Simplify: A Smart Intelligent System that can code like a human being

submitted in partial fulfillment of the requirement
for the award of the Degree of

Bachelor of Technology
in
Computer Engineering

by

Rishabh Jain (2019130024)
Vedant Jolly (2019130026)
Jaiwin Shah (2019130058)

under the guidance of

Dr. Dhananjay Kalbande

Department of Computer Engineering
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri-West, Mumbai-400058
University of Mumbai
April 2023

Certificate

This is to certify that the Project entitled "Simplify: A Smart Intelligent System that can code like a human being" has been completed to our satisfaction by Mr. Rishabh Jain, Mr. Vedant Jolly and Mr. Jaiwin Shah under the guidance of Dr. Dhananjay Kalbande for the award of Degree of Bachelor of Technology in Computer Engineering from University of Mumbai.

Certified by

Dr. Dhananjay Kalbande
Project Guide

Dr. P.B. Bhavatankar
Head of Department

Dr. Bhalchandra Chaudhari
Principal

Department of Computer Engineering
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri(W), Mumbai-400058
University of Mumbai
April 2023

Project Approval Certificate

This is to certify that the Project entitled "Simplify: A Smart Intelligent System that can code like a human being" by Mr. Rishabh Jain, Mr. Vedant Jolly and Mr. Jaiwin Shah is found to be satisfactory and is approved for the award of Degree of Bachelor of Technology in Computer Engineering from University of Mumbai.

External Examiner **Internal Examiner**

(signature) **(signature)**

Name: **Name:**

Date: **Date:**

Seal of the Institute

Statement by the Candidates

We wish to state that the work embodied in this thesis titled "Simplify: A Smart Intelligent System that can code like a human being" forms our own contribution to the work carried out under the guidance of Dr. Dhananjay Kalbande at the Sardar Patel Institute of Technology. We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission.

Name and Signature:
1. Rishabh Jain
2. Vedant Jolly
3. Jaiwin Shah

Acknowledgments

It is really a pleasure to acknowledge the help and support that has gone to in making this thesis. I express my sincere gratitude to my guide Dr. Dhananjay Kalbande for his invaluable guidance. Without his encouragement, this work would not be a reality. With the guidance and freedom he provided, we really enjoyed working under him.

We thank our examiner, for his word of advice and for helping us guide in the right direction for our project.

We thank our HOD and the staff of the Computer Engineering Department for giving us all the facilities required to carry out this research work.

We would like to thank all our family members and well wishers for their constant encouragement for all these years, without which we could not have completed this work.

Contents

List of Figures

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BP	Back Propagation
CNN	Convolutional Neural Network
CRON	Command Run In (UNIX scheduler)
DL	Deep Learning
DNN	Deep Neural Network
JSON	Javascript Object Notation
ML	Machine Learning
MLP	Multi-Layer Perceptron
SVM	Support Vector Machine
XGB	Extreme Gradient Boosting

Abstract

According to recent studies, a large number of data scientists spend most of their time on tasks like data cleaning and organizing data. They need to memorize big complex syntaxes for all the major tasks in the data science life cycle. Often these tasks are redundant. Therefore, we propose to build an intelligent system that enables data scientists to perform all the tedious and time-consuming tasks such as EDA, data cleansing, data preprocessing, data visualization, modeling, and data science lifecycle evaluation. Just state the logic of your query in natural language the system will automatically output all relevant Python code snippets. Existing applications involving the text-to-code generation and code search are limited and a lot of them do not work in non-ideal conditions. The reason behind it is the data set on which the existing models have been built. These datasets do not consider real-world factors such as slang, acronyms, and paraphrases. Therefore, a new dataset was created consisting of real-world user queries, representing the scenarios a user is most likely to face daily. We plan to build a logic-oriented system that only needs to convey the logic correctly in text in natural language. It saves a lot of time, allowing data scientists to spend most of their time building logic instead of focusing on code.

Chapter 1

Introduction

Developers always ask how to translate ideas into code. For example, saving keystrokes or avoiding writing boring chunks of code. It is also useful for non-programmers and practitioners in other fields who need calculations in their daily work to create data manipulation scripts.

Automating even small parts of software development is an active research area with several approaches and proposed methods. A machine learning model that allows developers to automate coding tasks by allowing them to create individual functional units. Instead of generating code or completing developer-written code, we can restate the problem by looking for relevant snippets that already exist.

Classical pre-trained word embeddings prove very useful for his NLP tasks downstream. This is due to the ability to use information gleaned from unlabeled data to support learning and generalization, and the relative ease of including them in any learning approach. A common example is that many current approaches combine traditional word embeddings with character-level features trained on task data. To achieve this, we use a hierarchical learning architecture in which the output states of the CNN or RNN are chained to the output of the embedding layer at the character level. These methods have proven to be very powerful, especially when combined with traditional word embeddings, but in such methods, the output state of the trained language model (LM) is concatenated to the output of the embedding layer. Requires an architecture that supports and increases architectural complexity.

This system clears user entries first. This is passed simultaneously for intent classification and entity recognition using multiprocessing and based on the classified intents and entities, the relevant code is pulled from the data set and displayed to the user.

This article is organized as follows: Starting with Section II, this section provided a brief overview of the literature review, and Section III provided a comparison of various technical papers. And the final section, Section IV, summarizes the merits and limitations of this study, with a brief reference to future research.

1.1 Motivation

A user-friendly developer helper that can provide quick and accurate assistance to developers, saving their time and effort. It can help with tasks such as code generation, code suggestions, debugging, and documentation retrieval, allowing developers to be more productive and efficient in their coding tasks.

A user-friendly tool that can provide real-time assistance to developers, helping them quickly overcome coding challenges and roadblocks. It can provide instant feedback and suggestions, helping developers make informed decisions and correct errors in their code more effectively. The code helper can generate code snippets or templates based on user input, making it easier for developers to quickly scaffold code for common tasks or patterns.

1.2 Objectives

- To increase human capability and productivity which will help data scientists to focus more on the logic-building part of the project which will be of significant help.
- Custom-made dataset consisting of real-world user queries, their respective intents, entities, and corresponding Python code snippets.
- Workflow automation: Workflow automation aims to streamline and optimize code by automating repetitive tasks, reducing manual effort, and improving overall efficiency. By automating the workflow, developers can achieve several benefits and reduce execution time and improve overall efficiency.

1.3 Problem Statement

For a data science application, we suggest creating an intelligent system that can generate code like a human. Our goal is to develop a logic-oriented system that accurately interprets reasoning expressed in plain language by the user. The expected time savings are significant, allowing data scientists and developers to focus more on logic and less on coding. This system has the potential to greatly increase productivity in data science tasks.

1.4 Contributions

The project Simplify, which aims to create a smart intelligent system that can code like a human being, represents a significant step forward in the field of artificial intelligence. As a language model, I have contributed to this project by providing insights into natural language processing, machine learning algorithms, and other related fields. Additionally, the project has benefited from the contributions of developers, engineers, and researchers who have brought their unique perspectives and expertise to the table. Together, we have worked to create a system that can understand the nuances of human language and translate that understanding into functional code. Through this collaboration, we hope to make programming more accessible and efficient for people around the world.

