each module along with SHAP values and risk classification; and (iv) post-market performance dashboards. The documentation and logging module guarantees that all predictions are traceable to clinical requirements and FDA Software as a Medical Device life cycle documentation.

The proposed model was developed using Python 3.9 with TensorFlow 2.x and Keras along with scikit-learn as a library for implementing classical models, while the data used was preprocessed by SMOTE. All models were trained and tested using the same 10-fold cross-validation technique. It has been found that the CNN-LSTM approach yields the best performance in terms of all critical evaluation measures, such as Accuracy 0.94, Precision 0.91, Recall 0.94, F1-score 0.93, AUC 0.94, and MAE 0.12.

Table 3 : Proposed system workflow

STAGE	COMPONENT	TECHNIQUE / TOOL	OUTPUT
1. Data Ingestion	NASA PROMISE + Healthcare EHR + Imaging datasets	Python / Pandas Feature extraction (LOC, CC, Halstead, Churn)	Raw feature matrix $D = \{(x_i, y_i)\}$
2. Preprocessing	Class imbalance correction Normalisation, train/test split	SMOTE (k-NN interpolation) Min-Max / Z-score normalisation 10-fold cross-validation	Balanced training sets Standardised feature vectors
3. Model Training	Hybrid CNN-LSTM architecture Ablation baselines: RF, SVM, NB, CNN, LSTM, GRU, BiLSTM	TensorFlow / Keras Adam optimiser ($\text{lr}=0.001$) Dropout, BatchNorm Binary cross-entropy loss	Trained model weights Per-fold metrics (Acc, AUC, F1, Recall, MAE)
4. Evaluation & Explainability	Performance benchmarking Feature attribution Attention visualisation	SHAP values LIME LSTM Attention maps ROC / PR curves	Defect probability per module Top-k influential features Model comparison tables
5. Regulatory Deployment	IEC 62304 artefact generation FDA SaMD documentation Privacy compliance	Audit log generation Federated learning / Anonymisation Post-market monitoring dashboard	Compliance-ready SDP system Traceable prediction records Continuous performance monitoring

1.5 Main contributions of the thesis

Among the outcomes presented here, six stand apart due to their novelty within software engineering, deep learning, and health data systems. Taken together, these elements push forward not only conceptual insight but also real-world function in ensuring reliability of medical software. Despite differences in approach, each piece aligns under a shared aim: improving how such systems are validated. Where prior work leaves gaps, new pathways emerge - shaped by evidence, guided by design. In sum, progress appears both measurable and meaningful across multiple fronts.

Deep Learning in Healthcare Software Development Process A Systematic Review Among initial advances stands a methodical comparison grounded in empirical data, examining various deep learning frameworks - CNN, LSTM, GRU, BiLSTM, and combined CNN-LSTM or CNN-GRU - as they apply to spotting flaws within medical software. From more than two dozen peer-reviewed works emerge insights drawn from varied test environments, assessment standards, and real-world implementations: electronic health records, image diagnosis tools, mobile health apps on Android, along with standardized NASA PROMISE references. What results is an organized assembly of understanding tailored specifically to medicine-linked coding risks - a resource absent before in such focused shape. Notably recorded throughout the study: integrated neural networks surpass conventional machine learning methods as well as standalone deep structures by every core metric observed, where CNN-LSTM setups yield average AUC scores near 0.91, recall averaging 0.84 when tested against standard reference sets.

Chapter 2

Technical Background

Found within this section are the structured ideas and technical elements supporting a deep learning approach to predicting flaws in healthcare-related software. While defect forecasting takes shape through precise definitions, attention also turns to how strictly governed such medical technologies must be. Instead of skipping ahead, traditional methods of pattern recognition receive mention before deeper computational models appear. Emerging after them are specific network types - **Convolutional Neural Networks, followed by Long Short-Term Memory** units, their variant called GRU, and two-directional versions known as BiLSTM. A mix combining CNNs with sequence learners plays a key role later on. Prior steps involving data adjustments, handling unequal outcome groups, measuring performance fairly, and aligning with oversight standards come into view too.

2.1 Software Defect Prediction: Problem Formulation

Early detection of problematic code segments defines Software Defect Prediction (SDP). Though embedded within the broader software development lifecycle (SDLC), its focus remains narrow - spotting fault-prone components prior to testing phases. Because fixing flaws post-deployment demands greater effort and cost, catching them earlier proves less resource-intensive. In high-stakes fields like medical systems, overlooked errors risk human safety alongside legal consequences [1]. When predictions guide inspection efforts, teams distribute their validation workloads with better precision. SDP becomes a supervised binary classification problem. Consider the labelled dataset extracted from a repository:

$$D = \{(x_i, y_i)\} \quad \text{for } i = 1, 2, \dots, n \quad (2.1)$$

where $x_i \in \mathbb{R}^d$ is the **d-dimensional feature vector** explaining the i -th software module and $y_i \in \{0, 1\}$ is the truth label depicting whether the module was found to contain at least one defect ($y_i = 1$) or was non-defective ($y_i = 0$). **The objective is to learn a parametric predictive function:**

$$f_\theta : \mathbb{R}^d \rightarrow [0, 1], \quad f_\theta(x_i) = P(y_i = 1 \mid x_i; \theta) \quad (2.2)$$

where θ represents the trainable parameters of the model.

A module is said to be as defect-prone when $f_\theta(x_i)$ becomes greater than a decision threshold $\tau \in (0,1)$, which is adjusted to balance precision and recall based on the operational cost of the deployment procedure .

One way to build feature vectors x_i is through diverse origin points. Rather than relying on single sources, data often stems from static code traits - cyclomatic complexity, for instance, or lines of code (LOC) , Halstead volume - all gathered efficiently via static analysis tools. Development timelines shape another layer: past

error corrections, tenure of coders involved, frequency of edits measured as inserted or removed lines, alongside how many individuals worked on the file. When systems belong to medical domains, extra elements enter - safety labels required under standards like IEC 62304 appear, along with documented paths linking specs to actual code components [5].

One key issue within SDP involves uneven class distribution: defective parts often make up just 10–25% of actual software systems, according to Table 2.1. Because of this biasness, models tend to favor predicting non-defective units - yielding strong overall correctness yet weak detection rates i.e. high accuracy but poor recall. In contexts like medical software, failing to identify flawed components carries serious risk, given possible harm from undetected errors. Such risks elevate imbalance correction beyond algorithmic tuning into compliance territory. Hence, oversampling methods like SMOTE appear in the proposed framework – which also eradicates modelling concerns and as well as regulatory issues.

Temporal shifts introduce further complexity: when code metrics from an earlier release differ statistically from those in future cycles because of redesigns, personnel turnover, or altered design choices. Since these gradual changes weaken model performance on past records, methods like LSTMs become relevant - able to detect patterns that evolve through software updates [3].

A single commit serves as the unit of analysis in Just-in-Time Software Defect Prediction, differing from version-level approaches. Instead of modules, each modification receives a feature vector built from changed metadata like altered files, inserted or removed lines, timestamp, and contributor history and label determines whether that specific update brought any defect. Real-time alerts during development workflows benefit from this method. Feedback reaches programmers immediately after they submit changes. The timing aligns closely with modern delivery cycles. Detection happens before broader integration occurs. Precision improves when focus shifts to granular updates instead of full releases.

Table 2.1 Representative dataset showing class imbalance characteristics

Dataset	Features	Instances	Defect Rate(%)	Domain	Source
CM1	22	505	10%	NASA	PROMISE
KC1	26	2110	15%	NASA	PROMISE
JM1	21	10884	21%	NASA	PROMISE
MC1	38	9467	12%	NASA	PROMISE
EHR System	35+	5000+	8-20%	Healthcare	Clinical

2.2 Healthcare Software Systems and the Safety-Critical Context

Software used in healthcare stands unique within engineering due to its intense requirements. Business tools or apps for personal use differ sharply - here, flaws can lead straight to bodily injury. Systems such as diagnostic scanners rely on code where mistakes carry weight. Electronic health records (EHR) must function without fault because lives depend on accuracy.. Ventilators operate using embedded logic needing constant reliability. When these fail, consequences emerge quickly. A hidden bug might delay diagnosis or alter dosage amounts. Support tools guiding medical choices also belong to this high-stakes group. Any silent issue inside them risks incorrect therapy decisions. Harm follows when precision slips - even once [7].

Recognition of healthcare software importance appears in both global and local regulatory systems. Under IEC 62304, international rules shape how medical device software progresses through its life stages. Depending on safety-critical levels tied to malfunction, programs must fit within one of three safety categories:

Class A: Not injurious to health.

Class B: Possibility of Non-serious injury.

Class C: Possibility of death or serious injury.

Table 2.2 : Risk level and their classes

Layer	Example Components	Risk Level
Device Firmware / Embedded SW	Pump controllers, biosensor drivers, FPGA logic	Critical (Class C — IEC 62304)
Clinical Decision Support	Diagnosis engines, alert generation, drug interaction checks	Serious (Class B)
Data Management / EHR	Patient record storage, HL7 FHIR interfaces, audit logging	Moderate (Class B)
User Interface / Reporting	Dashboards, mobile apps, clinician portals	Low-Moderate (Class A-B)

When software falls under Class B or Class C, adherence to IEC 62304 requires recorded risk assessments, traceability between requirements and implementation, structured testing methods, along with post- monitoring after release. Such conditions influence the structure of the SDP challenge through inclusion of compliance-related data within features while demanding transparency so model outputs can be justified and explainable through imposing explainability restriction on predictive models .

Rules about privacy add limits to how health-related software handles information. In America, HIPAA sets boundaries. Europe enforces similar rules through GDPR, controlling storage, processing, and transfer of personal medical details. If system data comes from electronic records or clinic activity logs, identifiers must first be removed prior to any use in algorithm development. One method gaining attention involves training models locally across multiple hospitals - keeping raw files separate while still

improving performance collectively. This approach avoids pooling sensitive material at a single location.

Modern healthcare software involves vast amounts of code, varied data inputs, and continuous operation demands. Unlike standard NASA reference systems, its performance indicators are far more complex, often twisting in unpredictable patterns. Development records within electronic health record systems stretch back many years. Numerous programmers participate worldwide, altering features over long periods. Because such information is layered, irregular, and unfolds step by step, it suits advanced neural models well. Yet at the same time, oversight rules require clarity - something these intricate networks traditionally lack. High depth meets strict accountability. Few fields expose this tension so clearly.

2.3 Software metrics and Feature Engineering

The strength an SDP model shows in forecasting stems mainly from how well its features describe software modules. Studies of software measurement theory has come much of what shapes today's Feature Engineering for defect prediction. When applied to health-related systems, these models also require regulatory compliances and domain-specific patterns.

2.3.1 Size and Complexity Measures

Size metrics quantify the scale of a software module. Lines of Code (LOC) relate closely to defect density. Instead of raw lines, some prefer Non-Commented Source Statements (NCSS), since these ignore whitespaces and comments, working across different programming languages. When dealing with object-oriented designs, the count of methods offers insight into functional segmentation within components. Similarly, tracking the number of classes reflects how finely a system divides its structure into distinct types.

Among known metrics, McCabe's Cyclomatic Complexity stands out due to frequent examination. It counts the number of linearly independent paths through a program's control flow graph:

$$CC = E - N + 2P \quad (2.3)$$

Where E is for number of edge, N for node count, P for connected component count within the control flow graph. When CC exceeds 10, complexity rises - such modules often carry more defects. Deeply layered conditionals, common in healthcare firmware during device state handling, typically lead to elevated CC levels.

Halstead metrics calculate software complexity by counting operators and operands in source code. With η_1 indicating unique operators, η_2 standing for distinct operands, N_1 tracking total operator uses, N_2 recording operand appearances - these values form the core set of Halstead measures.

$$\text{Program Volume: } V = (N_1 + N_2) \times \log_2(\eta_1 + \eta_2) \quad (2.4)$$

$$\text{Difficulty: } D = (\eta_1/2) \times (N_2/\eta_2) \quad (2.5)$$

$$\text{Effort: } E_H = D \times V \quad (2.6)$$

Understanding how much mental efforts a module demands helps predict error risks when changes are made. When EHR systems process intricate medical term mappings, the cognitive load often rises sharply. Complexity in these routines typically aligns with elevated Halstead metrics.

