AI and Software Engineering: Rapid Process Improvement through Advanced Techniques

By Meghasai Bodimani

Department of Computer Science, University of Missouri, Kansas City, USA

DOI: 10.55662/JST.2021.2101

Abstract –

In recent years, a variety of research have effectively applied machine learning approaches across a broad range of industries. This led to the creation of a large range of deep learning models, each adapted to a specific purpose in software development. There are various ways in which the software development business may benefit from employing deep learning models. Nowadays, nothing is more vital than consistently testing and maintaining software. Software engineers are responsible for a broad variety of duties during the lifespan of a software system, from original design to final delivery to clients via cloud-based platforms. It is evident from this list that all jobs involve meticulous planning and the availability of a range of materials. A developer may study a range of resources, including internal corporate resources, external websites with important programming material, and code repositories, before creating and testing a solution to the current issue. Finding out what went into building the recommended is what this inquiry is all about. Based on user feedback, this system examines the recommended's effectiveness and proposes methods to enhance the programme.

Keywords- Artificial Intelligence, Machine Learning, Privacy Mechanisms, Network

Introduction

A software engineer's tasks include developing code, testing it, delivering it to the cloud, and engaging with stakeholders via email and meetings [1]. Finding and utilizing a broad range of information and resources, as well as planning and preparing for the next action, are critical components of each of these jobs [2]. A developer might perform research by looking at other

code repositories, websites with comparable programming content, or asking colleagues for guidance before coming up with a solution and testing it [3].

People fresh to the field might feel intimidated by the notion of carrying out these activities [4]. Even for seasoned programmers, it is quite difficult to attain near-perfect performance in these tasks. In order to boost efficiency and make job execution simpler, recommended systems were invented in software engineering [5]. The word " recommendeds" is shorthand for computer systems that generate helpful data for software engineering activities [6]. Some recommendeds are really pertinent to software engineers' job, therefore they're accustomed to using them. Use of recommendeds in several IDEs, including Eclipse IDE4, may address the problem of missing declarations of imported objects in Java code [7].

Different recommendation systems have been constructed for different activities and procedures, such as code rearrangement, learning the upcoming set of instructions, and recognizing needs. An example of a productivity-enhancing tool is the Eclipse Mylyn recommendation, which delivers tailored recommendations on the source code connected with a given task. Recommendeds contain enormous unrealized potential in the software development process owing to their vast variety of capabilities [8].

A fundamental concern with the present recommended system is its inclination to anticipate things that the user would regard as irrelevant or dull. Consequently, a recommended system is essential to give services based on the similarity of items. By combining user and product data into a collaborative recommendation system, it is feasible to reliably discover customer preferences [9–11].

Building a recommended system starts with identifying the problem that needs solving and confirming the premise that the recommended may give useful ideas to the developer working on the issue. A procedure called as "framing the problem" defines what occurs at this level. In order to comprehend the issue and its solution, we may go back to the introduction's description of a software engineering recommended. Having a good knowledge of the job and situation in which a recommended will be employed is crucial when considering its construction.. Another issue to consider is the intended audience of the recommended: whether it is built for developers or end users. The idea of a task handled by a recommended applies to the particular aim of a developer at a given moment in time, such as carrying out a

defined feature in the source code. While a developer keeps ongoing awareness of the current task, it is not always directly represented in the code. The context of a recommended applies to the unique information and tool environment in which a task is carried out. This covers aspects like as the available source code and other artefact, as well as the variety of tools that may be employed to execute the work. The context additionally contains the developer's efforts in completing the stated objective. This statement describes the conditions and substance of information that a recommended may offer: folks who are inexperienced typically have differing information needs in contrast to those who are highly aware. Although the first group may benefit from frequent offers, the latter group typically has a limited tolerance for interruptions that give information they already know. The key contributions of this paper are as follows:

(a) The subject was framed as finding the inputs for creating the recommended.
(b) This system makes software development ideas based on customer satisfaction and examines the efficiency of the recommended.

Related Work

The work by Wen et al. [12] studied machine learning models in detail, giving particular attention to four important areas: the ML technique employed, estimate accuracy, model comparison, and estimating environment. They set out to explore the setting of estimation as their major research objective. All empirical papers published on the machine learning model between 1991 and 2010 were rigorously evaluated. Models for software defect prediction (SDEE) incorporate eight distinct kinds of machine learning (ML) methodologies, as found following a thorough analysis of eighty-four main sources. When comparing machine learning models to non-ML models, the former frequently displays greater and virtually equivalent forecasting accuracy. This rationale explains why certain ML models perform better than others in specific estimate scenarios. Machine learning models offer considerable potential in SDEE. However, ML model application in the sector is still restricted, demanding more efforts and financial incentives. The author gives ideas for future research and guidance for existing practitioners based on the outcomes of this assessment.

In light of the increased popularity of this method, Wan et al. [13] set out to examine how machine learning influences software development processes. After polling 342 individuals from 26 countries across 4 continents and interviewing 14 people, we discovered that machine learning systems have advanced significantly more slowly than non-machine learning systems. Our study demonstrates that the two groups differ greatly in terms of requirements, design, testing, and approach within software engineering, as well as in terms of job characteristics. Skills, problem-solving skills, and the capacity to recognise activities are some of these differences. The author made clear ideas and suggested possible future study areas based on our findings.