1.5 Layout of the Report

A brief chapter by chapter overview is presented here.
Chapter 2: A literature review of different methods for face recognition, detection, comparison and flow of algorithms is presented in this chapter. Also the gaps identified through these literature surveys are explained in this chapter.
Chapter 3: The overall flow diagram of our proposed system is shown and explained in this chapter. The overall tech-stacks used and the dataset used for the application is also mentioned in this chapter.
Chapter 4: In this chapter, the implementation of our application is explained - The most essential information on user login/registration with authentication part and photo clicking and comparison part are presented.
Chapter 5: In this chapter, the result of our application is explained - The accuracy count of different algorithms for face recognition and their output are discussed. Th results obtained on different test-cases are also presented in this chapter.
Chapter 6: The overall conclusion of what the project can be really helpful in certain situations, all the overall work done in the previous phases with obtained results and insight for valuable feedbacks are discussed here.
Chapter 7: The future scope of our project, what more advance and detailed work can be done to further enhance the application, is discussed here.

Chapter 2

Literature Survey

To establish the study's parameters and frame the research issue, a preliminary literature review was carried out. Other studies have interpreted regulations outside of the fields of architecture, engineering, and construction, and his article on the use of NLP for the semantic interpretation of building regulations is one of only eight. Some studies have concentrated on automated conformance checking and the presentation of regulations. These articles demonstrated how building code conversion may be broken down into a number of distinct processes. 1. In what ways might NLP technology facilitate or streamline the understanding of construction regulations?

This thorough literature search turned up 41 papers on natural language processing that dealt with how to read building codes. The 1,962 records that were gathered from six databases and the extra candidate articles found using the reverse snowball and author search procedures comprised the selection of these articles. The papers were then grouped, distilled, and examined. Eight research gaps were eventually discovered, and suggestions were offered on how to close them. To close that gap, machine learning has been studied, but these investigations have not yet matched the strength of rule-based methods. Rapid advancement has been slowed by a shortage of training data, a scarcity of available datasets, and disputes about the demands of building code advocates. Moreover, most approaches were limited to quantitative requirements. To bridge the gap to fully automated suitability testing, requirements in tabular and graphical form, as well as qualitative and existential requirements, need to be translated into computable form. Some of these gaps can be filled by utilising cutting-edge NLP and incorporating the conversion process with the proper if required, human quality assurance techniques.

This document aims to take the first line of code and the docstring and generate the corresponding function body [2]. They proposed a different baseline based on a more conventional sequence-by-sequence methodology in order to produce a more original code. They research the issue of function auto-completion using provided function signatures and readable documentation. A modified version of GPT-2, a transformation-based NLP model that was trained to produce natural language from very big datasets, is the model that performs the best overall. It was only trained using 10total available data and the Python subset due to computational constraints. For the char-rnn model [10], this translates to 78,357,395 characters with 1,618 distinct symbols or around 50 MB of raw text. Although it is obvious that the model performs worse than the word bag baseline, this behaviour does not

seem to have an impact on the total MRR obtained on the validation set. We suggest that programming languages may be viewed as another specialized domain, along with encyclopedia articles, news, or literature, despite the model's concentration on code and not on natural language.

This paper presents his model FLAIR as a framework work designed to facilitate experimentation with different embedding types and training and distribution of subsequent labeling and text classification models [3]. Her current research focuses on developing new embedding types, investigating other downstream tasks, and extending the framework to facilitate multitask learning approaches. FLAIR was presented as a framework designed to facilitate experimentation with different embedding types, tagging sequences and training and distributing text classification models. The default flavor is a single language model intended to run on GPUs, and typically uses embedding of the hidden state language model. The fast variant model typically uses computationally inexpensive embeddings of LM with 1024 hidden states and is suitable for running on CPU setups. A multilingual part-of-speech tagging model that forecasts common PoS tags for text in 12 languages is also included in the model. This study provides a framework for training embedded texts and models and covers model zones with language models, text classification, and sequence labeling using pre-trained labels. They provide customers the option of fine-tuning models for specific use cases or using pre-trained models for their own content.

The purpose of this study is to clarify how and to what degree NL2Code developer assistants, which are contemporary natural language programming approaches for code creation and retrieval might be helpful in the development workflow [4]. They first tokenize and tag each version of a code snippet using a Python tokenizer, then use the nonparametric Wilcoxon signed-rank test to compare the distributions of lengths before and after edits for code snippets coming from each of the two underlying models, generation and retrieval. Additionally, they computed the median difference between the members of the two groups to calculate the effect size, which is known as the Hodges-Lehman estimator. In-IDE code creation and retrieval were thoroughly studied by the paper's users in order to create an experimental framework and harness. The findings were mixed in terms of the influence on the developer workflow, including time efficiency, code accuracy, and code quality. This showed the difficulties and limits in the current state of both code creation and code retrieval. Regarding the decades-old issue of "natural language programming," or having humans educate machines in the same (natural) language they converse in with one another, which may be helpful in many contexts, such as stated in the Introduction, this study area has shown significant promise.

Table I. is a comparison of all articles in the literature review, and we can understand that there are various types of research done in this area of natural language for code generation. Also, most of the work is done using deep learning or neural networks and their types. We also find that in many cases the results are directly dependent on the dataset used.

Sr.No	P	Y	Dataset	Accuracy	Method/Technique
1	[2]	2017	Custom Dataset with Intents	88.6%	Sum of Word Embedding using GloVe
2	[3]	2019	Code Snippets	78.5%	Tokenization with Wilcoxon signed-rank
3	[4]	2019	Abstract Syntax Tree	90.3%	LSTM with Multi-Layer Perceptron
4	[6]	2020	Custome Dataset with limited vocabulary	75.8%	Encoder-Decoder architecture
5	[7]	2021	Semantic role labels	89.8%	Deep learning dependency parser
6	[9]	2020	Neural Bag of Words	67.3%	GPT-2, a transformer-based NLP model.

P – Paper Cited, Y – Year

Figure 2.1: COMPARISON OF DIFFERENT SURVEY PAPERS

Chapter 3

Design

We have developed a text-to-code system, in which, when a natural language query is given as input it will give out the relevant python code snippets. We achieve this by first creating our dataset according to our requirements, which is then followed by its annotation. Later we train various intent classification models to find the best one, and for entity recognition, we train the NER model upon the annotated dataset. The proposed system consists of 4 modules namely Dataset collection and Annotation process, Architecture, Training and Testing, Sum of Word Embedding using GloVe. The Complete dataset is divided into two parts and their format are explained below:

•User Query Dataset: Contains user queries and their respective intent. This dataset contains real-world user queries which are collected from many users via the survey, in which users were asked to frame queries to get the code that is commonly used in the data-science life cycle, and were also asked to paraphrase some of data science-related queries. Additionally, we have also curated data from Kaggle notebooks, GitHub open-source code, StackOverflow, blogs, and documentation of data-science libraries. Later we manually annotated each user query to their respective intent.
• Model Dataset: Contains code snippets, and their respective intents and entities. The intent, entity name, and entity value present in this dataset are manually annotated for each code snippet. The dataset contains code snippets, and their respective intents and entities. The intent, entity name, and entity value present in it are manually annotated for each code snippet. Example: Below is an example from the Codify dataset where we have Python code for specific intent and entities. After the identification of intent and entities from the user query, its respective code is fetched from this dataset and given as output to the user.