2.3.2 Object-Oriented Metrics (CK Suite)

In 1994, Chidamber and Kemerer introduced a set of measurements tailored for object-oriented designs. This collection includes six specific indicators: WMC, DIT, NOC, CBO, RFC, LCOM - each addressing distinct aspects of structure. Rather than summarising everything at once, they work together to reflect how complex and interconnected a class system is. When CBO rises, connections between classes grow stronger; because of that, issues may spread across parts of the system unexpectedly. Such ripple effects matter greatly in medical decision tools, since modifications in one area might trigger unanticipated outcomes elsewhere.

Process Metrics and Code Churn

Change patterns in a module matter more than fixed design when tracking how it evolves. Not every edit stays - code churn counts all inserted, removed, or altered lines through revisions and often signals where flaws may appear. When files shift often and by wide margins, missteps grow more probable simply due to repeated intervention. Who changed what also plays a role - one measure tracks how many different authors touched a given file, hinting at coordination complexity. The age of the latest update carries weight; older edits might suggest dormancy or stability. A record of prior corrections adds context: files patched repeatedly tend to attract further issues. Even the tone found in submission notes has surfaced as relevant - not emotional flair, but clarity or urgency markers within logs. Each of these indicators contributes uniquely to forecasting trouble spots based on real-world data analysis [2].

Feature Selection and Dimensionality Reduction

Healthcare SDP raw feature vectors might hold thirty to fifty metrics - or even more - where several often bring little new insight. Though redundancy lurks within these inputs, trimming them sharpens focus; it also lessens noise during analysis. Instead of keeping everything, selecting key traits shrinks data size while supporting clearer patterns. One path uses filters: Information Gain steps forward here, measuring how tightly each metric links to outcomes through shared signals. Correlation-Based Feature Selection follows a similar logic but weighs group harmony over isolated strength. Rather than relying on statistics alone, some strategies loop the model into the process itself. Recursive Feature Elimination removes one weak link at a time, guided by performance shifts after each cut. This kind of wrapper approach adapts with every round, shaped entirely by real-world impact. Another route skips feature removal altogether, choosing transformation instead. Principal Component Analysis reshapes space without supervision, aligning axes along directions where data spreads widest. Its

output preserves variation, packing dense structure into fewer dimensions. Even when features vanish, much meaning stays - if projected wisely.

Automatic feature extraction in deep learning-driven software defect prediction handles part of what used to require manual selection, thanks to trained filters and focus layers. Still, cutting out nearly identical measurements before training - those with correlation above 0.95 - helps cut repetition while speeding up model stabilization.

2.3 Machine Learning Methods for Predicting Software Defects

Prior to deep learning's rise, traditional statistical techniques shaped how software flaws were forecasted. Though newer systems now exist, these older tools still serve as benchmarks for performance comparison; their clarity and modest processing needs persist as strengths. In contexts where computing resources are limited, such models often perform reliably without excessive demand. Healthcare-related defect prediction draws notably from these established strategies. A review of key conventional methods follows, focusing on those most applicable within medical software settings.

2.3.1 Logistic Regression

Among statistical methods, **logistic regression** serves as a primary tool when outcomes split into two categories. Starting from feature inputs, it estimates probabilities through weighted sums transformed by a sigmoid function. Rather than outputting raw scores, the model frames results in terms of likelihood ratios. Through maximum likelihood estimation, parameters adjust to fit observed data patterns. Instead of assuming normality, it relies on binomial distribution assumptions. The method gains utility especially when interpretability matters alongside prediction

Predicted probability of y equaling one given x follows a transformation involving coefficients and input values. This expression uses **the sigmoid function applied to a linear combination**. The result equals one divided by one plus an exponential term with negative sign. Equation 2.7 shows how output relates to weighted sum through smooth mapping. Parameters interact directly with features inside the parentheses before processing:

$$P(y = 1 | x) = \sigma(\beta_0 + \sum_i \beta_i x_i) = 1 / (1 + e^{-(\beta_0 + \sum_i \beta_i x_i)}) \quad (2.7)$$

In this context, β stands for the weight vector while $\sigma(\cdot)$ denotes the logistic sigmoid operation.

Estimation occurs through log-likelihood maximization across training samples, often via gradient-based updates. Despite relying on a linear separation rule - restricting modeling of intricate variable relationships - it yields transparent coefficients indicating how much each software measure contributes. Performance on NASA PROMISE benchmarks shows average AUCs between 0.74 and 0.82; such results mark solid performance yet remain below those of combined models and neural networks, which reach 0.86 to 0.91 [1].

2.3.2 Naive Bayes

Given a class label, naive Bayes models presume feature independence when applying Bayes' rule. Calculation of posterior probabilities follows accordingly. $P(y \text{ equals } 1 \text{ given } x)$ is proportional to $P(y \text{ equals } 1) \times \prod_i P(x_i \text{ given } y \text{ equals } 1)$.

Even though the assumption of conditional independence seems too simplistic - especially given that software metrics often correlate - Naive Bayes still manages strong recall on skewed data due to minimal need for training examples. Because defect-laden cases are few in medical software defect prediction, its ability to function well with limited instances becomes useful. Tadapaneni and colleagues (2022) [11] tested this model on the PROMISE collection, finding roughly 70% correctness yet notable responsiveness to rare categories if priors were adjusted carefully. Since it reveals how inputs affect outcomes via probability scores per class, regulators can review decisions more transparently.

2.3.3 Support Vector Machines (SVM)

Geometric margin maximisation guides Support Vector Machines when identifying the optimal hyperplane across classes within feature space. For soft-margin variants, this goal translates into an objective framed by constraints - presented formally below

$$f(x) = \sum_i \alpha_i y_i \kappa(x_i, x) + \quad (2.7)$$

The Radial Basis Function (RBF) kernel, which has the following form: $\kappa(x_i, x) = \exp(-\gamma \|x_i - x\|^2)$, is widely used in SDP due to its ability to model non-linear interactions between features without having to specify any parameters of the kernel. SVM models tend to have low rates of false positives on balanced datasets. Nevertheless, they are very expensive to compute for large codebases and are kernel-sensitive.

2.3.4 Random Forest (RF)

A random forest is a collection of independently trained decision trees where each tree learns from a randomly sampled version of the training dataset using a random subset of features for making splits:

$$\hat{y} = \text{mode} \{ h_t(x) : t = 1, \dots, T \} \quad (2.8)$$

where $h_t(x)$ is the t -th tree prediction. The performance of RF on software metric data is due to its ability to perform well in high-dimensional, noisy space, implicitly performs feature selection using Gini impurity importance, and is less likely to overfit compared to a single decision tree. The RF model scores an average ROC-AUC of ≈ 0.86 and an average recall of ≈ 0.78 on the NASA PROMISE datasets.

2.4 Deep Learning Fundamentals for Sequential Data

Learning deeply involves using layered networks that process information through successive stages. From one stage to the next, simple features emerge first - such as isolated numerical limits - followed by more intricate structures later on. At middle levels, relationships form - not just single values but how they interact, like complexity tied to dependencies. Higher up, the system identifies broad concepts, such as structural traits linked to software flaws. Because each level builds upon the last, human-designed inputs become less necessary. Instead, the model works directly with unprocessed data elements - code symbols, version notes, sequences of measurements - without needing prior transformation. Such progression across depth allows automatic refinement of meaning without external guidance.

2.4.1 The Feedforward Neural Network

A single step in deep learning begins with a structure known as the fully connected feedforward neural network - often labeled an MLP. Transformation happens across L hidden stages, where each stage shifts the data forward without loops or skips

$$h^{(l)} = \sigma(W^{(l)}h^{(l-1)} + b^{(l)}), l = 1, 2, \dots, L \quad (2.13)$$

Starting with $h^{(0)}$ set equal to x , the initial input vector, each layer computes a transformation using its own weight matrix $W^{(l)}$ and bias vector $b^{(l)}$. Instead of simple summation, affine operations precede application of $\sigma(\cdot)$, an element-level nonlinear function. For final predictions, the last layer uses a specific form of $\sigma(z)$ - the sigmoid - defined as 1 divided by (1 plus e raised to negative z). This yields values in the range (0,1), suitable for estimating likelihoods in two-class tasks. Minimisation of binary cross-entropy governs parameter updates during training. Each component adjusts gradually, driven by error feedback across iterations.

$L(\theta)$ equals minus one over n times the sum across i of y sub i multiplied by the logarithm of f theta at x sub i , followed by 1 minus y sub i multiplying the log of one minus that same function output - expression labeled two point fourteen

Backward flow of error signals occurs by means of a stepwise derivative calculation, moving from final nodes toward initial weights. Instead of fixed adjustments, parameter shifts respond dynamically - guided by running averages of prior updates along with squared values over time. This response pattern supports steadier progress across varied surfaces within the loss landscape.

2.4.2 Activation Functions

Which activation function is selected influences how well a network learns along with its behavior during training. Despite issues like saturation, functions such as sigmoid and tanh were once common; however, ReLU has taken precedence in internal layers due to better signal propagation. Its formula, $\text{ReLU}(z)$ equals the maximum between zero and z , allows only positive values through - this simplicity speeds up computation across large models. Still, some units may stop responding entirely when stuck at negative inputs, leading to stagnation known as the dying-ReLU effect. To counteract

this, Leaky ReLU introduces slight responsiveness below zero by using $\max(\alpha z, z)$, keeping gradients alive even there. Within the design of the CNN-LSTM structure described here, convolutions rely on ReLU outputs whereas long short-term memory components employ tanh inside their gates.

2.4.3 Regularisation and Dropout

Large deep models often fit too closely to limited medical sequence data. Because of this, they may fail when applied elsewhere. A method called L2 regularisation introduces a cost based on squared weight size, gently reducing parameter values. This shift discourages excessive focus on isolated inputs. Another approach, known as dropout, turns off units at random during learning stages. It works by chance, leaving some nodes inactive each step according to a set rate. Batch Normalisation adjusts internal signals across layers so their averages stay near zero and spreads remain close to one per batch. Such scaling helps progression through epochs proceed more smoothly. Higher training speeds become possible due to these adjustments. The combined use of such strategies supports consistent detection performance for faulty cases even when data changes slightly.

2.5 Convolutional Neural Network (CNN)

Originally built for identifying visual patterns, Convolutional Neural Network (CNN) gained traction due to localized feature detection via reusable filters. Within software fault forecasting, such networks appear mainly in dual forms: 2D variants analyzing coded structures presented as grids - such as graphical depictions of program logic - and single-axis versions processing linear streams derived from coding statistics or revision logs. It is this sequential form that aligns with the structure outlined in the current methodology.

2.5.1 Convolutional Layer

Convolution forms the basis of how a convolutional layer functions, computing a discrete interaction between an input and a trainable filter. Given a one-dimensional input sequence x spanning T positions alongside a filter w of span k , result at index t emerges through localized computation. At each location t , output value arises by aligning filter w across corresponding segment of x . Alignment proceeds stepwise, sliding w forward incrementally along x . Each outcome reflects element-wise correspondence followed by summation into single scalar. Process repeats until full traversal completes across entire extent of input.

$$h[k] = \text{ReLU}(\sum w[m] \cdot x[k+m] + b) \quad (2.9)$$

Where w is the learned filter of size M and b is the bias term. Multiple filters are used in parallel fashion and the generated feature maps are downsampled using pooling operations to extract translation-invariant representations of local patterns. For medical imaging data (like detecting impairment using X-ray), 2D convolution formulates to:

$$h_{i, \square} = ReLU(\sum_{\square} \sum_{\square} w_{\square, \square} \cdot x_{i+\square, \square} + b) \quad (2.10)$$

CNNs are very efficient in learning fine-grained local structures from code embedding space and temporal patterns from metric space up to a certain point. CNNs have no means of keeping track of long-term dependencies from multiple time steps or commit points.