Del Carpio and Angarita [14] obtained promising achievements by using machine learning algorithms to a range of categories of knowledge. There is optimism for the software industry's future systematic exploration of deep learning model-assisted software processes, given various models are currently focused on diverse software procedures. Subprocesses involved in software testing and maintenance were the principal focus of the research. Subprocesses that deal with bug reporting, malware categorization, and recommendation creation commonly employ deep learning models like CNN and RNN. When it comes to testing and maintenance, there are a number of ways to prioritise tasks like operate estimation, software requirement categorization, graphical feature recognition, code author identification, source code similarity commitment, defect forecasting and categorization, and analysis of bug reports.

A lot of people are interested in AI today because of how it can automate hard or tiresome work, argue Meziane and Vadera [15]. No software engineering projects have departed from this criterion. The issues of AI and software maintenance are treated thoroughly in this thesis. Additionally, a detailed mapping investigation was done to evaluate the existing status of artificial intelligence as it applies to software maintenance activities. The study's four basic points were the following: the nature of the research, its influence on the field, the software maintenance domains, and the AI solution type.

In their systematic review and meta-analysis, Barenkamp et al. [16] engaged five software engineers for qualitative interviews. The outcomes of the study are classed according to numerous stages of producing software. Algorithms that automate arduous and repetitive software testing and development activities, like bug finding and documentation, represent a

huge breakthrough in artificial intelligence. Artificial intelligence also makes it feasible to methodically evaluate massive datasets in search of patterns and fresh information clusters. In addition, AI makes it easy to apply neural networks for systematic investigation of these datasets. The usage of AI shortens development times, decreases expenditures, and enhances efficiency. When compared to contemporary AI systems, software engineering automation is light years ahead. These systems depend on frameworks that people have established and generally try to duplicate current knowledge. Developers' inventiveness could be aided by AI technology.

According to Harman, software engineering approaches that incorporate artificial intelligence (AI) also deal with difficulties linked to software development [17]. The history of applying probabilistic reasoning and machine learning to software engineering offers a strong framework for search-based software engineering, despite the fact that it is a relatively new field. According to the author's research of the relationships between the two disciplines, there are a number of shared aspects between the two.

Software quality models were compared by Tate (18). To examine the present methodologies, case studies employ software quality models. The outcomes of the case study increase the evaluation of the empirical model. Predetermined selection criteria are utilised for both the suggestion and selection of models. Success criteria are developed in advance and used to assess approaches. Process models are appraised utilizing theoretical assessment approaches. We assess the model's relevance to software participants and how well it meets the standards of an ideal process quality model. To test the depth and breadth of a model's performance in actual software operations, empirical evaluation methodologies are utilised. There are approaches to figure out which process quality model to apply and whether they all produce different outcomes. We check for differences in the software techniques employed in the case studies.

Research by Fadhil et al. [11] looks at how AI may enhance strategies for anticipating and identifying software faults. Statistics demonstrate that AI has helped uncover software vulnerabilities and forecast when faults may occur. The application of AI in software engineering increases software quality by minimising excessive expenditure and generating optimum solutions.

Concerning these measures, Kothawar and Vajrapu [19] detailed the difficulties and solutions. Method: Fifteen case studies showcasing successful tactics were picked from eight different firms, all of which had distinct obstacles and found inventive ways to solve them. The findings of our study reveal that startups prioritise activities in various ways. Formal procedures were adopted by six of the eight organisations, but unstructured prioritisation was relied upon by the other two. Making priorities according to consumer feedback and return on investment (ROI) is the key to finding out how much firms are worth.

The key demands and obstacles experienced by startups are discussed in this research. The results of the research are supported by the literature. Finding answers helps specialists in their area. Software businesses located in Sweden should be included in the ballot. Practitioners wishing to build a software business and focus needs may also profit from some of these approaches.

This work's aggregation approach is clear, useful, and easy to grasp [9]. This approach assures that software quality assessments are trustworthy and can be repeated with the use of metrics and models. All apparent software artefacts are utilised to give probability to good and low quality. There were theoretical and empirical components to the validation approach. Evaluations of information quality, maintainability, and bug prediction were carried out. Through the use of software visualisation, we examined the performance of numerous aggregation techniques and the utility of aggregation for multivariate data. The author finished by examining the utility of MCR and applying it to real alternatives. A benchmark that handles regression concerns was produced by the author utilizing machine learning approaches. After that, they compared the aggregated result to a ground truth and made sure it truly mirrored the input data. Our system enables us to make assessments based on various factors, and it is both precise and responsive. Without utilizing any reference data, our suggested algorithm may act as an unbiased unsupervised forecaster.

Sentiment analysis on social media sites like Facebook and Twitter has lately increased in popularity as a valuable tool for grasping people's thoughts and sentiments. On the other hand, difficulties linked to NLP are confronted by sentiment analysis. Recently, deep learning models have demonstrated to be a great way for dealing with challenges in NLP. The most current work addressing challenges in sentiment analysis, particularly sentiment polarity, is covered in the study [10] via use of deep learning. Using word embedding and the TF-IDF

model, several datasets have been studied. Scientists have also compared the results of studies using alternative models and input factors.