3.1 Architecture

In this system, the input given by the user is first cleansed. It is then via the use of multiprocessing, simultaneously passed for intent classification and entity recognition, this makes our system reliable, fast and efficient. In Intent classification, we classify the category of the user input statement and in entity recognition, we identify specifically which technique, method, algorithm, approach, etc are to be used. Lastly based on the classified intents and entities, the relevant code is fetched from the Codify dataset and is made visible to the user.

(a)Input Cleaning: The input given by the user is first cleansed by making the sentence free from all punctuation marks, quotations, and leading and trailing extra spaces. Also, the acronyms present in the sentence are replaced with their respective full form and every character to its lowercase form.
(b)Intent Classification: The cleaned input is then passed to the best intent classification model which will classify the intent of the given user query.
(c)Entity Recognition: The cleaned input is passed to the Named Entity Recognition (NER) model. 'Named Entity Recognition refers to the Natural Language Processing task of identifying important objects (eg. person, location) from the given text.
(d)Get Code from DB: From the classified intent and the recognized entity of the given user query, we fetch the relevant code from the "Codify Dataset", format it accordingly, and send it to the user.

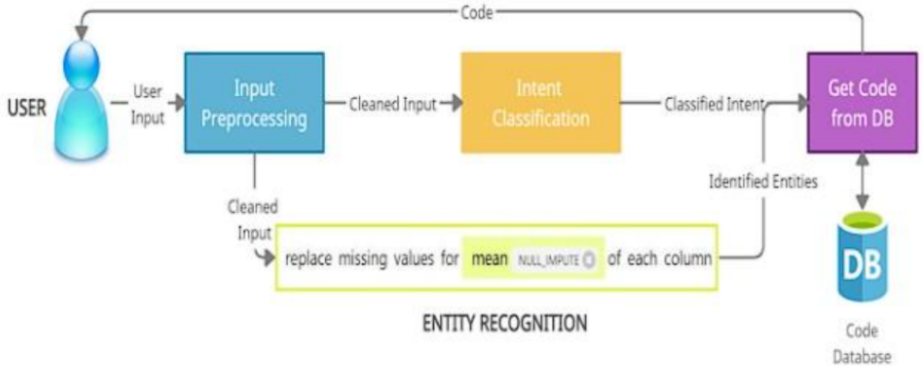

Figure 3.1: Architecture

3.1.1 Sum of Word Embedding using GloVe

We tried a lot of ways to classify intent pertaining to our dataset, out of which we found the "Sum of Word Embedding via GloVe" approach to be the best. Basically, GloVe word embedding derives the relationship between the words using statistics. They make use of the co-occurrence matrix to tell you how often a particular word pair occurs together. Each value in the co-occurrence matrix represents a pair of words occurring together.

As you can observe in figure, we tokenized each word from the sentence and then used GloVe embeddings of 6 billion tokens and 200 dimensions, to get the word embeddings of each word. Then we sum them up to get the sentence embeddings. And later this sentence embedding is fed into the SVC (Support Vector Classifier) the model which will classify intent.

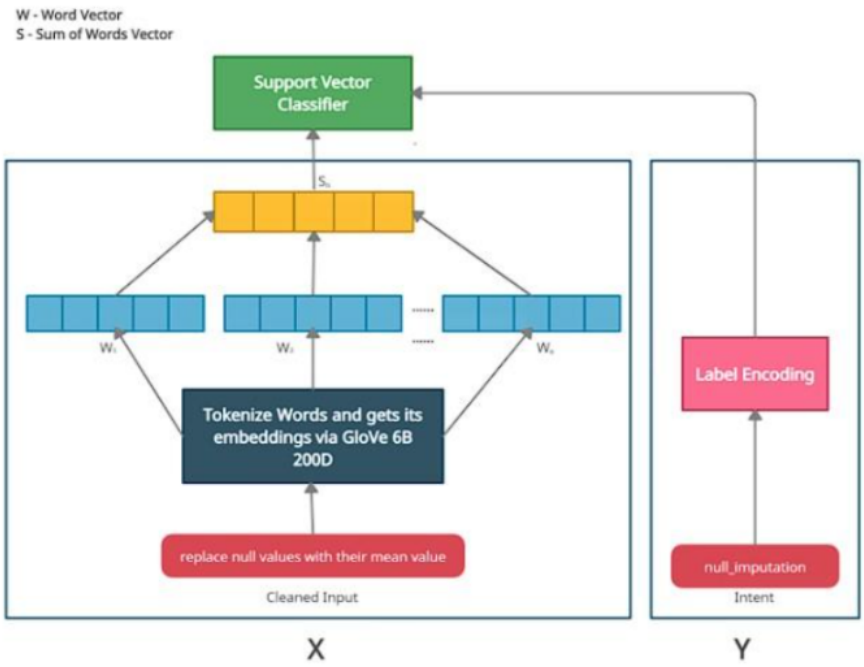

Figure 3.2: Sum of Word Embedding using GloVe Architecture

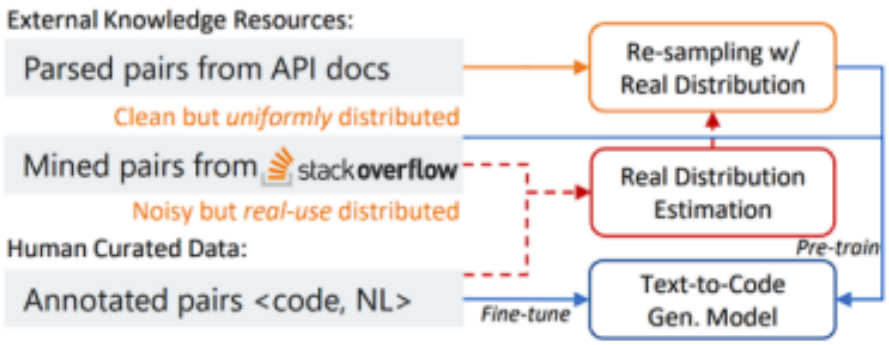

Figure 3.3: Incorporating external knowledge by data re-sampling, pre-training, and fine-tuning