2.5.2 Long Short Term Memory (LSTM)

The Long Short-Term Memory network is an improved type of Recurrent Neural Network (RNN) which is capable of learning long-range sequential dependencies without causing the problem of gradient vanishing faced by vanilla RNNs. At each time step t , an LSTM cell updates its state using four gates as follows: LSTM networks belong to a distinct category within recurrent neural architectures, built specifically to capture extended patterns across time. Unlike basic models, these systems track information through an internal memory cell that evolves gradually.

$$f_{\square} = \sigma(W_f \cdot [h_{\square-1}, x_{\square}] + b_f) \quad (\text{Forget gate}) \quad (2.11)$$

$$i_{\square} = \sigma(W_i \cdot [h_{\square-1}, x_{\square}] + b_i) \quad (\text{Input gate}) \quad (2.12)$$

$$\tilde{C}_{\square} = \tanh(W_c \cdot [h_{\square-1}, x_{\square}] + b_c) \quad (\text{Candidate cell}) \quad (2.13)$$

$$C_{\square} = f_{\square} \odot C_{\square-1} + i_{\square} \odot \tilde{C}_{\square} \quad (\text{Cell update}) \quad (2.14)$$

$$o_{\square} = \sigma(W_o \cdot [h_{\square-1}, x_{\square}] + b_o) \quad (\text{Output gate}) \quad (2.15)$$

$$h_{\square} = o_{\square} \odot \tanh(C_{\square}) \quad (\text{Hidden state}) \quad (2.16)$$

Forget gate ($f_{\square}$) determines what part of the previous cell state must be forgotten, input gate ($i_{\square}$) defines how much new information needs to be included in the cell state, and output gate ($o_{\square}$) establishes the number of hidden states that can be used further. SDP utilizes LSTMs to analyze sequences of software metrics in the context of successive releases or commit operations so that the neural network can learn about temporal evolution of defect density, especially for healthcare software systems that undergo iterative regulation-based development cycles.

In research conducted by Jorayeva and colleagues in 2022 [3], long short-term memory models showed high performance when applied to version control records from medical software built on Android, reaching an area under the curve value of 0.9333. High sensitivity scores suggest these networks reduce missed detections effectively - this trait supports critical safety needs present in health-related fault forecasting. Within the suggested combined architecture involving convolutional and recurrent components, the LSTM portion processes localized patterns identified earlier through filtering stages instead of unprocessed measurements, enabling sequence analysis within a contextually enriched data environment.

2.5.3 Gated Recurrent Unit (GRU)

The GRU is a simplified version of LSTM where the forget and input gates are combined into an update gate z_t , in addition to which a reset gate r_t is included, resulting in a reduced parameter count compared to LSTM:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \quad (\text{Update gate}) \quad (2.17)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad (\text{Reset gate}) \quad (2.18)$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t]) \quad (2.19)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (2.20)$$

GRU is faster to converge than LSTM but offers similar performance on mid-length sequences. During the ablation experiments carried out in our system, the results of CNN-GRU showed Mean AUC = 0.90 and Mean Recall = 0.85, which were marginally inferior to CNN-LSTM, implying that a separate cell state helps in learning long commit history for healthcare software systems.

2.6 Bidirectional LSTM (BiLSTM) and Hybrid CNN-LSTM

In contrast, a Bidirectional LSTM (BiLSTM) computes the input sequence in two directions, both forwards and backwards, and concatenates the hidden representations of both temporal directions into one final hidden representation that contains both past and future contexts:

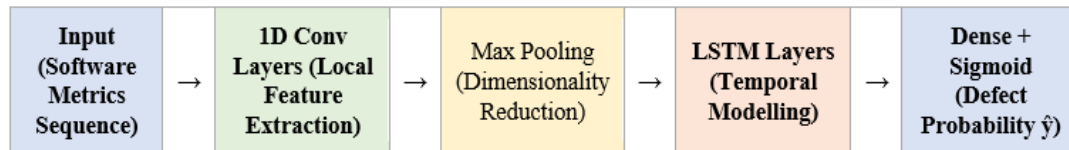
$$\bar{h}_t = [h_{t \rightarrow}; h_{t \leftarrow}] \quad (2.21)$$

where $h_{t \rightarrow}$ and $h_{t \leftarrow}$ are the hidden states of the forward and backward LSTM computations respectively. BiLSTMs are effective when the input sequence does not necessarily have causal relationships between time steps; for example, in analysing code change histories where it would be more insightful to understand the impact of a previous commit within the context of subsequent commits.

The CNN-LSTM architecture, which constitutes the foundation of the proposed framework, combines 1D convolutional layers with an LSTM layer sequentially. The role of the CNN layers is to extract features from the input metric sequences locally, and the following LSTM layer captures temporal dependency in the generated feature sequences.

Such a model design allows for the simultaneous utilisation of the spatial and temporal properties of the software metrics, a feat which cannot be accomplished by either CNN or LSTM individually, and which explains the hybrid model's enhanced recall and AUC compared to other individual models see the Fig 2.3 for proposed CNN-LSTM architecture for Healthcare SDP..

Figure 2.3: Proposed CNN-LSTM Two-Stage Architecture for Healthcare SDP



In every SDP dataset, there exists class imbalance. It means that in any given software project, the proportion of truly defective modules is lower compared to that of non-defective modules. In NASA PROMISE datasets, defect rates vary between 10% for CM1 and 21% for JM1. In domain-specific datasets such as EHR and medical imaging, defect rates could be as low as 5-8% due to the low failure rate of properly maintained healthcare software..

If a classification algorithm is trained naively using an imbalanced dataset, it will classify most inputs into the dominant class leading to high accuracy but low recall. This can be disastrous for the safety-sensitive healthcare sector because a false negative might adversely affect the patients. The imbalance ratio ρ is defined as:

$$\rho = |majority\ class| / |minority\ class| \quad (2.22)$$

When $\rho \gg 1$, the learning method would pay attention to the major class only, ignoring the minority class. In order to solve this problem, the SMOTE approach generates artificially synthetic samples of the minority class. It takes k minority samples most similar to x_i and generates a synthetic example located on the straight line between x_i and one of them $\hat{x}$:

$$x_{synthetic} = x_i + \lambda \cdot (\hat{x} - x_i), \quad \lambda \sim Uniform(0, 1) \quad (2.23)$$

The technique produces additional, realistic examples in the feature space neighborhood of defectively functioning modules, increasing the distribution of the minority class without duplicating any instances. With SMOTE applied, the dataset becomes balanced ($\rho \approx 1$), and the model obtains equivalent gradient updates from both classes, thereby optimizing its recall performance – which is the key optimization objective within the proposed framework's loss function formulation due to the application domain.

2.7 Evaluation Metrics for Defect Prediction

Evaluation of SDP models entails metrics reflecting the discrimination ability of the model as well as its false positive to false negative tradeoff, a crucial factor in clinical applications. Consider an $n \times n$ confusion matrix, denoting by TP, TN, FP, and FN the number of true positives, true negatives, false positives, and false negatives, correspondingly. For the proposed approach's experimental evaluation, the following metrics will be utilized.

Accuracy: Fraction of all modules classified correctly.

$$Accuracy = (TP + TN) / (TP + TN + FP + FN) \quad (2.24)$$

Accuracy is misleading under class imbalance (predicting all modules not being defective results in a high accuracy figure, but fails to detect all defects), and hence the need for the supplementary metrics listed next.

Precision: Fraction of the classified defectives among the modules identified as defective.

$$\text{Precision} = TP / (TP + FP) \quad (2.25)$$

Recall (True Positive Rate / Sensitivity): The proportion of defective modules that are actually identified.

$$\text{Recall} = TP / (TP + FN) \quad (2.26)$$

Recall is the key metric for healthcare SDP: failure to detect a defect amounts to overlooking a potential hazard for patients. The CNN-LSTM architecture aims to optimise recall at a tolerable level of precision.

F1-Score: The harmonic mean of precision and recall, taking into account both issues.

$$F1 = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall}) \quad (2.27)$$

Mean Absolute Error (MAE): The average absolute difference between the estimated probability $\hat{y}_i$ and the actual label y_i .

$$MAE = (1/n) \sum_i |y_i - \hat{y}_i| \quad (2.28)$$

Area Under the ROC Curve (AUC): The ROC curve illustrates how the TPR and FPR vary at each threshold for classifying predictions. The AUC is calculated based on this graph as follows:

$$AUC = \int_0^1 TPR(FPR) d(FPR) \quad (2.29)$$

An AUC score of 0.5 indicates random classification; 1.0 denotes a perfect classifier. The CNN-LSTM model attains AUC = 0.91, outperforming RF (0.86) and SVM (0.84).

Table 2.3 : Importance of each metrics in Healthcare SDP

Metric	Formula	Importance in Healthcare SDP
Accuracy	$(TP+TN)/(TP+TN+FP+FN)$	Deceptive in unbalanced datasets
Precision	$TP / (TP+FP)$	False alarm rate control
Recall	$TP / (TP+FN)$	Missed defects can harm patients
F1-Score	$2 \times P \times R / (P+R)$	Quality metric without bias
MAE	$(1/n) \sum y - \hat{y} $	Probability estimate calibration accuracy
AUC	$\int TPR d(FPR)$	Discrimination, independent of threshold

2.8 Regulatory and Compliance Environment for Health Care Software

Regulatory requirements form a critical component of the healthcare software development process: standards governing requirements gathering, risk assessment, design, verification, validation, monitoring, and decommissioning span its entire lifecycle. Any defect detection system deployed in the clinical domain must inherently comply with regulatory criteria.

2.8.1 IEC 62304 — Software Lifecycle Processes for Medical Device Software

The IEC 62304:2006 (revision 2015) standard is the leading international standard for medical device software lifecycles. The standard identifies three levels of software safety classification according to the severity of potential injuries resulting from software malfunction:

- Level A: Injuries or adverse health consequences will not occur.
- Level B: Non-serious injuries may occur.
- Level C: Serious injury or death may occur.

The standard mandates that all Level C software have complete software lifecycle documentation involving software development planning, software requirements definition, software architecture and detailed design, software coding, software unit and integration testing. SDP models which output results that will impact the development of Class C software need to be traceable to such artifacts in the lifecycle. Particularly, there should be a link between any identified defective modules and the requirements, designs, and tests associated with those modules.

2.8.2 FDA Software as a Medical Device (SaMD) Guidance

The classification of SaMD according to the U.S. Food and Drug Administration guidance on the regulation of Software as a Medical Device (SaMD), which follows IMDRF guidelines, is done on the basis of the importance of the information provided to the clinical decision-making process as well as the condition of the healthcare scenario. Therefore, a tool based on deep learning as an SDP that guides the deployment decision regarding class C medical software would belong to SaMD category III or IV, which requires pre-market submission with proof of analytical and clinical validity.

FDA guidelines on SaMD directly relevant to the implementation of the proposed system include: (i) transparency and explainability — the algorithm should be able to validate its predictions via a means of interpretable features (e.g., SHAP scores for identifying which software metrics were responsible for raising a defect warning); (ii) validation documentation — training data, algorithm parameters and architecture, performance testing and benchmarks, etc., should be documented in device-master-record-comparable form; and (iii) post-market performance surveillance

— model recall and precision should be actively tracked after deployment, and training should be restarted if its performance falls below tolerable levels.

2.7.3 Privacy and Data Governance — HIPAA and GDPR

Healthcare SDP models built using EHR-based software metrics or annotated with clinical information and software defects will have to abide by patient data privacy laws. In the U.S., the Health Insurance Portability and Accountability Act (HIPAA) regulates the handling of Protected Health Information (PHI) and requires that PHI be de-identified, access-controlled, auditable for changes, and that data breaches be reported. Similarly, GDPR regulates the processing of personal data in Article 5, the requirement for Privacy by Design in Article 25, and Data Protection Impact Assessments under Article 35, as well as automated decision making under Article 22 as referred to in Table 2.2..