In order to prioritise testing and locate problematic areas of code more readily, software defect prediction is an approach applied. Previous research relied on machine learning approaches to construct reliable prediction models, with an emphasis on manually encoding programme data. When it comes to constructing reliable prediction models, classical attributes fall short because they don't take semantic variations across programmes into account [8]. Integrating programme semantics with fault prediction features is the core objective of deep learning. Using Abstract Syntax Tree (AST) token vectors, the deep belief network (DBN) learns semantic features on its own. We demonstrate that our algorithmically acquired semantic characteristics considerably enhance fault prediction both within and across projects, using 10 open-source projects as examples. This surpasses the usefulness of standard features. As a consequence, the WPDP's F1 score rises by 14.2%, recall improves by 11.5%, and accuracy increases by 14.7%. In F1 for CPDP, our semantic feature-based method provides an 8.9% improvement over TCA+.

The approach LEMNA, which the authors [20] described, precisely and comprehensively describes security measures. In order to clarify the categorization of an input sample, LEMNA creates a restricted collection of features. Building a simple, intelligible model that can reach near to deep learning's decision boundary is the aim. The system leverages nonlinear local constraints to boost explanation accuracy and effectively controls feature dependence to allow interaction with security applications like binary code analysis. Two deep learning security applications—a malware classifier and a function start detector for binary reverse engineering—were utilised to examine our technique. Evidence from rigorous testing confirms LEMNA's hypothesis as the most accurate explanation. By utilizing it to assess model behaviour, solve classification issues, and automatically remedy target model faults, the author illustrates how LEMNA may aid machine learning developers.

Reference [7] looked at software project management-related machine learning literature. Scholarly papers addressing areas like methodology, software project management, and machine learning may be discovered in venues like Web Science, Science Direct, and IEEE Explore. In total, 111 articles are scattered out across three separate collections. Management of software projects is the topic of the first series of articles. Machine learning methodologies

and techniques employed in projects make up the second category. The third category comprises of research that have explored the different stages and evaluations of machine learning management, along with the findings of these investigations. Research on the impacts and development of machine learning project forecasting is also a component of it. As a consequence, attempts to control project risks in the future will have a better basis. Maximising the growth output ratio, minimising the likelihood of failure, enhancing project outcomes, and avoiding losses are all potential effects of employing machine learning for project risk assessment.

Recent breakthroughs in machine learning have generated interest in discovering methods to incorporate AI into information technology applications and services. Many firms modified their methods to development in order to accomplish this aim. The author explains the outcomes concerning the Microsoft AI app development teams. Search and natural language processing are two areas where the platform is aimed to assist the creation of artificial intelligence applications. R and Python, two data science tools, are utilised to achieve this. Bug reporting systems and diagnostic tools are two examples of potential usage. Research mentioned in [5] suggests that this workflow has been adopted into the software engineering processes of multiple Microsoft teams. As a consequence of this integration, more light has been thrown on a number of important technical barriers that firms may encounter when mass-producing AI goods for sale. Because of these issues, Microsoft's best practices had to be employed. Furthermore, the author identified three significant contrasts in AI: (1) software development teams sometimes lack the specialist expertise required for model change and reuse. The second difficulty is that, in compared to standard software components, AI components are more difficult to handle as standalone modules. Vital information was communicated via Microsoft Teams.

"Deep neural networks" (DNNs) were initially suggested by Yang et al. [6] and an upgraded model training technique was also offered. Alpha Go exhibited deep learning's remarkable skills in 2016. Experts in software engineering (SE) may construct cutting-edge research tools with the aid of deep learning. Software engineering (SE) deep neural networks' (DNNs') performance is impacted by tuning, internal structure, and model selection. Deep learning's potential applications in software engineering have gotten little academic attention. The author looked extensively for relevant studies published as far back as 2006. First, the notion

of deep learning is presented in the context of software engineering. A number of SE's deep learning techniques have been categorised. While researching possible uses of deep neural networks (DNNs) in software engineering, the author looked at strategies for improving deep learning models. Our investigation's results shed light on critical difficulties and guide the way towards possible future research topics.

More and more, machine learning is being utilised by software developers to make current software smart and able to learn on its own. Researchers in the area of software development are increasingly looking at methods to integrate machine learning into different phases of the SDLC. Presented here are the findings of an analysis on the application of machine learning across the software development life cycle. Overall, the purpose of [3] was to obtain a full grasp of how various phases of the software development life cycle interact with different kinds of machine learning tools, processes, and approaches. We undertake a thorough investigation to address the issue of whether machine learning is biassed towards specific phases or methodologies.

The usage of recommendation systems is becoming increasingly vital for commercial transactions, revenue, and general success. This research focuses on recommendation systems and their implementation approaches. The contents and qualities of a recommended system may vary based on the needs of the company. This article explains the design concepts and basic features of recommended systems. Several notable approaches are open to examination. To recap, [4] provided movie recommendation algorithms from the three primary sectors: cinema, music, and e-commerce. The survey seeks to offer readers a full grasp of the scenarios in which particular recommended systems are acceptable.