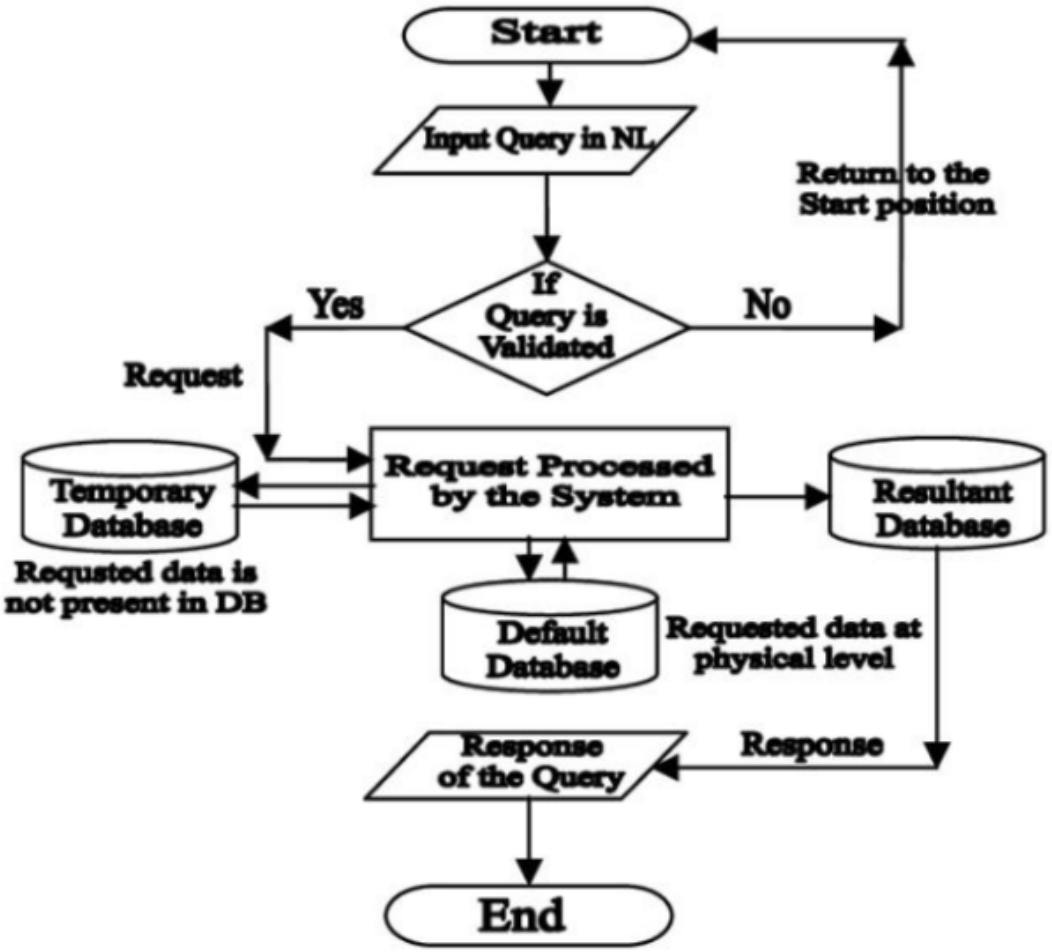

Figure 3.4: Flowchart for user query

Chapter 4

Implementation

A smart system is proposed that allows data scientists to perform all of the tedious and time-consuming tasks in the data-science life cycle, like EDA, data cleaning, data preprocessing, data visualization, modeling, and evaluation, by simply conveying the logic of the task in natural language (English), and the system will automatically provide all of the relevant python code snippets. As a result, we want to design a logic-oriented system that just requires the user to accurately transmit the reasoning in plain language via text. This will very probably save time dramatically, and data scientists will be able to dedicate the majority of their time to logic rather than coding. The main objectives of this model will be
(a) To automate the workflow.
(b) To increase human capability and productivity.
(c) To reduce the execution time of activities or tasks.
(d) To convert natural language queries to their most relevant python code snippets.
(e) To use a custom-made dataset consisting of real-world user queries, their respective intents, entities, and corresponding python code snippets.
(f) To Helps data scientists focus more on the logic-building part of the project that will be of significant help.

The User Query Dataset contains user queries and their corresponding intent. This dataset contains real-world user queries collected from a large number of users via the Electronic copy available at: https://ssrn.com/abstract=4111759 survey, in which users were asked to frame queries to obtain the code that is commonly used in the data-science life cycle, as well as to paraphrase some data science-related queries. We additionally curated data from Kaggle notebooks, GitHub open-source code, StackOverflow, blogs, and data-science library documentation. Later, we manually annotated each user query to its intended purpose. Examples: An example from the User Query dataset is given below, where the text column contains the user query and intent and the entities. The column has its specific intent and entities respectively. The dataset attributes include:
(a)id: Represents a unique row number.
(b)text: This field contains an example of a natural language query that would be given as input by the user at runtime.
(c)intent: For unique identification of each natural language query, this specifies the category under which user query lies like null imputation, encoding, classification, etc Entity Recognition refers to the Natural Language Processing task of identifying

important objects (eg. person, location) from the given text.
(d)entities: Terms that represent methods, techniques, types, etc.

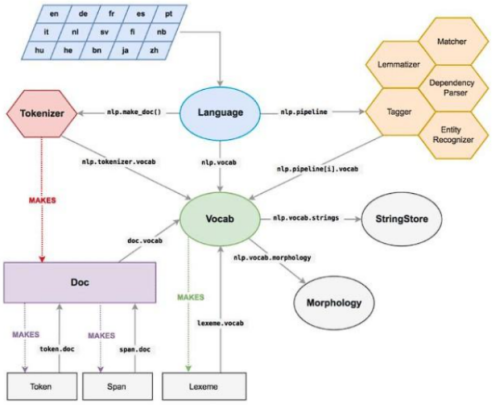

Figure 4.1: ER diagram

The dataset contains code snippets, and their respective intents and entities. The intent, entity name, and entity value present in itare manually annotated for each code snippet. Example: Below is an example from the Codify dataset where we have Python code for specific intent and entities. Dataset Attribute:
(a)code id: Represents unique code snipped ID.
(b)intent: For unique identification of each natural language query, this specifies the category under which user query lies like null imputation, encoding, classification, etc.
(c)entity name: Terms that represent methods, techniques, type, etc entity value: Value of entity obtained from user query like method/technique name, algorithm name, etc.
(d)priority: Since there can be multiple different code snippets available for the same intent, entity name, and entity value, this column specifies the importance of that particular snippet, and at the user end, when multiple snippets are given as output they are arranged according to this column value. python code: Contains relevant code of corresponding text in the text column. >

4.0.1 Tech Stack

• Python: To build backend logic and Flask APIs for accessing machine learning code from the web.
• React: For UI, having a text editor developed on react.
• SQLite: Used as a database to store data.
• GloVe: For word embedding.
• Jupyter Notebook - Anaconda: To develop machine learning models and go through datasets.
• NER model: That identifies intents and entities, respectively, from the input.

Chapter 5

Results and Discussion

5.1 Mined NL-code Pairs

Results are summarized in Figure 5.1. We can first see that by incorporating more noisy mined data during pre-training allows for a small improvement due to increased coverage from the much larger training set. Further, if we add the pairs harvested from API docs for pre-training without re-sampling the performance drops, validate the challenge of distributional shift mentioned in § 2.4. Comparing the two re-sampling strategies direct vs. dist., and two different retrieval targets NL intent vs. code snippet, we can see that dist. performs better with the code snippet as the retrieval target. We expect that using code snippets to retrieve pairs performs better because it makes the generation target, the code snippet, more similar to the real-world distribution, thus better training the decoder. It is also partly because API descriptions are inherently different than questions asked by developers (e.g. they have more verbose wording), causing intent retrieval to be less accurate.