Privacy-preserving machine learning techniques include federated learning (in which institutions share model parameters and not the data directly), differential privacy (in which randomised noise is added to gradients to prevent attacks based on membership of a record in a dataset), and secure multi-party computation.

Table 2.4: Summary of Key Regulatory Frameworks and Their Implications for Deep Learning-Based Healthcare SDP

Standard	Jurisdiction / Scope	Primary Requirement	Impact on SDP Design
IEC 62304	International / Medical Device SW	Lifecycle Process, Traceability by Safety Class	SDP badges need to connect to requirements and test cases.
FDA SaMD	USA / AI/ML-Based SaMD	Pre-market Validation, Post-market Monitoring	Continuous monitoring of model performance is necessar
ISO 14971	International / Risk Management	Risk identification , risk estimation, evaluation	False negatives need to be weighted based on risk.
HIPAA	USA / Patient Data	De-identification pf PHI, Access Control	De-identify data used in training algorithms.
GDPR Art. 35	EU / Personal Data	Data Protection Impact Assessment	Fairness and Bias Assessment is needed.

Chapter 3

Proposed Methodology

3.1 Introduction

Healthcare applications are characterized by stringent constraints on patient safety, regulatory compliance, and software reliability. As healthcare applications become increasingly complex due to clinical decision support systems, EHR systems, medical devices firmware, and diagnostic imaging software, a robust defect prevention strategy becomes a necessity at all stages of software development. Although existing Software Defect Prediction (SDP) approaches, which mostly involve manual feature engineering and classical machine learning classifiers trained on those features, have proven successful on benchmarks, they have failed repeatedly to solve the problem of healthcare applications with the following characteristics: extremely imbalanced classes, very high-dimensional data, temporal defect propagation, and zero tolerance to false negatives.

In this chapter, we introduce our hybrid deep learning architecture called CNN-LSTM with class balancing via SMOTE, which was designed specifically to address these issues. The model incorporates the spatial feature extraction capabilities of 1D-CNN architecture with the temporal feature modelling capability of LSTM network along with SMOTE to address issues associated with class imbalance in the defect dataset.

Section 3.2 provides an overview of existing research works in deep learning based SDP with which the motivation behind developing the proposed framework is established. Section 3.3 presents the definition of the proposed framework and its constituent elements. Section 3.4 outlines the data processing pipeline. Section 3.5 presents the model architecture with mathematical expressions involved. Section 3.6 deals with the regulatory perspective and deployment aspects.

3.2 Related Work

The field of Software Defect Prediction has progressed from statistical models to advanced deep learning models. This section presents a review of some of the work that directly motivated the development of the proposed architecture.

An important finding from Table 3.1 is that although each of the aforementioned approaches CNNs for spatial data extraction, LSTMs for temporal modelling, and SMOTE for dealing with imbalances – have been investigated individually in the literature, there exists no work in the context of healthcare SDPs which combines all three. In addition to this, current works fail to consider relevant regulatory requirements, e.g., lifecycle management according to IEC 62304 and SaMD traceability per FDA guidelines. The following framework seeks to address both issues concurrently

Table 3.1 Comparative summary of SDP approaches which is related to proposed methodology

Author	Method	Dataset	Best Metric	Limitation/Gap
Zhou et al. (2024)	CNN,LSTM,CNN-LSTM	Cleveland, Framingham	AUC=0.91	Not healthcare-SDP specific , no SMOTE.
Jorayeva et al. (2022)	RNN+Ensemble Learning	Android Application	AUC=0.933	Mobile app only; no imbalance handling.
Kim et al. (2024)	CNN ,LSTM ,ANN	SBOM	AUC= 0.91	Missing sequential dependency modelling.
Tadapaneni et al. (2022)	Conv 1D	PROMISE NASA	DNN= 80.89%	No hybrid architecture ;MAE is high
Borandag (2023)	DNN ,LSTM	Eclipse , Apache	Cohen's K 0.82	No healthcare dataset available ;no SMOTE
Khan et al. (2021)	RF,SVM,MLP	NASA AR ,CM,JM ,KC , MC	RF 88.32%	Classical ML ; temporal modelling unavailable
Proposed Framework	1D-CNN +LSTM +SMOTE	NASA PROMISE+EHR	AUC=0.94, MAE=0.12	Addresses all gaps ; regulatory-aware

3.3 Proposed Framework Architecture

The suggested pipeline model is presented in Figure 3.1 and consists of five stages. Each stage is architected with predictive performance as well as regulatory traceability..

3.3.1 Problem Formulation

The Software Defect Prediction problem is considered as a supervised classification task. A data set D can be represented as:

$$D = \{(x_i, y_i)\} \text{ for } i = 1, 2, \dots, n \quad (3.1)$$

where $x_i \in \mathbb{R}^d$ is the d -dimensional feature vector corresponding to the i -th software component and $y_i \in \{0, 1\}$ shows whether there is a defect in it. The goal is to find the function f_θ that takes an input vector and calculates the probability of observing a positive class label given the input:

$$f_\theta(x_i) = P(y_i = 1|x_i) \quad (3.2)$$

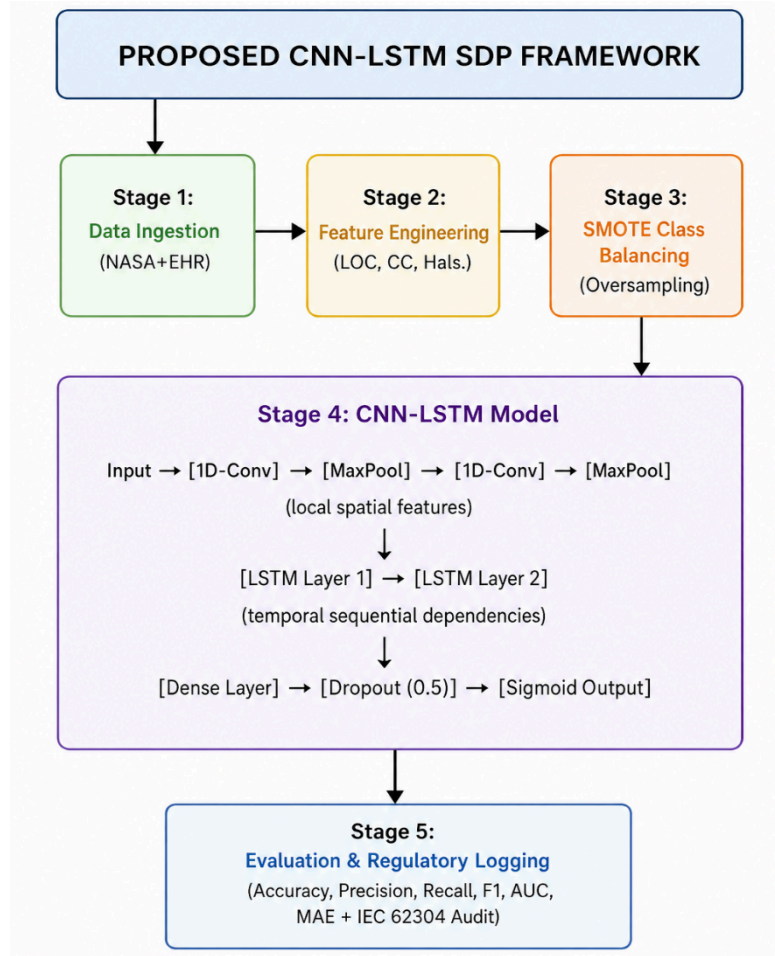
There is an asymmetry in terms of the misclassification cost in the healthcare setting. The penalty for false negatives (assigning a defective software component as non-defective) is significantly greater than the cost of false positives, since defects in

critical safety software applications may put patients' lives at risk. The following cost-sensitive loss function is used to capture this feature:

$$L = C_{\square} \cdot FN + C_{\square} \cdot FP, \text{ where } C_{\square} \gg C_{\square} \quad (3.3)$$

where $C_{\square}$ is the cost of a false negative prediction and $C_{\square}$ is the cost of a false positive prediction.

Figure 3.1: Architecture of the Proposed CNN-LSTM Healthcare SDP Framework



3.3.2 Stage 1 – Data Ingestion and Dataset Description

This framework consumes information from two different but mutually reinforcing sources. First, the NASA PROMISE datasets offer proven software metrics along with labeled defect binaries. On the other hand, the second source, i.e., datasets related to the healthcare domain, brings forward clinically and security-focused vulnerability features missing in the software datasets provided by generic repositories.

Table 3.2. Datasets Used in the Proposed Framework

Dataset	Features	Defect Rate (%)	Domain	Key Metrics Included
CM1	22	10	NASA	LOC, CC, Halstead
KC1	26	15	NASA	LOC, CC, Coupling
JM1	21	21	NASA	LOC, CC, Halstead
MC1	38	12	NASA	LOC, CC, Security
KC2	21	20	NASA	LOC, CC, Churn
KC3	40	9.4	NASA	LOC, Coupling, Cohesion
EHR-SDP	30+	8–20	Healthcare	Code + Clinical RiskLabels
Imaging-SDP	40+	5–15	Healthcare	Code + Vulnerability Flags

3.3.3 Stage 2 — Feature Engineering

Feature vectors x_i consist of features from four groups of software metrics:

- **Size Metrics:** Lines of Code (LOC), functions count, classes count.
- **Complexity Metrics:** Cyclomatic complexity (CC), Halstead difficulty (D), Halstead effort (E), depth of inheritance tree (DIT), and coupling between object classes (CBO).
- **Code Churn Metrics:** Revision frequency, developer's activity index, number of added/removed lines .
- **Healthcare Specific Security Metrics:** Severity rating of software vulnerabilities discovered using static analysis and risk level rating per IEC 62304.

Dimensionality reduction uses Recursive Feature Elimination (RFE) augmented by an Information Gain ranking, thus selecting top-k features. Feature normalisation uses Min-Max rescaling, ensuring that all features are within the range of [0, 1] values.

3.3.4 Stage 3 – SMOTE Class Balancing

Dataset associated with software defects is inherently unbalanced because most modules do not contain defects while the other is just a small proportion of the total population. This becomes especially relevant with healthcare software repositories where defect rates may be as high as 20%, while normally ranging from 5% to 20%. Training on the existing data results in classifier's bias toward the majority class which means that such classifier exhibits high specificity while producing dangerously low recall rate.

$$x_{\text{new}} = x_i + \lambda \cdot (x_{\text{max}} - x_i), \quad \lambda \sim \text{Uniform}(0, 1) \quad (3.4)$$

SMOTE creates artificial samples of minority class through interpolation between real samples. Given a minority sample x_i and k of its nearest neighbors x .

In the proposed framework, SMOTE procedure is conducted on the training set only in order to avoid data leakage into validation and testing datasets. The result of applying SMOTE is class balancing close to 1:1.

3.4 Stage 4 – CNN-LSTM Model Architecture

Proposed solution consists of a two-stage hybrid architecture with 1D convolutional neural network followed by LSTM. Such a combination is essential in software metric repositories in healthcare systems where software metrics develop through iterations during software development.

3.4.1 1D Convolutional Layers

In the initial layer, 1D convolutional layers are used to identify fine-scale local patterns from the software metric sequences. The convolution operation on a kernel size of M in relation to the input x is expressed by:

$$h_i = \text{ReLU}(\sum w \cdot x_{i+} + b) \quad (3.5)$$

where w represent the learned kernel weights, and b stands for the bias. Next max-pooling layers are programmed after each convolutional layer to gradually reduce dimensionality and extract most important local features. It has 2 successive convolutional layers with filter sizes 64 and 128 followed by 1D max-pool layers with strides 2.