Data scientists typically construct machine learning models to handle diverse difficulties in both industry and academia; nonetheless, these models have their own hurdles. An challenge in machine learning development is the lack of understanding among practitioners about the possible benefits of following to the methods outlined in the software engineering development lifecycle (SEDL). Indeed, owing to the intrinsic distinctions between machine learning systems and normal software systems, there will necessarily be some quirks in the development process. In the context of software engineering, the purpose of [2] was to evaluate the issues and techniques that arise when constructing models, with an emphasis on

how developers may receive benefits from adopting or adapting the standard workflow for machine learning.

Deep learning has been recently employed in the area of software engineering (SE). There are still unsolved questions. Li et al. [1] analysed 98 software engineering articles that employ deep learning to solve these issues. Deep learning approaches have simplified 41 software engineering jobs across all phases. Deep learning models and their numerous variations are employed to handle 84.7% of software engineering challenges in academic articles. The practicality of deep learning is being put into doubt. In the future, there may be a bigger number of software engineering researchers who are interested in upgrading deep learning-based solutions.

Approach

Within this portion, we have provided a new framework of Long Short-Term Memory (LSTM) that is capable of delivering suggestions for software development features. These suggestions are produced from the dataset of consumers. Figure 1 displays the recommended framework procedure of the current investigation:

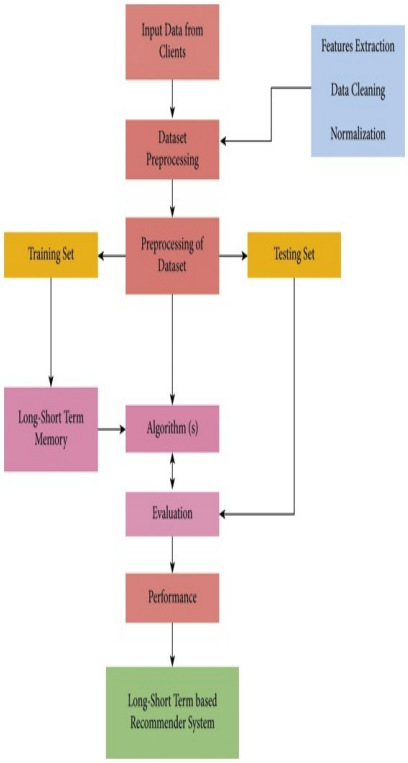


Figure 1

The proposed framework workflow.

Description Of Raw Data

An Excel-based synthetic dataset, hand-picked from a dataset obtained using real Business Intelligence tools, is used in this research. The dataset has eleven characteristics and one hundred rows; one of those rows reflects the program's rating. If a programming gets a 3 or above, it's highly recommended. If it is so, the proposed model is useless. Table 1 provides a description and explanation of the attributes of the dataset.

Figure 3 shows the feature business scale distribution for several deployment methods and operating systems, including on-premise, hybrid, and cloud. You can also see the prices for the open source, corporate, and freemium editions there.

.

Table 1

Dataset description and feature explanation.

Features	Description	Variables type
Category	Category comprises the type of BI tool, as well as the industry in which it can be used	Input variable
Business scale	This identifies the size of the company that the BI tool is designed to serve, such as small, medium, or large	Input variable
User type	Whether they are a business user or an analyst with data science skills, this indicates the sort of user (not all BI tools are easy to use and not all tools possess powerful data processing capabilities)	Input variable
No of users	This offers information on how the BI tool may be implemented, such as cloud, on-premise, or hybrid OS: this specifies the type of operating system necessary for the tool installation	Input variable
Pricing	Pricing: this reveals whether the program has a freemium version or an enterprise edition for data visualization on mobile devices	Input variable
Ratings	On a scale of 5.0, users rate this product	Output variable

Dataset samples from the acquired dataset.

Category	Industry	Business scale	User type	No. of users	Deployment	OS	Mobile apps	Pri
100001	Data management	Utilities	Large	Business	Single	Cloud	Linux	
100002	Database/ERP	Food	Large	Business	Single	Premise	Mac	
100003	Data analysis	Manufacturing	Large	Business	Single	Premise	Linux	
100004	Data analysis	IT	Medium	Business	Multiple	Premise	Mac	
100005	Benchmarking	Food	Medium	Analyst	Multiple	Cloud	WIN	

Processing Of Raw Data

Data purification was accomplished after raw data collection by employing a number of methods, one of which was eliminating null values and duplicate entries. This is how data mining converts raw, unstructured information into a form that can be easily studied; this is helpful because raw, unstructured information from the real world is rare and inconsistent. Classification machine learning approaches typically display equal frequencies in each class, but prediction models become complicated when categories aren't evenly distributed. Following this, resampling methods achieved remarkable progress, enabling us to remove entries from every cluster while preventing undersampling and keeping records from the majority class [21].

When doing data mining research, it is crucial to ensure that the dataset is balanced and consistent. Finding dataset outliers is a doable task. Dataset outliers are values that deviate significantly from the norm and stand out from the rest due to their extreme uniqueness.

Outliers might be caused by reading mistakes, device malfunctions, or human error. Prior to conducting any statistical analysis or study, it needs to be extracted from the dataset. Outlier data poses a risk of inadequate or incorrect conclusions, which could affect the analysis and subsequent treatment [22, 23]..