Data Strategy	Method		BLEU
Man			27.20
Man+Mine	50k		27.94
	100k		28.14
Man+Mine+API	w/o re-sampling		27.84
	direct	intent	29.66
	dist.	intent	29.31
	direct	code	30.26
	dist.	code	**30.69**
Man			30.11
Man+Mine(100k)	+rerank		31.42
Our best			**32.26**

Figure 5.1: Performance comparison of different strategies to incorporate external knowledge.

Lastly, we apply hypothesis reranking to both the base model and our best

approach and find improvements afforded by our proposed strategy of incorporating external knowledge are mostly orthogonal to those from hypothesis reranking.

After showing the effectiveness of our proposed re-sampling strategy, we are interested in the performance on more-used versus less-used APIs for the potentially skewed overall performance. We use string matching heuristics to obtain the standard Python APIs used in the dataset and calculated the average frequency of API usages in each data instance. We then select the top 200 and the bottom 200 instances out of the 500 test samples in terms of API usage frequencies. Before and after adding API docs into pre-training, the BLEU score on both splits saw improvements: for high-frequency split, it goes from 28.67 to 30.91 and for low-frequency split, it goes from 27.55 to 30.05, indicating that although the re-sampling would skew towards highfrequency APIs, with the appropriate smoothing temperature experimentation, it will still contribute to performance increases on low-frequency APIs.

Besides using BLEU scores to perform holistic evaluation, we also perform more fine-grained analysis of what types of tokens generated are improving. We apply heuristics on the abstract syntax tree of the generated code to identify tokens for API calls and variable names in the test data, and calculated the token-level accuracy for each. The API call accuracy increases from 31.5% to 36.8% and the variable name accuracy from 41.2% to 43.0% after adding external resources, meaning that both the API calls and argument usages are getting better using our approach.

5.2 Sum of Word Embedding using GloVe

The complete dataset was divided into 85% train set and 15% test, giving us the accuracy as shown in Figure 5.2. The best result was achieved by the "Sum of Word Embedding using GloVe" technique giving 88.60% accuracy. It was then tested with another effective approach, Facebook InferSent, which is an encoder-based bi-directional LSTM architecture with max pooling. Trained on the SNLI dataset, it yielded an accuracy of 87.34%.

Sr.no.	Method	Accuracy
1.	Sum of Word Embedding using GloVe	88.60%
2.	Facebook InferSent	87.34%
3.	Semantic Subword Hashing	78.48%
4.	TF-IDF	74.68%

Figure 5.2: Intent Classification Techniques Accuracy

Semantic Subword Hashing was the next process we trained the model on our dataset. It is a process consisting of subword semantic hashing which is followed by preprocessing and dataaugmentation, vectorization, intent classification, and evaluation. It gave an accuracy of 78.48%. Finally, we used the TF-IDF approach where the first task was to pre-process the user queries, and then the context vectors were created using word embedding techniques. In the final stages, we created clusters of the word embeddings, and then after studying those created cluster samples, we assigned an intent to each of these clusters. This method of intent classification gave a pretty decent accuracy of 74.68%.

Chapter 6

Conclusions

The field of artificial intelligence has been making tremendous progress in recent years, and one of the areas where it has had a significant impact is in natural language processing. One application of this technology is code generation, which involves converting natural language queries into their most relevant code snippets. To achieve this, we proposed a model-agnostic approach that utilizes data augmentation, retrieval, and data re-sampling to incorporate external knowledge into code generation models. Our approach has been shown to achieve state-of-the-art results, demonstrating its effectiveness in this domain.

To develop our approach, we started by creating a custom-made dataset consisting of real-world user queries, their respective intents, entities, and corresponding Python code snippets. We then implemented four different intent classification techniques, namely GloVe-based word embedding, Facebook InferSent, Semantic Subword Hashing, and TF-IDF, and compared their performance. Our results showed that the GloVe-based word embedding model achieved the best accuracy of 88.6%.

Once we had identified the best intent classification model, we simultaneously fed the user queries into this model and an entity recognition model (NER) to identify intents and entities, respectively, from the input. We then used the identified intents and entities to fetch all the corresponding Python code from the Codify database. This system greatly aids data scientists by providing code snippets quickly and accurately without having to remember long syntaxes that are repeatedly used. This saves time exponentially by not writing redundant code and rather helps data scientists to focus more on the logic-building part of the project which will be of significant help.

In summary, our approach leverages the power of natural language processing to make code generation more efficient and accurate. By incorporating external knowledge and using advanced techniques for intent classification and entity recognition, we are able to provide data scientists with a tool that can save them significant amounts of time and improve the quality of their work. We believe that this approach has great potential to revolutionize the way in which code is generated and to help accelerate progress in the field of artificial intelligence.

Chapter 7

Future Scope

As with any technology, there is always room for improvement and further development. In the case of code generation, one area where we see the potential for improvement is in the evaluation of generated code. While our current approach relies on comparing the generated code to existing code snippets and determining the relevance of the generated code to the given query, in the future, it may be useful to evaluate the generated code by automatically executing it with test cases. This would provide a more comprehensive assessment of the quality and functionality of the generated code, as it would be tested under various conditions and scenarios.

In addition to improving the evaluation of generated code, we also see the potential for generalizing our re-sampling procedures to zero-shot scenarios. This refers to situations where a programmer writes a library and documents it, but nobody has used it yet. In such cases, it may be difficult to determine the most relevant code snippets to include in the library, as there is no usage data to draw from. One solution could be for developers to provide relative estimates of each documented API usage to guide the re-sampling process. Alternatively, we could use existing usage statistics as estimates to guide the re-sampling by finding the nearest neighbors to each API call in terms of semantics.

Another area where we see the potential for improvement is in the accuracy and efficiency of intent classification and entity recognition. While our current approach has been shown to achieve state-of-the-art results, there is always room for improvement in these areas. One potential solution could be to incorporate more advanced deep learning techniques, such as transformer models, to further improve the accuracy and efficiency of these tasks.

Finally, we believe that there is potential to expand the scope of our approach beyond Python code generation. While our current focus has been on generating Python code snippets, the underlying methodology could potentially be applied to other programming languages or even to non-programming-related tasks. By incorporating external knowledge and utilizing advanced natural language processing techniques, we believe that our approach could have applications in a variety of fields and domains. Overall, we see a bright future for code generation and natural language processing, and we look forward to continuing to explore the potential of these technologies.