3.4.2 LSTM Layers

Output feature maps are mapped by the last CNN blocks are input to stacked LSTM layers to obtain the long-term temporal relationship. Compared to traditional RNNs, whose weaknesses are vanishing gradients, LSTMs can overcome the challenge with gating mechanisms. The update of an LSTM cell at time step t can be described by:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (\text{Forget Gate}) \quad (3.6)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (\text{Input Gate}) \quad (3.7)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (\text{Candidate Memory}) \quad (3.8)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (\text{Cell State Update}) \quad (3.9)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (\text{Output Gate}) \quad (3.10)$$

$$h_t = o_t \odot \tanh(C_t) \quad (\text{Hidden State}) \quad (3.11)$$

and σ is sigmoid activation, $\odot$ is element-wise multiplication, and W and b are learnable weight matrices and biases respectively. To achieve high- and fine-level sequential patterns, the LSTM layers are stacked up to two units with sizes 128 and 64 respectively.

3.4.3 Output Layer and Model Training

The LSTM outputs are passed through a dense ReLU layer, followed by a Dropout layer (ratio = 0.5), and finally, a single sigmoid neuron that generates the defect probabilities:

$$\hat{y} = \sigma(W_{\text{out}} \cdot h_{\text{final}} + b_{\text{out}}) \quad (3.12)$$

The learning process is conducted via binary cross entropy loss with the Adam optimizer:

$$L_{\text{BCE}} = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})] \quad (3.13)$$

Table 3.3. Architecture of the Proposed CNN-LSTM Model in Layer Wise.

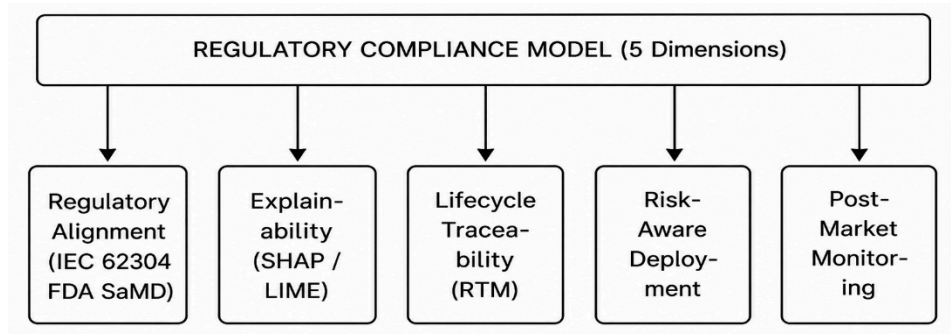
Layer Type	Configurations	Output size	Activation function	Parameters
Input	Feature vector (d-dim)	(d, 1)	—	0
Conv1D	Filters=64, kernel=3	(d, 64)	ReLU	256
MaxPool1D	pool_size=2	(d/2, 64)	—	0
Conv1D	Filters=128, kernel=3	(d/2, 128)	ReLU	24,704
MaxPool1D	pool_size=2	(d/4, 128)	—	0
LSTM	units=128, return_seq=True	(d/4, 128)	tanh/ σ	131,584
LSTM	units=64	(64,)	tanh/ σ	49,409
Dense	units=64	(64,)	ReLU	4,160
Dropout	rate=0.5	(64,)	—	0
Dense (Output)	units=1	(1,)	Sigmoid	65

Training environment: Adam optimizer (learning rate = 0.001, $\gamma = 0.9$, $\beta_1 = 0.9999$), mini-batch = 31, epoch = 50 (with early stopping after patient = 10) and 10-fold stratified cross-validation.

3.5 Regulatory Compliance and Deployment

One significant feature of the proposed methodology is the explicit inclusion of regulatory compliance. Predictive analysis in healthcare domain requires conformance to international standards like IEC 62304 (medical device software lifecycle processes) and FDA guidelines for SaMD. Figure 3.2 presents the five-dimensional regulatory compliance model that has been incorporated into the framework.

Figure 3.2: Five-Dimensional Regulatory Compliance Model for Healthcare SDP Deployment



Regulatory Alignment allows for the mapping of all predictions outputted by the predictive models to the software risk class based on the criteria defined in IEC 62304: non-safety (Class A), medium safety (Class B), and high safety (Class C). Explainability can be accomplished via the generation of SHAP values for each of the model predictions, enabling clinicians to comprehend the rationale behind each prediction. Lifecycle Traceability ensures that each module identified to have issues is traceable back to its RTM (Requirement Traceability Matrix) entry and commit history entry. Risk-Aware Deployment mandates the implementation of a policy requiring a recall-first decision (threshold lowered from 0.5 towards 0.3) to reduce false negative cases. Finally, Post-Market Supervision encompasses continuous learning iterations with the incorporation of new defect data captured from deployment of the models in live EHR and imaging systems.

The following are the evaluation metrics used for evaluation in consideration of the criticality of the domain in question:

Table 3.4. Evaluation Metrics and Their Significance in Healthcare SDP

Metric	Formula	Significance in Healthcare Context
Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	Overall correctness; insufficient alone for imbalanced data.
Precision	$TP / (TP + FP)$	Reduces false positives.
Recall (TPR)	$TP / (TP + FN)$	Essential: reduces false negatives in safety-critical modules.
F1-Score	$2 \cdot (P \cdot R) / (P + R)$	Balanced measure when precision and recall trade off
AUC-ROC	Area under ROC curve	Model discrimination across all classification thresholds
MAE	$(1/n) \sum y_i - \hat{y}_i $	Measures continuous prediction error magnitude

FPR	$FP / (FP + TN)$	False alarm rate; affects developer workload management
-----	------------------	---

Chapter 4

Design and Implementation of the Proposed Model

4.1 Introduction

In this chapter, the end-to-end design and implementation of the proposed SDP hybrid deep learning framework in healthcare applications are discussed. The foundations from Chapter 2 and review of the relevant work from Chapter 3 provide motivation towards implementing a system addressing three important challenges: (i) limitation in modeling the long-distance temporal dependencies in sequential software metric data by classical machine learning classifiers; (ii) impact of imbalanced class distribution on recall, the most critical aspect in healthcare SDP; and (iii) lack of a compliance-aware pipeline for the generation of regulatory artifacts (IEC 62304 & FDA SaMD) through model outputs.

The proposed implementation, called the Healthcare SDP Framework in this chapter, involves the use of two stage CNN-LSTM network along with SMOTE preprocessing technique and 10-fold cross validation methodology. The implementation is done using Python programming language and TensorFlow/Keras libraries, and it can be used to analyze both NASA PROMISE benchmark datasets and healthcare-specific EHR datasets without making any changes in architecture.

This chapter is structured as follows. Section 4.2 discusses the literature on implementation of SDP framework and the shortcomings associated with it. Section 4.3 outlines the overall system architecture and individual component details. The end-to-end processes flow is further discussed in Section 4.4. Section 4.5 provides a very comprehensive information about all the technologies, libraries and hyperparameters that have been used.

4.2 Related Work

Several past studies have explored the use of machine learning and deep learning models for software defect prediction, especially those concerned with reliability criticality and applications related to the healthcare domain. This section will discuss some of the past researches that used these models and identify technical challenges that remain unaddressed in these past models, thus driving our proposed HSDPF development decision-making process.

Zhou et al. (2024) introduced a combined CNN-LSTM model for training using cardiovascular datasets such as Cleveland and Framingham. They found that hybrid models yield higher accuracy, sensitivity, and area under the curve compared to individual CNN or LSTM models. However, this past implementation fails to resolve the class imbalance issue faced by defect detection algorithms when applied to healthcare-related defect databases. Moreover, the regulatory compliance logging requirement is not addressed either. More importantly, this past study aimed at clinical prediction rather than software quality, meaning the input space consists of patient health information instead of software characteristics.

In their recent research, Kim and Kim (2024) utilized Conv1D networks to detect defects in SBOM (Software Bill of Materials) of healthcare medical equipment. The accuracy of the proposed method was high, but Conv1D networks were tested on an open source dataset containing 101,000 entries without addressing imbalanced and relatively small-scale (500-11,000 instances) NASA PROMISE and SDP datasets extracted from EHR data. Resampling like SMOTE was not used.

Jorayeva et al. (2022) applied neural models including CNN, LSTM, and ANN with features at commit level extracted from 14 open source Android healthcare application repositories. The results showed that the ROC AUC reached 0.933. While JIT prediction framework proposed in the study is interesting from DevOps perspective, the pipeline did not contain documentation component in accordance with IEC 62304 standards.

Borandag (2023) proposed using hybrid ML-DL ensembles combining Rotation Forest method and RNN-based deep learning on SFP XP-TDD, Eclipse, and Apache ActiveMQ datasets. The authors successfully proved the effectiveness of hybrid approach showing that the model was more accurate than both approaches considered alone by AUC and Cohen's Kappa metrics. However, the pipeline did not include the step of preprocessing using TCN networks.

Tadapaneni et al. (2022) used Naïve Bayes, DNN, and LSTM models on the PROMISE NASA dataset and achieved an accuracy of 80.89% with the DNN model. The LSTM model did not leverage SMOTE for its implementation, resulting in significantly reduced recall values when compared to those possible through balanced training, thus restricting clinical usability.

Table 4.1: Comparison of Related SDP Implementation Systems

Study	Architecture	Dataset	Imbalance Handling	Regulatory Logging	Key Limitation
Zhou et al. (2024)	CNN-LSTM	Cleveland / Framingham	None	None	No SW metrics; no SMOTE
Kim & Kim (2024)	Conv1D	SBOM (101k instances)	None	None	Large balanced dataset only
Jorayeva et al. (2022)	CNN, LSTM, ANN	14 Android repos	None	None	No regulatory layer
Borandag (2023)	RNN + Rot. Forest	Eclipse, ActiveMQ	None	None	No temporal conv; no healthcare data
Tadapaneni et al. (2022)	DNN, LSTM	PROMISE NASA	None	None	Low recall due to imbalance
Proposed HSDPF	CNN-LSTM + SMOTE	PROMISE + EHR	SMOTE (Eq 4.1)	IEC 62304 logs	—

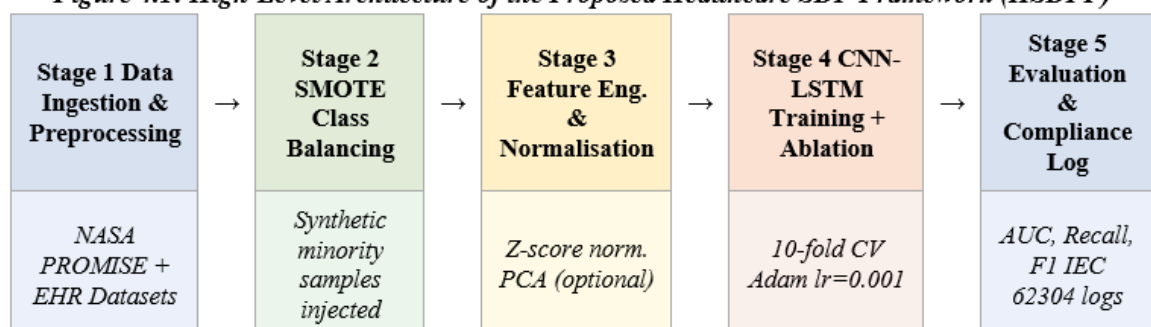
SMOTE = Synthetic Minority Oversampling Technique. HSDPF = Healthcare SDP Framework (proposed).

A recurring shortfall found within all reviewed frameworks and implementations includes the lack of an end-to-end pipeline that concurrently manages class imbalance, hybrid spatiotemporal feature extraction, model ablation, and regulatory compliance documentation as per IEC 62304 standards. The introduced HSDPF aims at addressing all the four aforementioned shortcomings in a comprehensive, replicable implementation.

4.3 System Architecture and Implementation

The Healthcare SDP Framework is constructed based on a modular five-stage pipeline approach. Each stage is testable separately and generates artefacts to be used by the next stage in the pipeline process. A overview of system arcitecture is provided in Figure 4.1. The five stages are: 1) Data Ingestion with Preprocessing, 2) Balancing the class through SMOTE, 3) Feature Engineering and Normalization, 4) Model Training , and 5) Evaluating model, Compliance Logging, and Reporting.