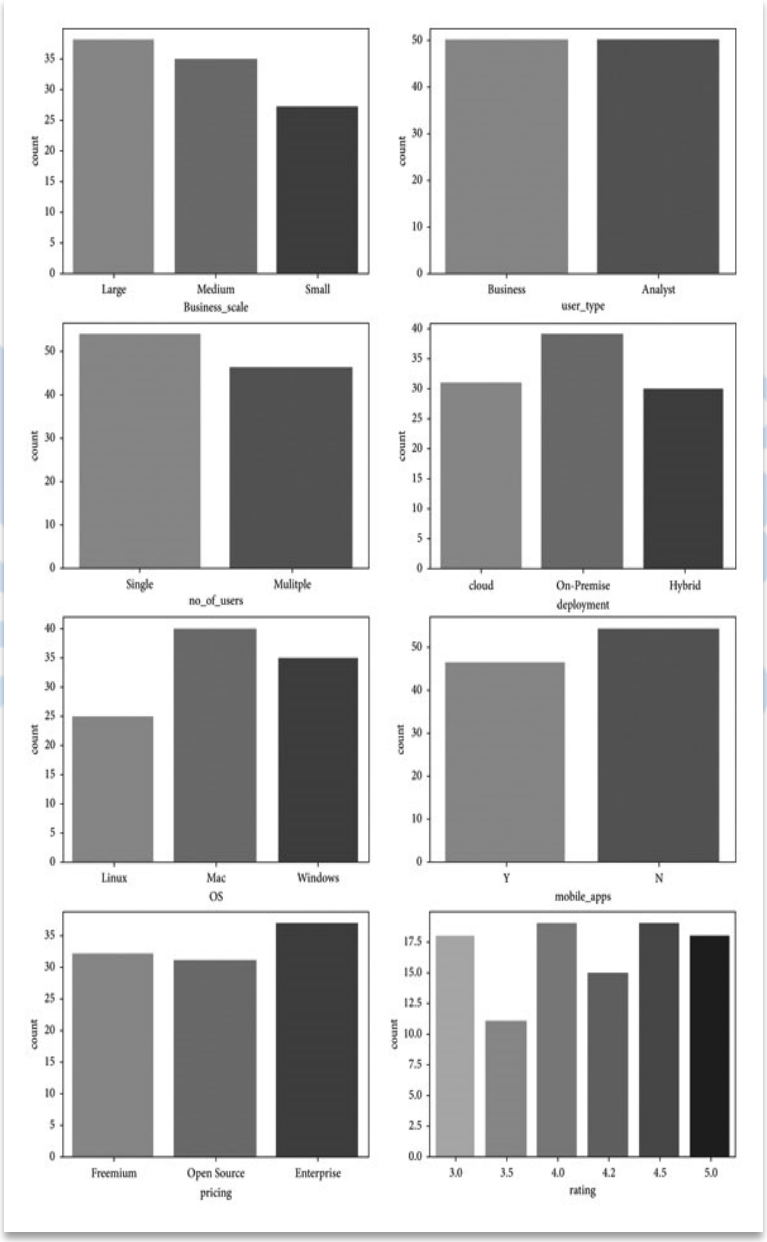
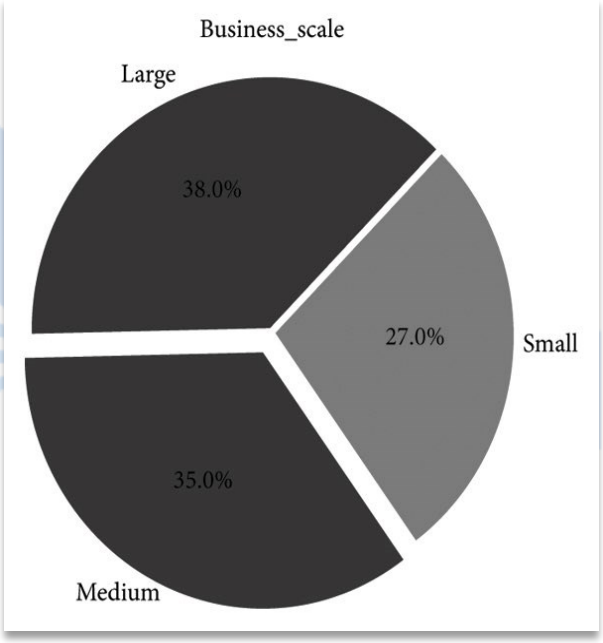


Figure 2

Data visualisation and attribute distribution of frequency.

Enhancing Features

These capabilities might be exploited by learning machines given the correct domain data. Machine learning representations cannot be created from raw data without human intervention. Researchers utilised correlation matrices to determine the interrelationships of the variables. Matrixes for correlation and covariance are identical. The correlation could be used to determine the degree of a linear relationship, among other potential uses. One way to measure the direction and strength of a linear relationship between the two numbers is through correlation, which is a statistical tool. There is a wide range of possible values for r, from -1 to +1.