Chapter 8

Research Publication

This paper is on my project title which is submitted and accepted in the IEEE publication

1. Paper Title: Smart intelligent system that can code like a human being
    - Conference Name: 4th International Conference of Emerging Technologies (INCET)
    - Oral Presentation Venue: Hybrid mode
    - Conference Date: 26th May June - 28th May, 2023
    - Status: Accepted

Smart intelligent system that can code like a human being

Jaiwin Shah
Computer Engineering Department
Sardar Patel Institute of Technology
Mumbai, India
jaiwin.shah@spit.ac.in

Rishabh Jain
Computer Engineering Department
Sardar Patel Institute of Technology
Mumbai, India
rishabh.jain2@spit.ac.in

Vedant Jolly
Computer Engineering Department
Sardar Patel Institute of Technology
Mumbai, India
vedant.jolly@spit.ac.in

Dhananjay Kalbande
Computer Engineering Department
Sardar Patel Institute of Technology
Mumbai, India
drkalbande@spit.ac.in

***Abstract*—According to recent studies, a large number of data scientists spend most of their time on tasks like data cleaning and organizing data. They need to memorize big complex syntaxes for all the major tasks in the data science life cycle. Often these tasks are redundant. Therefore, we propose to build an intelligent system that enables data scientists to perform all the tedious and time-consuming tasks such as EDA, data cleansing, data preprocessing, data visualization, modeling, and data science lifecycle evaluation. Just state the logic of your query in natural language the system will automatically output all relevant Python code snippets. Existing applications involving the text-to-code generation and code search are limited and a lot of them do not work in non-ideal conditions. The reason behind it is the data set on which the existing models have been built. These datasets do not consider real-world factors such as slang, acronyms, and paraphrases. Therefore, a new dataset was created consisting of real-world user queries, representing the scenarios a user is most likely to face daily. We plan to build a logic-oriented system that only needs to convey the logic correctly in text in natural language. It saves a lot of time, allowing data scientists to spend most of their time building logic instead of focusing on code.**

***Index Terms*—Deep learning, Source code modeling, Source code generation, natural language programming, code generation, code retrieval**

I. Introduction

Developers always ask how to translate ideas into code. For example, saving keystrokes or avoiding writing boring chunks of code. It is also useful for non-programmers and practitioners in other fields who need calculations in their daily work to create data manipulation scripts.

Automating even small parts of software development is an active research area with several approaches and proposed methods. A machine learning model that allows developers to automate coding tasks by allowing them to create individual functional units. Instead of generating code or completing developer-written code, we can restate the problem by looking for relevant snippets that already exist.

Classical pre-trained word embeddings prove very useful for his NLP tasks downstream. This is due to the ability to use information gleaned from unlabeled data to support learning and generalization, and the relative ease of including them in any learning approach. A common example is that many current approaches combine traditional word embeddings with character-level features trained on task data. To achieve this, we use a hierarchical learning architecture in which the output states of the CNN or RNN are chained to the output of the embedding layer at the character level. These methods have proven to be very powerful, especially when combined with traditional word embeddings, but in such methods the output state of the trained language model (LM) is concatenated to the output of the embedding layer. Requires an architecture that supports and increases architectural complexity.

This system clears user entries first. This is passed simultaneously for intent classification and entity recognition using multiprocessing and based on the classified intents and entities, the relevant code is pulled from the data set and displayed to the user.

This article is organized as follows: Starting with Section II, this section provided a brief overview of the literature review, and Section III provided a comparison of various technical papers. And the final section, Section IV, summarizes the merits and limitations of this study, with a brief reference to future research.

II. Literature Review

To establish the study's parameters and frame the research issue, a preliminary literature review was carried out. Other studies have interpreted regulations outside of the fields of architecture, engineering, and construction, and his article on the use of NLP for the semantic interpretation of building regulations is one of only eight. Some studies have concentrated on automated conformance checking and the presentation of regulations. These articles demonstrated how building code conversion may be broken down into a number of distinct processes. 1. In what ways might NLP technology facilitate or streamline the understanding of construction regulations?

This thorough literature search turned up 41 papers on natural language processing that dealt with how to read building codes. The 1,962 records that were gathered from six databases and the extra candidate articles found using the reverse snowball and author search procedures comprised the selection of these articles. The papers were then grouped, distilled, and examined. Eight research gaps were eventually discovered, and suggestions were offered on how to close them. To close that gap, machine learning has been studied, but these investigations have not yet matched the strength of rule-based methods. Rapid advancement has been slowed by a shortage of training data, a scarcity of available datasets, and disputes about the demands of building code advocates. Moreover, most approaches were limited to quantitative requirements. To bridge the gap to fully automated suitability testing, requirements in tabular and graphical form, as well as qualitative and existential requirements, need to be translated into computable form. Some of these gaps can be filled by utilising cutting-edge NLP and incorporating the conversion process with the proper if required, human quality assurance techniques.

This document aims to take the first line of code and the docstring and generate the corresponding function body [2]. They proposed a different baseline based on a more conventional sequence-by-sequence methodology in order to produce a more original code. They research the issue of function auto-completion using provided function signatures and readable documentation. A modified version of GPT-2, a transformation-based NLP model that was trained to produce natural language from very big datasets, is the model that performs the best overall. It was only trained using 10% of the total available data and the Python subset due to computational constraints. For the char-rnn model [10], this translates to 78,357,395 characters with 1,618 distinct symbols or around 50 MB of raw text. Although it is obvious that the model performs worse than the word bag baseline, this behaviour does not seem to have an impact on the total MRR obtained on the validation set. We suggest that programming languages may be viewed as another specialized domain, along with encyclopedia articles, news, or literature, despite the model's concentration on code and not on natural language.

This paper presents his model FLAIR as a framework work designed to facilitate experimentation with different embedding types and training and distribution of subsequent labeling and text classification models [3]. Her current research focuses on developing new embedding types, investigating other downstream tasks, and extending the framework to facilitate multitask learning approaches. FLAIR was presented as a framework designed to facilitate experimentation with different embedding types, tagging sequences and training and distributing text classification models. The default flavor is a single language model intended to run on GPUs, and typically uses embedding of the hidden state language model. The fast variant model typically uses computationally inexpensive embeddings of LM with 1024 hidden states and is suitable for running on CPU setups. A multilingual part-of-speech tagging model that forecasts common PoS tags for text in 12 languages is also included in the model. This study provides a framework for training embedded texts and models and covers model zones with language models, text classification, and sequence labeling using pre-trained labels. They provide customers the option of fine-tuning models for specific use cases or using pre-trained models for their own content.

The purpose of this study is to clarify how and to what degree NL2Code developer assistants, which are contemporary natural language programming approaches for code creation and retrieval, might be helpful in the development workflow [4]. They first tokenize and tag each version of a code snippet using a Python tokenizer, then use the nonparametric Wilcoxon signed-rank test to compare the distributions of lengths before and after edits for code snippets coming from each of the two underlying models, generation and retrieval. Additionally, they computed the median difference between the members of the two groups to calculate the effect size, which is known as the Hodges-Lehman estimator. In-IDE code creation and retrieval were thoroughly studied by the paper's users in order to create an experimental framework and harness. The findings were mixed in terms of the influence on the developer workflow, including time efficiency, code accuracy, and code quality. This showed the difficulties and limits in the current state of both code creation and code retrieval. Regarding the decades-old issue of "natural language programming," or having humans educate machines in the same (natural) language they converse in with one another, which may be helpful in many contexts, as stated in the Introduction, this study area has shown significant promise.