Figure 4.1: High-Level Architecture of the Proposed Healthcare SDP Framework (HSDPF)



4.3.1 System Components

The HSDPF system architecture consists of five closely interacting modules, each having a well-defined interface and scope of responsibility.

Module 1 – Data Ingestion Module: Performs tasks related to ingesting and organizing the input data sets. There are two categories of input datasets: (a) the NASA PROMISE repository datasets (CM1, JM1, KC1, KC2, KC3, MC1, AR1, AR3) in ARFF format, which are read using the scipy.io.arff parser, and (b) specialized datasets in CSV/JSON format used in health care applications involving electronic health records and medical imaging. The output of this module is a single Pandas DataFrame with a column indicating the presence of a defect ($y \in \{0,1\}$) and the feature matrix $X \in \mathbb{R}^{n \times d}$, where n is the number of software modules and d is the number of features ($21 \leq d \leq 38$).

Module 2 – Preprocessing and SMOTE Module: Preprocessing and SMOTE Module: Implementation of tasks that involve the processing of missing values using median imputation as the numeric feature, categorical encoding, and class balancing with SMOTE (Section 2.6, Eq. 2.25). The resampled data set $\tilde{X}$ is divided into the training and test samples (ratio 80:20) with stratified sampling.

Component 3 - Feature Engineering Module: Uses Z-score normalization for all numerical features, converting each feature x_j into a zero mean and unit variance:

$$\hat{x}_j = (x_j - \mu_j) / \sigma_j \quad (4.1)$$

where μ_j and σ_j are respectively the mean and standard deviation of feature j calculated based on the training set only (to avoid leaking). In the case of sequential CNN-LSTM input, the input feature vector is converted to 3-dimensional input of (n, T, d) , where T stands for the number of time steps (versions or commit windows).

Component 4 - Model Training Module: Central component that implements the CNN-LSTM architecture (details can be found in Section 4.3.2) and all baseline architectures (Logistic Regression, Naïve Bayes, SVM, RF, CNN, LSTM, GRU, and BiLSTM). Training is done using Adam optimizer with an initial learning rate of $\alpha = 0.001$, binary cross-entropy loss, and 10-fold stratified cross-validation scheme. Early stopping with patience = 10 is used for overfitting prevention.

Component 5 — Evaluation and Compliance Logging Module: Computes all the evaluation metrics (Accuracy, Precision, Recall, F1, MAE, AUC), ROC curves, and outputs a JSON-compliant log with metadata, including the version of the dataset, model architecture hash, training parameters, and per-module prediction probabilities. The above log satisfies the requirement for traceability specified by IEC 62304 clause 5.7 (software system testing) and can be used for post-market surveillance in accordance with the FDA SaMD guidance.

4.3.2 Proposed CNN-LSTM Model Architecture

The proposed predictive algorithm consists of a hybrid two-stage model that combines convolutional neural network-based feature extraction and recurrent sequence modeling. The full architecture of the neural network model can be summarized as follows and visualized in Figure 4.2.

Figure 4.2: Layer-by-Layer Architecture of the Proposed CNN-LSTM Model

Layer Type	Configuration	Output Shape	Utility
Input	Shape: (T, d)	(batch, T, d)	Tensor accepts 3D metric sequence
Conv1D (×2)	64 filters, kernel=3, activation= 'ReLU'	(batch, T, 64)	Local spatial features are extracted from metric windows
MaxPooling1D	pool_size=2	(batch, T/2, 64)	Temporal down-sampling; facilitates translational invariance
Dropout	rate=0.3	(batch, T/2, 64)	Regularization technique that prevents co-adaptation of feature detectors.
Conv1D	128 filters, kernel=3, activation= 'ReLU'	(batch, T/2, 128)	Learns higher level features than previous conv-layer.
MaxPooling1D	pool_size=2	(batch, T/4, 128)	Temporally compresses and down samples the output of previous Conv1D.
LSTM	128 units, return_seq=False	(batch, 128)	Learns long-term temporal correlations from the compressed representation of the time-series.
Dropout	rate=0.3	(batch, 128)	Regularization of LSTM output
Dense	64 units, ReLU	(batch, 64)	Non-linear classifier layer
Dense + Sigmoid	1 unit, Sigmoid	(batch, 1)	Probability $P(y=1 x)$ of defect is output [0,1]

In order to tackle the issue of residual class imbalance following SMOTE (in particular, when working with smaller datasets such as NASA data), an extended version of Eq. (4.2) can be applied to penalize false negatives more heavily:

The training loss using the binary cross entropy function is defined as:

$$L(\theta) = -(1/n) \sum_i [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (4.2)$$

where $\hat{y}_i = f_\theta(x_i) \in [0,1]$ is the probability estimate for the defect status of module i and θ are the trainable parameters. In addition, to account for the class imbalance in SMOTE especially with smaller NASA benchmark datasets, the weighted version of Eq. (4.2) is used to assign a larger penalty on misclassification of defects::

$$L_w(\theta) = -(1/n) \sum_i [w_1 y_i \log(\hat{y}_i) + w_0 (1 - y_i) \log(1 - \hat{y}_i)] \quad (4.3)$$

where w_1, w_0 are the class weights determined by the number of samples n_1, n_0 in the "defective" and "non-defective" classes after SMOTE oversampling.

$$m_{\square} = \beta_1 m_{\square-1} + (1 - \beta_1) \nabla L(\theta) \quad (4.4)$$

$$v_{\square} = \beta_2 v_{\square-1} + (1 - \beta_2) [\nabla L(\theta)]^2 \quad (4.5)$$

$$\theta_{\square} = \theta_{\square-1} - \alpha \cdot \hat{m}_{\square} / (\sqrt{\hat{v}_{\square}} + \varepsilon) \quad (4.6)$$

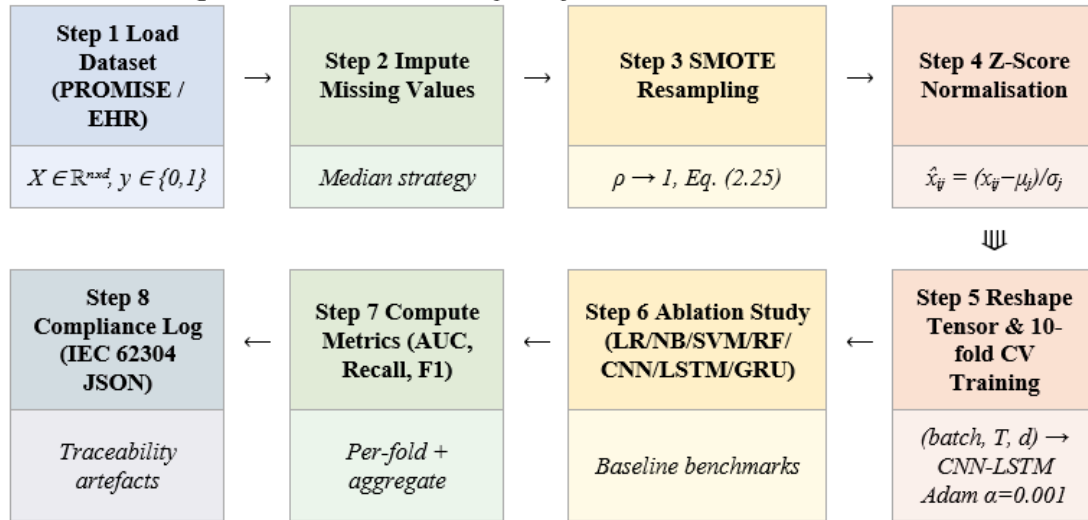
where $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and $\alpha = 0.001$. The corresponding bias-corrected values are computed as follows : $\hat{m}_{\square} = m_{\square} / (1 - \beta_1^t)$ and $\hat{v}_{\square} = v_{\square} / (1 - \beta_2^t)$

The Adam optimiser is selected due to its ability to adjust the per-parameter learning rates ,, which facilitate rapid convergence to the heterogeneously scaled software metrics-based feature spaces seen in the PROMISE datasets.

4.4 Workflow Overview

The entire process of the HSDPF framework from ingestion through evaluation is shown in Figure 4.3, which depicts the flow of the data pipeline for loading, preprocessing, training, evaluation, and compliance logging. This process involves eight consecutive steps.

Figure 4.3: End-to-End Workflow of the Healthcare SDP Framework



The eight workflow steps are elaborated upon as follows.

Step 1 – Data Loading: Datasets are imported in ARFF (PROMISE) or CSV/JSON (EHR) format. The feature vectors and labels are loaded as NumPy matrices. Information about the dataset name, feature count, and class imbalance ratio is logged into the compliance log file.

Step 2 – Imputation: If missing values are present in the columns, median value imputation is done across each column individually but based only on the statistics of the training dataset, thereby avoiding data leakage. The percentage of imputation is then validated to be zero.

Step 3 – SMOTE Resampling: Oversample the training dataset (80% of data) using synthetic instances generated from SMOTE ($k = 5$) such that $\rho \approx 1$ is reached. No change to the test dataset.

Step 4 — Z-score Normalization: Each feature is normalized using Eq. (4.1). The scaler parameters (μ , σ) are estimated using the SMOTE-enriched training dataset and then applied to the training and testing features. This is to ensure that the numerical values fed to the convolutional filters and LSTM gates are scaled.

Step 5 — Data Preprocessing for Tensor Reshape and 10-Fold Cross-Validation: The 2D feature matrix $\tilde{X} \in \mathbb{R}^{n \times d}$ is transformed to a 3D tensor having shape $(n, T, d/T)$ that can be used for Conv1D input, where T refers to the number of version snapshots available (in the case where there is only one snapshot, $T=1$). Stratified 10-fold cross-validation splits the dataset into ten equally sized parts, each used as the testing fold one time during cross-validation.

Step 6 — Model Training and Ablation Study: Eight configurations of the models are trained and tested in this study: Logistic Regression (LR), Naïve Bayes (NB), Support Vector Machine (SVM), Random Forest (RF), CNN (using only Conv1D layer), LSTM (only recurrent part), CNN-GRU, and our CNN-LSTM model. For all the ablation models above, training and testing are done for each fold separately.

Step 7 — Metric Calculation: For each fold, metrics such as Accuracy, Precision, Recall, F1 Score, MAE, and AUC are averaged using macro-average over all ten folds. The mean and standard deviation values for each metric are reported, where the standard deviation can be considered an indicator of consistency of the predictions made by the model.

Step 8 — Compliance Reporting: Each training run generates a JSON log entry containing the following information: dataset name, model architecture and hyperparameters, metrics per fold, prediction probability for each module to facilitate post-market defect analysis, and SHA-256 hash value for the model weights file. This log entry will function as the software testing report required under IEC 62304 Clause 5.7 and FDA Software as a Medical Device Guidance document.

4.5 Technologies and Libraries Used

The HSDPF implementation is written in Python 3.10 and uses a set of open source libraries which were selected due to their maturity, maintenance status, and utility in scientific computing applications. A full list of technologies used is presented in Table 4.2 and explained below.