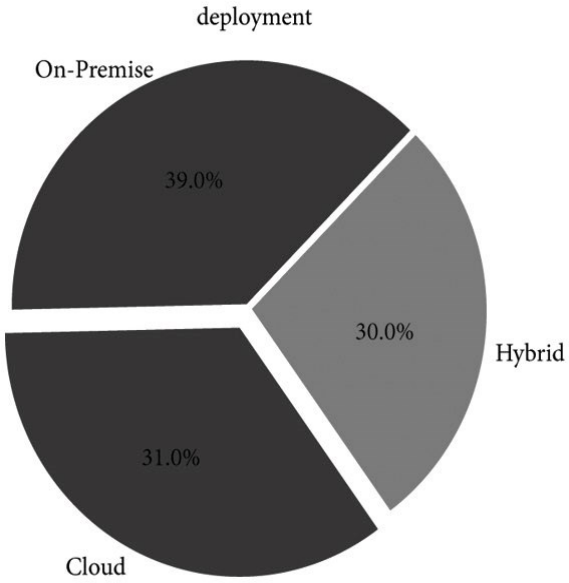


Figure 3

Dispersal of features: (a) enterprise-level, (b) operational system, (c) cost.

Model Proposal

Figure 4 of the model shows that the fulfilled layer is in charge of extracting input sequence features and embedding them. The Keras-Tuner module's hyperband optimisation strategy is one approach to optimising TensorFlow model hyperparameters. Maybe just a few lines of code will be enough to accomplish this. It is standard practice to use a validation dataset that is randomly sampled 10% of the training data when adjusting hyperparameters. Finally, we used sparse categorical accuracy to rank the optimisation experiments. After experimenting with various parameters, the batch size was finally settled on as 512. Using a combination of hyperparameters that optimise its performance to the maximum, the final model is trained with data acquired from earlier optimisation processes. To determine how well our newly designed suggested system performed in contrast to IRnoD, we employed a back-testing strategy.

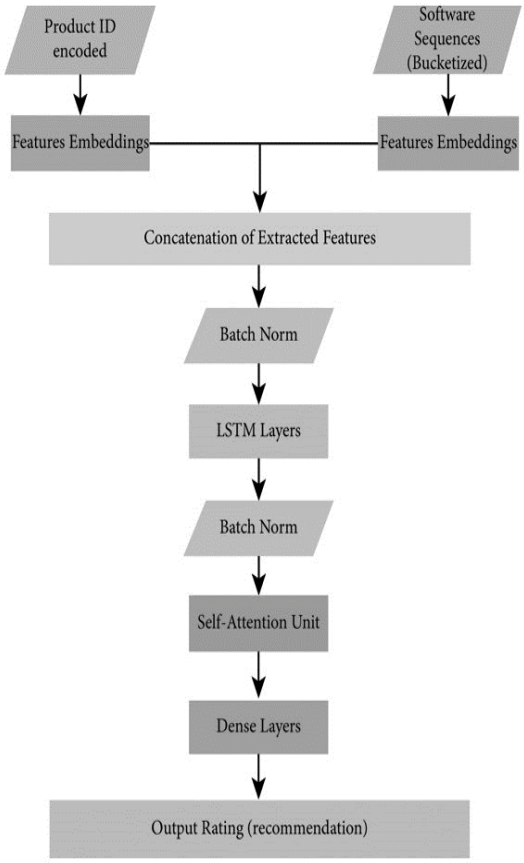


Figure 4

Proposed model architecture.

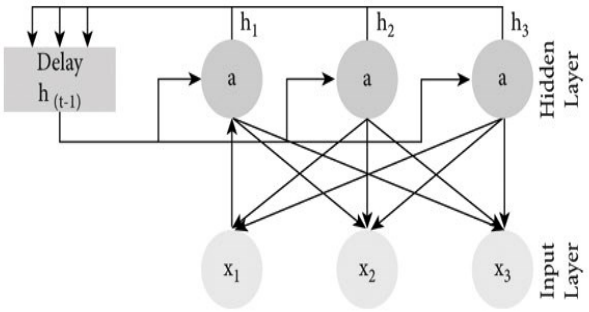


Figure 5

Recurrent nodes of modified LSTM.

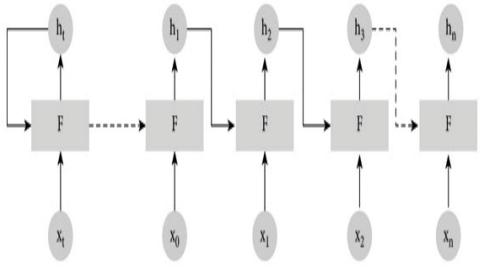


Figure 6

Unrolled nodes.

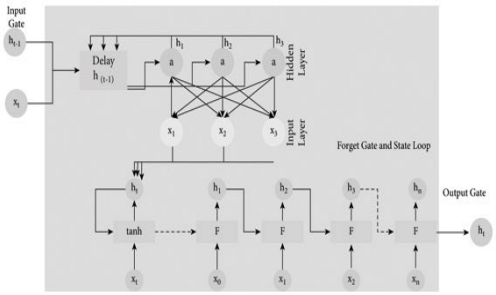


Figure 7

Modified cell of LSTM

New Long Short-Term Memory (LSTM) Cell

Long short-term memory (LSTM) networks are a subgroup of RNNs. Many aspects of human behaviour are temporally or sequentially dependent; for instance, language, market pricing, and electricity demand. Such events are what recurrent neural networks aim to model and represent. One way to do this is to use the input from a neural network layer at time t+1 and feed it the output from the same layer at time t. Figure 5 shows that the updated LSTM contains different recurrent units:

Figure 6 demonstrates that the data is regularly delivered to the network and contains the previous output, F ($h t−1$).