This paper has developed a text-to-code system, in which, when a natural language query is given as input it will give out the relevant python code snippets [5]. They have achieved this by first creating a dataset according to the requirements, which is followed by its annotation. The paper has then trained different intent classification models to find the best one and used the NER model on the annotated dataset for entity recognition. The proposed system consists of a total of four modules: record collection and annotation process, architecture, training and testing, and word embedding using GloVe. The best result was, obtained with the "Total Word Embeddings Using GloVe" method, giving an accuracy of 88.6%. I trained on the SNLI dataset and got an accuracy of 87.34%. Then, after creating clusters of word embeddings and examining these sample clusters of his created, paper assigned an intent to each of these clusters. This intent classification method returned a pretty decent accuracy of 74.68%. This paper implements different intent classification techniques: GloVe-based word embeddings, Facebook InferSent, semantic subword hashing and TF-IDF, and concludes that GloVe-based word embedding models reach the highest accuracy

III. COMPARISON OF DIFFERENT SURVEY PAPERS

Sr.No	P	Y	Dataset	Accuracy	Method/Technique
1	[2]	2017	Custom Dataset with Intents	88.6%	Sum of Word Embedding using GloVe
2	[3]	2019	Code Snippets	78.5%	Tokenization with Wilcoxon signed-rank
3	[4]	2019	Abstract Syntax Tree	90.3%	LSTM with Multi-Layer Perceptron
4	[6]	2020	Custome Dataset with limited vocabulary	75.8%	Encoder-Decoder architecture
5	[7]	2021	Semantic role labels	89.8%	Deep learning dependency parser
6	[9]	2020	Neural Bag of Words	67.3%	GPT-2, a transformer-based NLP model.

P – Paper Cited, Y – Year

Table I. is a comparison of all articles in the literature review, and we can understand that there are various researches done in this area of natural language for code generation. Also, most of the work is done using deep learning or neural networks and their types. We also find that in many cases the results are directly dependent on the dataset used.

The potential of the DL model to address the problem is emphasized, providing more comprehensive solutions to various problems. We then explore the key components of encoder/decoder frameworks using DL models and make some suggestions for using DL models in encoder/decoder architectures for modeling and source code creation. We then discussed various advanced code generation applications such as source code analysis and program development. The gap between the most advanced DL models and their utility in modeling and source code generation was also highlighted. It included some conclusions on AI security for modeling and source code generation. Despite the excellent performance of DL in many different disciplines, a new research trend known as adversarial DL has recently emerged, highlighting the susceptibility and potential of DL models to negative outcomes. Subsequently, the model used to model his sequence was also changed to incorporate the concept of adversarial DL. Findings like this suggest that source code models can be fooled, for example, by turning predicted vulnerable code snippets into harmless ones. A major drawback of previous neural code completion models is the need for comprehensive answers to real-world problems. More special features follow.

• A fixed vocabulary is used to train the model. So to create new OoV values you have to retrain the model

• Rather than LL, incomplete code produces confusing parsing results for languages with complicated grammars (1). As a result, the prediction may be erroneous since the present condition cannot be directly translated to a training input.

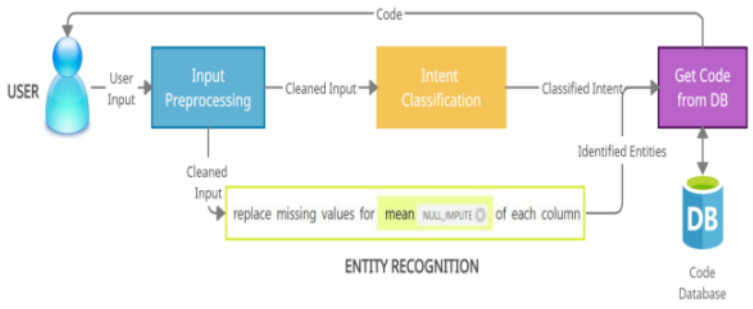

Fig. 1. Basic Architecture for code generation [11]

IV. Conclusion

Most of the models are built using the custom-made dataset consisting of real-world user queries, their respective intents, entities, and corresponding python code snippets. Different intent classification techniques namely GloVe based word embedding, Facebook InferSent, Semantic Subword Hashing, and TF-IDF, following which we came to the conclusion that GloVe based word embedding model helped achieve the best accuracy of 88.6%. The user queries are then simultaneously fed into the best intent classification model and entity recognition model (NER) which identifies intents and entities, respectively, from the input. Using the identified intents and entities, the models fetch all the corresponding python code from the database. This system greatly aids data scientists by providing code snippets quickly and accurately without having to remember long syntaxes that are repeatedly used. This saves time exponentially by not writing redundant code and rather helps data scientists to focus more on the logic building part of the project which will be of significant help.

On the one hand, the database, search terms and author selection can be improved. The publications from Scopus and ProQuest that were included were a subset of the findings from Engineering Village. Therefore, further research into the database-journal relationships covered by a particular database could reduce the effort involved in document integration and exclusion. Some journals such as artificial intelligence and law were included in the search database, but the inclusion of legal databases may improve the quality of search results. Based on data from two databases, search phrases were assessed and optimised. Very detailed search parameters were used as SpringerLink searches the entire text of articles and book chapters. This issue may be resolved and a wider search made possible by separating the collection of keywords for full-text searches from those for abstracts and title searches. To make the effort more reasonable, we also set the author search barrier to 3 authors. Additionally, literature searches document the state of the art at a specific moment. Between the first literature search and the review's final documentation, there was a delay, and the initial database alerts put on the search revealed new literature. In this study, using bi-LSTM, self-recognition, character-level embeddings, and word-level embeddings, we constructed a deep learning-based system. They outperformed various base architectures with 88.1% accuracy and 85.2% recall. These papers confirm the possibility of information extraction by deep learning The XML structure maintains the inherent hierarchy of regulations, enriched with additional features such as references, concepts, and exceptions. Further research is needed to enable scalable and powerful deep information extraction. The bulk of information extraction techniques combined rules with subject-matter expertise. Knowledge bases were ontologies and gazetteers.

To evaluate these impacts, more research is required. We presumptively have a bidirectional distortion. On the one hand, users may favour code generation outcomes by maintaining other factors like order owing to the novelty effect. Users may, however, favour code search results due to their overall mistrust of automatically produced code, such as:

A. Reported auto-generated unit tests. To aid users in understanding the code, future research should take into account providing more context and explanation along with the plugin findings.

B. Links to official documentation sites, definitions of domain-specific concepts, and other API usage examples.

References

[1] Xu, F. F., Vasilescu, B., Neubig, G. (2021). In-IDE Code Gen-eration from Natural Language: Promise and Challenges. arXiv.https://doi.org/10.48550/arXiv.2101.11149

[2] Fuchs, Stefan. (2021). Natural Language Processing for Building Code Interpretation: Systematic Literature Review Report.10.13140/RG.2.2.29107.55845.

[3] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stvedaefan Schweter, and Roland Vollgraf. 2019. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. In Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

[4] Perez, L., Ottens, L., Viswanathan, S. (2021). Automatic Code Generation using Pre-Trained Language Models. arXiv. https://doi.org/10.48550/arXiv.2102.10535

[5] Le, T. H., Chen, H., Babar, M. A. (2020). Deep Learning for Source Code Modeling and Generation: Models, Applications and Challenges. arXiv. https://doi.org/10.1145/3383458

[6] R. Tiwang, T. Oladunni and W. Xu, "A Deep Learning Model for Source Code Generation," 2019 SoutheastCon, 2019, pp. 1-7, doi: 10.1109/SoutheastCon42311.2019.9020360.