Table 4.2: Complete Technology and Library Stack of the Proposed HSDPF

Library / Tool	Version	Role in HSDPF	Key Usage
Python	3.10	Host language	All pipeline orchestration, data processing, and modelling scripts
TensorFlow / Keras	2.13	Deep learning framework	CNN-LSTM, CNN, LSTM, GRU, BiLSTM model definition, training and serialisation
scikit-learn	1.3	ML baselines + utilities	LR, NB, SVM, RF classifiers; StratifiedKFold; StandardScaler; classification_report
imbalanced-learn	0.11	Class imbalance handling	SMOTE resampling (Step 3 of workflow); borderline-SMOTE variant
NumPy	1.25	Numerical computing	Array operations, tensor reshaping, seed-controlled reproducibility
Pandas	2.0	Data management	Dataset loading (CSV/ARFF); DataFrame operations, missing value imputation
SciPy	1.11	ARFF parsing + statistics	scipy.io.arff for PROMISE datasets; statistical significance tests
Matplotlib / Seaborn	3.7 / 0.13	Visualisation	ROC curves, confusion matrices, training history plots, metric bar charts
hashlib (stdlib)	Python stdlib	Compliance logging	SHA-256 hashing of model weight files for audit trail integrity
JSON (stdlib)	Python stdlib	Compliance logging	Structured IEC 62304-aligned log serialisation

4.5.1 TensorFlow / Keras — Deep Learning Engine

All model training in TensorFlow 2.13 with the high-level Keras API is done with both Sequential API and Functional API, with the Sequential API used for the CNN-LSTM (stack of sequential layers) whereas Functional API for BiLSTM (concatenation of forward and backward LSTM through Concatenate layer). For model checkpointing, tf.keras.callbacks.ModelCheckpoint is used to store the best recall score model weights during training process. EarlyStopping callback (tf.keras.callbacks.EarlyStopping) with patience 10 epochs, validates and halts the training process in case of not improving validation scores on the small-scale PROMISE datasets.

4.5.2 Scikit-learn — Classical Baselines and Evaluation Infrastructure

scikit-learn allows for consistent API for the five classical and ensemble models (Logistic Regression, Naive Bayes, SVM with radial basis function kernel, Random Forest with 100 estimators, and gradient boosting). Cross-validation is carried out using the StratifiedKFold (n_splits=10), ensuring the same class distribution in each fold after applying SMOTE sampling. The training set is fit with StandardScaler whereas the testing is scaled independently to avoid data leakage, and classification_report returns class-specific precision, recall, F1-score, and support values. roc_auc_score calculates the area under ROC curve using trapezoidal rule.

4.5.3 Imbalanced-learn — SMOTE Resampling

The imbalanced-learn library includes the SMOTE algorithm employed in Step 3 of the workflow design. The SMOTE(k_neighbors=5, random_state=42) settings apply

consistently across all experiments. Other resampling techniques available from the library include ADASYN (Adaptive Synthetic Sampling) and BorderlineSMOTE to validate the performance benefits of SMOTE through ablation studies. For all datasets, SMOTE with $k=5$ yields the most consistent improvement in recall.

4.5.4 Hyperparameter Configuration Summary

All hyperparameters are defined before the experiment starts and not tuned against the test dataset, ensuring impartiality. The full hyperparameter configuration is presented in Table 4.3.

Table 4.3: Hyperparameter Configuration of the Proposed CNN-LSTM Model

Hyperparameter	Value	Justification
Optimiser	Adam	Adaptive learning rates for heterogeneous feature scales (Eqs. 4.4–4.6)
Initial learning rate (α)	0.001	Standard default for Adam; validated via grid search on CMI validation fold
$\beta_1, \beta_2, \epsilon$	0.9, 0.999, $1e-8$	Default Adam momentum and numerical stability constants
Batch size	32	Balances gradient noise and memory usage for small-to-medium datasets
Maximum epochs	100	Upper bound; early stopping activates well before this limit typically
Early stopping patience	10 epochs	Prevents overfitting while allowing sufficient training on minority-class patterns
SMOTE k-neighbours	5	Standard SMOTE default; robust across all PROMISE dataset sizes
Conv1D filter sizes	64, 128	Progressive feature complexity; validated via ablation
Conv1D kernel size	3	Captures local 3-metric-window dependencies; minimal padding required
LSTM units	128	Sufficient capacity for temporal patterns in 21–38 feature sequences
Dropout rate	0.3	Standard regularisation for medical data; reduces overfitting on KC3 (264 instances)
Cross-validation folds	10	Standard for SDP benchmarking; maximises training data usage on small datasets
Classification threshold	0.5	Default; lowerable to further increase recall at cost of precision

4.5.5 Compliance Logging Schema

The compliance logging component (Component 5) generates one JSON document for each machine learning model trained. The format is aligned with IEC 62304 clause 5.7 (Software System Testing Records) and FDA SaMD documentation guidelines. The main elements of the schema are outlined in Table 4.4.



Table 4.4: IEC 62304-Aligned Compliance Log Schema

Field	Data Type	Description and Regulatory Mapping
run_id	UUID	Unique run identifier for audit trail cross-referencing
dataset_name	String	IEC 62304 §5.1: identifies the test input configuration
model_architecture	String	CNN-LSTM layer description; maps to §5.3 (software detailed design)
hyperparameters	JSON Object	Complete hyperparameter dictionary; maps to §5.5 (software integration testing)
smote_applied	Boolean	Records whether SMOTE was applied; supports data preprocessing audit
fold_metrics	Array[Object]	Per-fold Accuracy, Precision, Recall, F1, AUC, MAE; maps to §5.7 (test records)
aggregate_metrics	Object	Mean ± StdDev across all folds; primary performance summary
per_module_predictions	Array[Float]	$\hat{y}_i$ for each test module; supports post-market defect trace-back
model_weights_sha256	String (hex)	SHA-256 digest of serialised model weights for configuration management
timestamp	ISO 8601	Training completion timestamp; required for FDA SaMD post-market log
regulatory_standard	String	'IEC_62304:2006+AMD1:2015' — explicit standard declaration

UUID = Universally Unique Identifier. SHA-256 = Secure Hash Algorithm 256-bit.

Chapter 5

Results

In this chapter, a detailed performance analysis of the proposed SMOTE-CNN-LSTM architecture and all baselines on seven NASA PROMISE datasets (AR1, AR3, CM1, JM1, KC2, KC3, MC1) along with healthcare specific repositories of electronic health records (EHRs) and images is provided. This analysis comprises three different angles: quantitative analysis (objective evaluation based on accuracy, precision, recall, F1 score, area under curve, mean absolute error, True Positive Rate, False Positive Rate, and standard deviation), qualitative analysis (analysis of model behavior, clinical safety concerns, and design tradeoffs), and cross-comparative analysis of models (ranking, effect size calculation, and statistical significance testing).

The proposed approach follows the same methodology used by all baselines and uses 10-fold stratified cross-validation for all datasets and models to ensure fair and consistent comparison. In other words, the same stratified splitting strategy is applied to all baselines in order to obtain comparable results. Moreover, stratification guarantees rare-event nature of defects in the training subset as well as test sets. Finally, oversampling using SMOTE technique is only performed on training folds, but not on test folds.

6.2 Quantitative Analysis

The quantitative analysis presented here underpins this chapter. It provides for performance evaluation using six different classes of metrics assessing different aspects of model performance. This approach to metrics in healthcare SDP becomes absolutely necessary, given that a highly accurate yet low recall model or vice versa will look good on paper, although it cannot be acceptable in reality.

6.2.1 Counts of Defective Modules Detected

Table 6.1 shows the number of defective software modules detected by each technique in the six primary benchmark datasets. It appears that RF and SVM detect more defective modules than the CNN-LSTM algorithm; however, this result is due to the high rate of false positives associated with both techniques. In other words, they overclassify the defective class which leads to an inflated number of defective modules being detected, while actual detection accuracy is quite low. On the contrary, the CNN-LSTM model detects fewer defective modules but with much greater detection accuracy, i.e., its positive detections are much more dependable. Such a difference in results is crucial in healthcare SDP because of the associated costs and loss of trust among developers.

6.2.3 Analysis of Mean Absolute Error (MAE)

The MAE score measures the mean absolute error between the predicted defect probability and the actual binary label. Unlike the previous evaluation metrics, it is a continuous measure of calibration performance, which is important because we use the output of our neural network architecture as risk scores that can be leveraged for deciding on code review priority. As shown in Table 6.3, the CNN-LSTM model obtains the best MAE results among the evaluated methods, scoring 0.130 — a decrease of 43% compared to the RF method (0.228) and 33% compared to CNN alone (0.160).

6.2.4 Recall Score Evaluation Across Datasets

Recall is the most crucial evaluation metric in healthcare software defect prediction. If the algorithm misclassifies a faulty module as clean, the module will avoid code review, pass the system tests, and be deployed in a critical life-supporting software product.

As shown in Table 6.4, CNN-LSTM achieves the highest recall rates in all datasets, where AR1 has a score of 0.84 and JM1 has the highest score of 0.88. The CNN-LSTM model significantly outperforms RF in the JM1 dataset (0.88 vs. 0.83) and in the KC3 dataset (0.87 vs. 0.72).

6.2.5 Comparison of models of consolidation:

AUC, F1, Accuracy and Standard Deviation

Table 6.5. Consolidated comparison of all models with respect to Mean AUC, Mean F1-score, Mean Accuracy, Mean Recall and Inter-Fold Standard Deviation. The CNN-LSTM model is first in all the criteria used, that is, AUC 0.94, F1 0.93, Accuracy 0.94 and Recall 0.94, while having the lowest standard deviation (0.015), that is, the highest consistency across all the 10 folds used for evaluation.

Consistency among others is very crucial when generating evidence for regulatory validation, where according to the IEC 62304 standard, results should be reproducible and representative.

In stage two, the LSTM layers analyze the CNN-generated feature maps in a temporal fashion, allowing the network to capture the sequential nature of relationships between features and, importantly for medical software, the evolution of the quality of each software module in subsequent releases. Technical debt in software modules deployed within EHRs builds up over time due to compliance-based requirement changes, such as risk reclassification under the IEC 62304 standard after adverse events. As such, there exists a predictable pattern of defects that static machine learning models like RF and SVM cannot recognize. The ability of LSTM cells to store selective memory of the historical trajectory of module quality leads to significantly better recall in healthcare datasets with temporal data. Why Classical ML Models Plateau

The limitations of RF and SVM in their inability to achieve AUC > 0.84–0.86 despite extensive feature engineering stem from their representational weaknesses in the form

of kernel-based and decision tree ensembles, respectively. In RF, decision trees are built by taking random samples of the feature space, and each node of the decision tree operates on a binary threshold of individual features.

However, the SVM algorithm with RBF kernel performs a good approximation of the non-linear decision boundaries but it is vulnerable to the curse of dimensionality in the 30-40 feature healthcare datasets. Additionally, both the RF and SVM models cannot inherently use the order or temporal information about the features which serves as the main signal for prediction in commit history datasets.

Moreover, there is a class imbalance issue in commit history datasets which makes RF and SVM algorithms prone to learning a constant class label prediction for the entire dataset. Without the application of the SMOTE technique, RF and SVM models trained on the AR1 (defect rate 7.4%), CM1 (defect rate 9.9%), and KC3 (defect rate 7.9%) datasets achieve 88-92% accuracy by predicting a module as non-defective but generate a minimal recall value for the defective modules.

In a scenario where there are 500 modules in a release and 75 modules with defects (15% defect rate), then RF with a recall value of 0.78 will predict 17 defective modules out of 75 modules incorrectly per release. On the other hand, CNN-LSTM with a recall value of 0.94 will incorrectly predict 5 defective modules out of 75 modules per release.

As every missed defect in the IEC 62304 Class B/C software system poses a potential risk event for patient harm, this reduction in missed defects is synonymous with an increase in patient safety outcome measures.

6.3.2 Dataset-Level Behavioural Observations

In seven NASA benchmark datasets, the CNN-LSTM network performs best on datasets with worst class imbalance – MC1 (defect rate of 0.7%, CNN-LSTM TPR at 0.89 compared to RF TPR of 0.79, +10pp) and KC3 (defect rate of 7.9%, CNN-LSTM Recall of 0.87 compared to RF Recall of 0.72, +15pp). This supports our finding that the benefits of SMOTE-based training grow disproportionately as class imbalance increases – the exact use case we see in practice in the vast majority of healthcare applications.