The conventional neural network layers are replaced by LSTM cell blocks in long short-term memory (LSTM) networks. The input, forget, and output gates—which will be examined more later on—are part of these cells. Our proposed LSTM cell is shown visually in Figure 7.

Findings And Analysis

For this method's testing, we consulted the Steam project's data. Currently, we do not have any datasets available that would be suitable for testing our technique. For testing reasons, we used the most current information as our test set and the remaining data as our training sets.

Table 4

The suggested LSTM's recall rates using various methods.

Approach	Recall rate
IR	0.0824
IF	0.064
TD	0.014

A serial filling strategy was used to carry out dimensionality reduction in the experiment, with a target dimension of k=50 and a time series length of T=12. At the close of business, we sent a catalogue including the fifty most sought-after items (N = 50) to every user. We conducted two independent control tests to evaluate the overall efficacy of our method. Neither dimensionality reduction (DR) nor serial filling (SF) were used in a single experiment..

Table 5

How long does the suggested LSTM take using various methods?.

Approach	Training time (seconds)	Testing time (seconds)
IR	1500	16
IF	1250	13

The collaborative filtering methods for temporal decay (TD) and implicit feedback (IF) were used as a baseline against which our solution was benchmarked. The recall rate allowed us to measure the efficacy of our suggestions. We measured the system's efficiency by looking at its training and execution timings. Finally, we had to see whether the average amount of time it took each software and approach to make a proposal varied significantly..

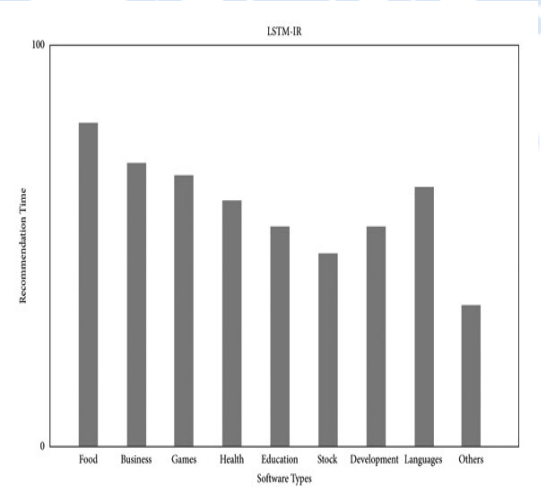


Figure 8

The LSTM with IR is suggested.

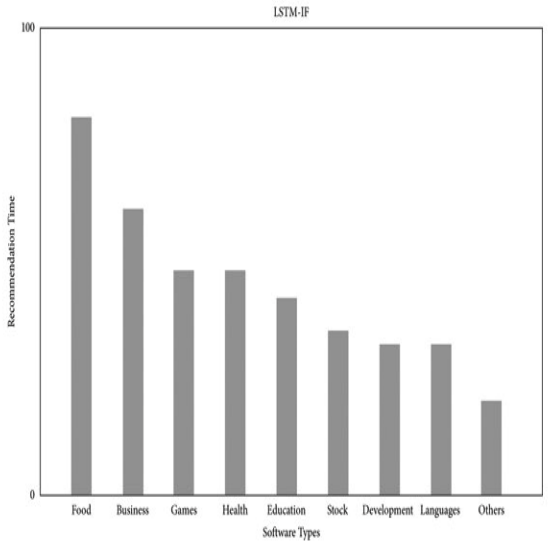


Figure 9

Recommendation of LSTM with IF.

Table 4 displays the recall rates for various techniques. Results revealed that IR and serial filling outperformed the baseline treatments in terms of memory. Table 5 displays the total amount of time spent in IR. Since matrix factorization yields a recall rate for information retrieval (IR) that is comparable, it is an excellent technique for minimising a system's dimensionality.

The IR data is shown in Figure 8, and the distribution of the most recommended software's interface (IF) at various eras is displayed in Figure 9. From the fact that our information retrieval (IR) technique recommends a greater diversity of items than baseline collaborative filtering, it follows that our strategy is more varied.

Conclusion

A long short-term memory (LSTM) recommendation model for interaction records was suggested in this research. Based on our evaluations, our model performed very well in all three areas: precision, efficiency, and diversity. Our next step is to experiment with other datasets to see whether our method works on a wider scale. Additionally, as a measure of

quality in this study, we included the total amount of time spent communicating with others. With so many different viewpoints represented and so many different items being appraised, the possibility of bias in ratings is high. The takeaway here is that going forward, we need to pay close attention to making our rating vectors better. We will investigate a wide variety of other approaches and models to deal with time series.