[7] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[8] C. Jeong, S. Jang, E. Park, and S. Choi, "A context-aware citation recommendation model with bert and graph convolutional networks", March 2019, Scientometrics, vol.124, pp. 1–16.

[9] Roman, Muhammad Shahid, Abdul Khan, Shafiullah Koubaa, Anis Yu, Lisu, "Citation Intent Classification Using Word Embedding", January 2021, IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3050547.

[10] Conneau, Alexis Kiela, Douwe Schwenk, Holger Barrault, Loïc Bordes, Antoine, "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data", Proceedings of the 2017 Conference on Empirical Methods in Natural Language, July 2018, 670-680. 10.18653/v1/D17-1070.

[11] K. Prasad, P. S. Sajith, M. Neema, L. Madhu and P. N. Priya, "Multiple eye disease detection using Deep Neural Network," TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019, pp. 2148-2153, doi: 10.1109/TENCON.2019.8929666.

[12] Shridhar, Kumar Dash, Ayushman Sahu, Amit Grund Pihlgren, Gustav Alonso, Pedro Pondenkandath, Vinaychandran Kovacs, Gyorgy Simistira, Fotini Liwicki, Marcus, "Subword Semantic Hashing for Intent Classification on Small Datasets", 2019 International Joint Conference on Neural Networks (IJCNN), Sept 2019, 1-6. 10.1109/IJCNN.2019.8852420.

[13] S. Borade and D. Kalbande, "Survey paper based critical reviews for Cosmetic Skin Diseases," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), Coimbatore, India, 2021, pp. 580-585, doi: 10.1109/ICAIS50930.2021.9395803.

[14] F. Lara, J. Arellano, M. Castillo, L. Topón and E. V. Carrera, "Source Coding for Text Transmission using a Deep Neural Network as a Lossy Compression Stage," 2020 IEEE ANDESCON, Quito, Ecuador, 2020, pp. 1-5, doi: 10.1109/ANDESCON50619.2020.9272105.

Bibliography

[1] Xu, F. F., Vasilescu, B., Neubig, G. (2021). In-IDE Code Gen-eration from Natural Language: Promise and Challenges. arXiv.https://doi.org/10.48550/arXiv.2101.11149

[2] Fuchs, Stefan. (2021). Natural Language Processing for Building Code Interpretation: Systematic Literature Review Report.10.13140/RG.2.2.29107.55845.

[3] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stvedaefan Schweter, and Roland Vollgraf. 2019. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. In Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

[4] Perez, L., Ottens, L., Viswanathan, S. (2021). Automatic Code Generation using Pre-Trained Language Models. arXiv. https://doi.org/10.48550/arXiv.2102.10535

[5] Le, T. H., Chen, H., Babar, M. A. (2020). Deep Learning for Source Code Modeling and Generation: Models, Applications and Challenges. arXiv. https://doi.org/10.1145/3383458

[6] R. Tiwang, T. Oladunni and W. Xu, "A Deep Learning Model for Source Code Generation," 2019 SoutheastCon, 2019, pp. 1-7, doi: 10.1109/SoutheastCon42311.2019.9020360.

[7] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[8] C. Jeong, S. Jang, E. Park, and S. Choi, "A context-aware citation recommendation model with bert and graph convolutional networks", March 2019, Scientometrics, vol.124, pp. 1–16.

[9] Roman, Muhammad Shahid, Abdul Khan, Shafiullah Koubaa, Anis Yu, Lisu, "Citation Intent Classification Using Word Embedding", January 2021, IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3050547.

[10] Conneau, Alexis Kiela, Douwe Schwenk, Holger Barrault, Loïc Bordes, Antoine, "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data", Proceedings of the 2017 Conference on Empirical Methods in Natural Language, July 2018, 670-680. 10.18653/v1/D17-1070.

[11] K. Prasad, P. S. Sajith, M. Neema, L. Madhu and P. N. Priya, "Multiple eye disease detection using Deep Neural Network," TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019, pp. 2148-2153, doi: 10.1109/TENCON.2019.8929666.

[12] Shridhar, Kumar Dash, Ayushman Sahu, Amit Grund Pihlgren, Gustav Alonso, Pedro Pondenkandath, Vinaychandran Kovacs, Gyorgy Simistira, Fotini Liwicki, Marcus, "Subword Semantic Hashing for Intent Classification on Small Datasets", 2019 International Joint Conference on Neural Networks (IJCNN), Sept 2019, 1-6. 10.1109/IJCNN.2019.8852420.

[13] S. Borade and D. Kalbande, "Survey paper based critical reviews for Cosmetic Skin Diseases," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), Coimbatore, India, 2021, pp. 580-585, doi: 10.1109/ICAIS50930.2021.9395803.

[14] F. Lara, J. Arellano, M. Castillo, L. Topón and E. V. Carrera, "Source Coding for Text Transmission using a Deep Neural Network as a Lossy Compression Stage," 2020 IEEE ANDESCON, Quito, Ecuador, 2020, pp. 1-5, doi: 10.1109/ANDESCON50619.2020.9272105.

Black_Book

ORIGINALITY REPORT

6%	2%	4%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.safaribooksonline.com Internet Source	2%
2	"Intelligent Data Communication Technologies and Internet of Things", Springer Science and Business Media LLC, 2021 Publication	2%
3	Dou Wei. "Prediction of Stock Price Based on LSTM Neural Network", 2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM), 2019 Publication	2%

Exclude quotes	On	Exclude matches	< 2%
Exclude bibliography	On		

Res_Paper

ORIGINALITY REPORT

13%	4%	9%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Frank F. Xu, Bogdan Vasilescu, Graham Neubig. "In-IDE Code Generation from Natural Language: Promise and Challenges", ACM Transactions on Software Engineering and Methodology, 2022 Publication	6%
2	Shwetambari Borade, Dhananjay Kalbande. "Survey paper based critical reviews for Cosmetic Skin Diseases", 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021 Publication	2%
3	www.researchgate.net Internet Source	2%
4	www.inf.uni-hamburg.de Internet Source	1%
5	itc.scix.net Internet Source	1%
6	Muhammad Asif Hanif, Farwa Nadeem, Rida Tariq, Umer Rashid. "Geothermal energy	<1%

production", Elsevier BV, 2022
Publication

7	Submitted to Adtalem Global Education, Inc. Student Paper	<1%
8	"Performance Evaluation and Benchmarking for the Era of Cloud(s)", Springer Science and Business Media LLC, 2020 Publication	<1%
9	Sudhir Rao Rupanagudi, Varsha G Bhat, Bharath Kumar Revana, Jeevitha Gowda Chandramouli et al. "Optic Disk Extraction and Hard Exudate Identification in Fundus Images using Computer Vision and Machine Learning", 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), 2021 Publication	<1%
10	aclanthology.org Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		