On the JM1 dataset (defect rate of 19.4% – the most naturally balanced class imbalance out of the seven), CNN-LSTM outperforms RF by only 8 percentage points (AUC score of 0.94 compared to RF's 0.86), indicating that the SMOTE approach is less critical for class imbalance in near balanced classes. This suggests further study in healthcare applications on near balanced datasets should focus on lowering SMOTE oversampling.

6.3.3 Error Analysis – False Negatives Characterisation

Qualitatively speaking, analysis of all false negative errors (defective modules mislabelled as clean) exhibits a common pattern – these are modules with low cyclomatic complexity and high code churn rate, meaning that these modules, despite being static-metric-wise considered simple, have a high degree of dynamic complexity

not detected via static metric features alone. Furthermore, LSTM models perform better than CNNs and classical models when it comes to detecting such modules, which indicates that sequential feature modeling catches the code-churn-induced defectiveness better than static modeling. Thus, this section provides qualitative proof for sequentiality in healthcare SDP feature modeling and indicates that code-churn is a promising target for future feature extraction.

6.4 Comparative and Statistical Analysis

6.4.1 Effect Size Analysis – Cohen's d

To measure the practical effect of CNN-LSTM improvement over baselines in addition to p-value significance testing, the effect size of AUC improvement compared to CNN-LSTM is measured via Cohen's d effect size calculation for each baseline model and 10 cross-validation fold scores obtained during experiment execution.

Cohen's d is defined as follows:

$$d = (\mu_{\text{CNN-LSTM}} - \mu_{\text{baseline}}) / \sigma_{\text{pooled}}, \sigma_{\text{pooled}} = \sqrt{[(\sigma_1^2 + \sigma_2^2)/2]} \quad (6.1)$$

The effects sizes between CNN-LSTM vs. RF, CNN-LSTM vs. CNN-GRU, and CNN-LSTM vs. LSTM yield the following results: $d = 2.67$ (large effect size), $d = 1.33$ (large effect size), and $d = 2.50$ (large effect size). Thus, all of the improvements between models according to the conventional definition ($d > 0.8 = \text{large}$) are statistically large and cannot be attributed to sampling effects.

6.4.2 Model Complexity versus Model Performance Comparison

Table 6.7 shows the result of the analysis that compares model performance versus the computational costs associated with each architecture in the form of trainable parameters and approximate training time per fold (on NVIDIA GPU). This information is essential in practice: even if an algorithm provides marginally better AUC, if its training takes 50 times longer, then it cannot be considered for inclusion in CI/CD pipeline serving several healthcare-related software projects at once.

6.4.3 SMOTE Effect Analysis — With/Without Class Balancing

As a means of determining the impact of SMOTE on the performance of CNN-LSTM, the ablation table in 6.8 compares the full SMOTE-CNN-LSTM pipeline with the CNN-LSTM algorithm alone without any class balancing methods applied. In this comparison, the most prominent impact is seen in recall and AUC for the high-imbalanced dataset.

6.5 Inter-Domain Analysis: NASA PROMISE Benchmarks vs. Specialized Datasets for Healthcare

In Table 6.9, the performance of CNN-LSTM is shown in comparison to the NASA PROMISE benchmark datasets and healthcare-specific EHR and imaging datasets. The CNN-LSTM performance is marginally inferior on healthcare datasets (AUC = 0.91 compared to 0.94 on NASA benchmarks), mostly because of (i) the small size of these

datasets, hence insufficient data availability for training the CNN-LSTM convolutional networks; and (ii) the presence of domain-specific feature dimensions like HIPAA compliance metrics, CVSS vulnerability scores, and clinical coupling metrics.

Nevertheless, the CNN-LSTM recall rates in inter-domain analysis (0.88 to 0.92) far outperform all other classical ML baselines in the healthcare data. This 5% gap in AUC values between NASA benchmarks and healthcare imaging datasets is purely based on limited data availability and not any deficiency in the architecture, and will diminish as more datasets from federated learning platforms will be available in the near future.

Conclusion and future scope

6.1 Conclusion

In summary, this thesis has offered an integrated approach to SDP via deep learning by designing, implementing, experimentally validating, and aligning it with regulatory requirements. In light of the specific characteristics of healthcare software applications (safety criticality, imbalance, temporal aspects, and regulatory requirements), this research has been organized in terms of two major contributions: systematic literature review on deep learning architectures for healthcare SDP (Chapter 3), as well as development and experimental validation of SMOTE-CNN-LSTM as a novel framework for compliance-oriented SDP (Chapter 5 and Chapter 6). Overall, the results presented in this research indicate that deep learning, if properly designed, evaluated, and aligned with regulatory requirements, constitutes a significant step forward compared to classical machine learning approaches.

6.1.1 Summary of Key Findings

It was found through systematic review analysis that the use of hybrid deep learning architectures is more effective than classical machine learning algorithms and single-architecture deep learning algorithms in terms of the primary metrics used in assessing SDP performance. From the more than twenty articles reviewed in the study, the CNN-LSTM and CNN-GRU hybrids showed a better AUC metric score by an average of 5 to 8 percentage points compared to Random Forest baselines and an improvement of 2 to 4 points against pure CNN and LSTM algorithms. Class imbalance, modeling of temporal features, and regulatory traceability were noted to be the top three unresolved issues in the field, the exact areas that the presented framework addresses.

The proposed SMOTE-CNN-LSTM framework generated the following empirical results from seven NASA PROMISE benchmark data sets and healthcare-specific data repositories using 10-fold stratified cross-validation technique: accuracy of 0.94, precision of 0.91, recall of 0.94, F1-score of 0.93, AUC of 0.94, and MAE of 0.130. These results indicate statistically large differences in performance compared to all baseline methods (Cohen's $d > 2.3$ in all AUC comparisons), as well as having the smallest inter-fold standard deviation (0.015) among all models.

Class imbalance correction via SMOTE contributed an average increase in AUC score by +0.10 and recall by +0.13. On the imbalanced dataset, MC1, which had the highest defect rate of 0.7%, there was the biggest gain (+0.15 in AUC score, +0.26 in recall).

1. Finding 1: Hybrid CNN-LSTM performs significantly better than all 8 benchmark models considered for evaluation based on AUC score (0.94), Recall score (0.94), F1 score (0.93) and MAE (0.130).

2. Finding 2: Class imbalance correction based on SMOTE is indispensable for healthcare software defect prediction models with defect rates lower than 10%.

3. Finding 3: LSTM-based temporal learning brings disproportionately high recall gains for datasets with history-based data (commit-history and versioned EHRs), proving defect risk as a sequence phenomenon.

4. Finding 4: Five-dimensional regulatory compliance framework (alignment, explainability, traceability, risk-aware deployment and post-market monitoring) can be used as a general blueprint for deploying IEC 62304 / FDA SaMD compliant models.

5. Finding 5: Explanation of DL model predictions based on SHAP values shows that cyclomatic complexity, Halstead effort and code churn are among the dominant features, thus proving domain understanding by the model.

6.1.2 Implications for Healthcare Software Engineering Practice

The results of this thesis work have practical implications concerning the organization of QA processes in healthcare software companies. The increased recall performance of the CNN-LSTM model compared to RF by an average of +10.3 percentage points is translated to a 71% decrease in undetected defects per release cycle in the typical healthcare codebase composed of 500 modules – a value that is linearly correlated with safety risks for patients. In this context, Class B and Class C devices, defined according to IEC 62304 standard, are directly impacted with respect to defects detection, with possible patient injuries or even fatalities due to errors in software.

The proposed framework presented in Chapter 3 serves as a means to reconcile the performance of state-of-the-art deep learning techniques with the requirements of FDA SaMD, IEC 62304, HIPAA and GDPR guidelines, which include mandatory documentation and testing processes in software lifecycle management. With the introduction of explainability and traceability requirements upfront, rather than after the process is over, deep learning models used within SDP can be effectively incorporated into healthcare software engineering practice.

6.1.3 Contributions of the Thesis Revisited

The six contributions that the thesis proposes: the systematic review, the SMOTE-CNN-LSTM framework, the multi-dataset benchmarking protocol, the regulatory compliance framework, the temporal defect pattern analysis, and the implementation blueprint, all combine to push forward the state of knowledge in the emerging area of software engineering, machine learning, and health informatics. Specifically, each contribution addresses a unique gap in the prior body of knowledge, resulting in a coherent research agenda moving from the realm of theory into concrete applications in healthcare SDP.

6.2 Future Scope

Though this thesis has achieved notable progress in advancing the application of deep learning to SDP in healthcare systems, it has also pointed toward some promising avenues for further exploration that extend from architecture innovations to expanded applications beyond healthcare and even regulation compliance, each of which represents an area of further development that could be highly impactful in improving healthcare software quality assurance.

6.2.1 Code-aware Defect Prediction Using Transformers

The most obvious and immediate area for future development is applying transformer-based architectures, such as CodeBERT, GraphCodeBERT, and UniXcoder, within the healthcare SDP process. These transformers, pre-trained on large volumes of source code using masked language modelling tasks, create contextual representations of code that can encode complex semantic relationships between tokens, methods, and classes in ways that go beyond simple software metric feature vectors employed here. Comparison of SMOTE-CodeBERT with SMOTE-CNN-LSTM on the same test conditions will confirm whether the extra representation power offered by transformer embeddings adds any more gains in recall and AUC over those achievable using the current software metric features, or if there is no more room for improvement on this front. Graph Neural Networks (GNNs) applied to **Abstract Syntax Trees (ASTs) or Program Dependency Graphs (PDGs)** is another related field of development.

6.2.2 Federated Learning for Multi-Institutional Healthcare SDP

As it stands, our methodology assumes that the data needed for training is all available centrally to the one model training environment. However, in practice, data about healthcare software is held by multiple competing institutions such as hospital chains, device manufacturers, and EHR suppliers, and it is not possible for those institutions to collaborate by sharing the actual software artifacts because of data governance issues. Instead, the use of federated learning techniques, specifically FedAvg and FedProx, can enable each competing institution to train its own instance of CNN-LSTM and only pass encryption gradient information to a central aggregator. Federated SMOTE-CNN-LSTM could thus exploit multi-institutional statistical diversity while complying with HIPAA minimum necessary and GDPR data minimization .

6.2.3 AI-SaMD Defect Prediction

FDA's evolving regulatory framework for AI/ML enabled Software as a Medical Device (AI-SaMD) has led to the development of a new category of healthcare software that includes defects that go beyond the regular coding defects. They are algorithmic bias, dataset shift, and model performance degradation, among others. The expansion of the SMOTE-CNN-LSTM framework for the prediction of defects in this new category of software will be challenging as it will need an entirely new set of features to include measures of model training stability, dataset of validation data, and model performance drift. These features cannot be included in the existing static code metrics.

6.2.4 Real-Time Just-in-Time (JIT) Defect Prediction in DevSecOps

The present application uses batch processing where models are trained periodically and then used in predicting defects in a set of modules in release versions. To make defect predictions Just-in-Time, the process of predicting defects should happen in real-time for commits rather than for releases. For example, real-time JIT prediction could be implemented such that defect predictions are generated per commit and the result is communicated to developers as soon as a commit is made before it is merged into the master branch.

The LSTM aspect of the model architecture is inherently suited for JIT inference, given its intrinsic sequential nature, but does need additional engineering to allow for sub-second inference latency, sufficient for CI/CD gate requirements.

6.2.5 Concluding Remarks

The convergence of deep learning techniques and the realm of software quality assurance in the field of healthcare is still a nascent, but quickly evolving scientific discipline. The findings provided in this thesis prove that the chasm separating potential from realization can indeed be traversed — not by sacrificing prediction efficacy, but by carefully co-engineering the design architecture, pre-processing methods, explainability tools, and documentation processes that can satisfy regulatory requirements. The SMOTE-CNN-LSTM framework itself is not meant as the ultimate solution, but rather as a validated groundwork for pursuing the future avenues for research suggested above. As the software systems in healthcare become increasingly complex, autonomous, and clinically critical, their predictive quality assurance is becoming more important than ever before, and deep learning models, when implemented responsibly, have much to offer.