References

1. X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, "Deep learning in software engineering," 2018, https://arxiv.org/ftp/arxiv/papers/1805/1805.04825.pdf.
2. G. Lorenzoni, P. Alencar, N. Nascimento, and D. Cowan, "Machine learning model development from a software engineering perspective: a systematic literature review," 2021, https://arxiv.org/abs/2102.07574.
3. Pargaonkar, Shravan. "Bridging the Gap: Methodological Insights from Cognitive Science for Enhanced Requirement Gathering." Journal of Science & Technology 1.1 (2020): 61-66.
4. Pargaonkar, Shravan. "A Review of Software Quality Models: A Comprehensive Analysis." Journal of Science & Technology 1.1 (2020): 40-53.
5. S. Shafiq, A. Mashkoor, C. Mayr-Dorn, and A. Egyed, "A literature review of using machine learning in software development life cycle stages," IEEE Access, vol. 9, pp. 140896–140920, 2021.
6. N. Koneru, S. Rai, S. S. kumar, and S. Koppu, "Deep learning-based automated recommendation systems: a systematic review and trends," Turkish Journal of Computer Mathematics Education, vol. 12, no. 6, pp. 3326–3345, 2021.
7. S. Amershi, A. Begel, C. Bird et al., "Software engineering for machine learning: a case study," in Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300, Montreal, QC, Canada, May 2019.
8. Pargaonkar, S. (2020). A Review of Software Quality Models: A Comprehensive Analysis. Journal of Science & Technology, 1(1), 40-53.
9. Y. Yang, X. Xia, D. Lo, and J. Grundy, "A survey on deep learning for software engineering," ACM Computing Surveys, vol. 54, no. 10, 2022

10. Pargaonkar, Shravan. "Bridging the Gap: Methodological Insights from Cognitive Science for Enhanced Requirement Gathering." Journal of Science & Technology 1.1 (2020): 61-66.
11. M. Z. M. Hazil, M. N. Mahdi, M. S. Mohd Azmi, L. K. Cheng, A. Yusof, and A. R. Ahmad, "Software project management using machine learning technique - a review," in Proceedings of the 2020 8th International Conference on Information Technology and Multimedia (ICIMU), pp. 363–370, Selangor, Malaysia, August 2020.
12. S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," Proceedings of the 38th International Conference on Software Engineering, vol. 14-22, pp. 297–308, 2016.
13. Pargaonkar, Shravan. "Future Directions and Concluding Remarks Navigating the Horizon of Software Quality Engineering." Journal of Science & Technology 1.1 (2020): 67-81.
14. M. Ulan, Aggregation as Unsupervised Learning in Software Engineering and Beyond, Linnaeus University Press, Cambridge, MA, USA, 2021.
15. N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: a comparative study," Electronics, vol. 9, pp. 483–3, 2020.
16. J. A. Fadhil, K. T. Wei, and K. S. Na, "Artificial intelligence for software engineering: an initial review on software bug detection and prediction," Journal of Computer Science, vol. 16, no. 12, pp. 1709–1717, 2020.
17. Pargaonkar, S. (2020). Future Directions and Concluding Remarks Navigating the Horizon of Software Quality Engineering. Journal of Science & Technology, 1(1), 67-81.
18. J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," Information and Software Technology, vol. 54, no. 1, pp. 41–59, 2012.
19. Raparthi, M., Dodda, S. B., & Maruthi, S. (2020). Examining the use of Artificial Intelligence to Enhance Security Measures in Computer Hardware, including the Detection of Hardware-based Vulnerabilities and Attacks. European Economic Letters (EEL), 10(1).

20. Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" IEEE Transactions on Software Engineering, vol. 47, no. 9, pp. 1–1871, 2020.
21. Raparthi, Mohan, Sarath Babu Dodda, and SriHari Maruthi. "Examining the use of Artificial Intelligence to Enhance Security Measures in Computer Hardware, including the Detection of Hardware-based Vulnerabilities and Attacks." European Economic Letters (EEL) 10.1 (2020).
22. F. Del Carpio and L. B. Angarita, "Trends in software engineering processes using deep learning: a systematic literature review," in Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 445–454, Kranj, Slovenia, August 2020.
23. F. Meziane and S. Vadera, Artificial Intelligence in Software Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, 2010.
24. M. Barenkamp, J. Rebstadt, and O. Thomas, "Applications of AI in classical software engineering," AI Perspect, vol. 2, no. 1, pp. 1–15, 2020.
25. M. Harman, "The role of artificial intelligence in software engineering," in Proceedings of the 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE), pp. 1–6, Zurich, Switzerland, June 2012.
26. J. Tate, Software Process Quality Models: A Comparative Evaluation, Citeseerx, Pennslyvennia, PA, USA, 2003.
27. S. Kothawar and R. G. Vajrapu, "Software requirements prioritization practices in software start-ups: a qualitative research based on start-ups in India," vol. 57, 2018.
28. W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemma," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 364–379, Toronto, Canada, October 2018.
29. M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "CorrAUC: a malicious bot-IoT traffic detection method in IoT network using machine learning techniques," IEEE Internet of Things Journal, vol. 8, no. 5, pp. 3242–3254, 2021.
30. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "IoT malicious traffic identification using wrapper-based feature selection mechanisms," Computers & Security, vol. 94, Article ID 101863, 2020.

31. Shafiq, Z. Tian, A. K. Bashir, A. Jolfaei, and X. Yu, "Data mining and machine learning methods for sustainable smart cities traffic classification: a survey," Sustainable Cities and Society, vol. 60