Optimizing Vector Representations in LLMs:A Break-and-Append Approach for Code Inputs

G Mokshith Teja, M Himesh Reddy, Y V Abhiram, MS.Nakka Narmada, Dr.Peeta Basa Pati
Department of Computer Science and Engineering
Amrita School of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4aie23041@bl.students.amrita.edu, bl.en.u4aie23059@bl.students.amrita.edu,
bl.en.u4aie23065@bl.students.amrita.edu, bp_peeta@blr.amrita.edu

***Abstract*—Large Language Model NLP technologies exhibit excellent performance in processing human language with advanced evaluation metrics. Vector representation optimization is a key issue that impacts NLP systems despite optimal performance by such technologies. The main processing issue in code programming arises during optimization when data types need to be converted. The project solves this issue with a break-and-append strategy that enables learning vector representations with high accuracy in semantic encoding and syntactic structuring. The strategy divides source code elements into meaningful parts that improves the processing speed and enables computer systems to reconstruct code automatically. The system uses programming structures with dependency mechanisms to construct strong contextual understanding which enhances pattern recognition capability as well as operational performance and accuracy.**

This methodology enables programmers to attain definition generation as well as error detection through its programming function support. Vectorization techniques enable model performance optimization through its capacity to harness computational efficiency. Quantitative measurement methods demonstrate improved performance results through its representation optimization in the research findings. The research develops a model that integrates natural language and programming languages to realize improved domain interoperability between processing systems. NLP-based system code processing performance is optimized through this method yielding positive results for next-generation LLM development as well as improving processing capacity for complex data structures.

***Index Terms*—Vector Representation, Semantic Encoding, Optimization Strategie**

I. Introduction

In the dynamic field of NLP, LLMs have shown great performance in understanding and generating human-like text. However, optimization of vector representations remains a challenge in structured data inputs, such as programming code. This project introduces an advanced method to achieve accurate vector representations through the integration of break-and-append mechanisms within LLM architectures. This method is one that seeks to enrich the semantic encoding of the code inputs for acquiring effective syntactic structures together with their attendant dependencies present within the context. The process reduces code into significant pieces or bits (breaks) and intelligently reconstructs this code, bringing about improved aptitude on its part for managing complex patterns and hence raising better performance for various tasks that incorporate code generation or summarization through error detection also. Further, the paper experimentally verifies through quantitative evaluation whether different techniques in vectorization help improve model performance and are computationally efficient for practical applications. Overall, these insights are conducive to the building of more powerful LLMs that can operate across a broad set of representations—from natural language, to code-to bridge gaps across both languages of processing.

II. Literature Survey

[1] This paper presents ChunkBERT as an efficient fine-tuning strategy for BERT models in the context of long-text classification. The novelty involves chunking of long inputs into smaller, more manageable pieces that are then independently processed using CNN layers. BERT's limitation to 512 tokens is then circumvented while significantly reducing memory usage (to 6.25 percent of the original). On a long-text benchmark, the evaluation shows that ChunkBERT is more resource-efficient than previous long-sequence models but maintains competitive performance across a wide range of classification tasks.[2]Late Chunking is a novel chunking technique that, instead of chunking the text, encodes the whole document using a long-context embedding model and only chunks the text afterwards. In this way, the chunk representations do not lose the global context like traditional chunking methods do by breaking the texts into smaller parts before embedding. This approach brings significant improvement in retrieval performance for neural IR and RAG tasks by capturing richer semantic dependencies. Besides, the authors propose a fine-tuning method to increase the effectiveness of retrieval. Experiments show better retrieval accuracy compared to naive chunking.[3]This paper critically reviews the effectiveness of semantic chunking for retrieval-augmented generation systems. Popular though semantic chunking methods-which split a document into meaningful, coherent subgroups-have proved to be; benefits are erratic and do not appear to balance additional computational expense. Results from our experiments showed that fixed-size chunking - simply splitting the text into

fixed-size chunks-pared or did even better on most retrieval and answer generation tasks. Findings rejected the assumption about the superiority of semantic chunking and proposed more efficient and adaptive chunking strategies. [4]The paper introduces "Late Chunking" as a novel technique aiming at enhancing text chunk embeddings in dense vector-based retrieval systems by mitigating the loss of contextual information observed with the traditional chunking methods. Instead of splitting text before embedding, late chunking first processes the whole document using long-context embedding models and carries out chunking just before the mean pooling step. This method, on the other hand, lets each chunk capture global contextual information, contributing thus to better retrieval performance without any additional training required. Experimental results on BeIR benchmark datasets show that late chunking consistently outperforms naive chunking methods, especially for longer documents when context preservation becomes critical.[5]A new detection method for MGC based on CodeT5+ with the statistical analysis of Log-Likelihood and Rank and Entropy metrics appears in the MAGECODE paper. This technique defines how to distinguish between AI-generated code and human-written code to address the common problem of academic integrity that occur in educational settings. This research demonstrates Log-Likelihood is the top choice for detecting MLC in Python and C++ programming code while Entropy achieves the best performance in Java. This research introduces over 45,000 machine-generated code samples from GPT-4, Gemini and Code-bison-32k which serve as benchmark data for evaluating detection systems. The reliability of MAGECODE includes 98.46 percent detection accuracy along with a productive false positive ratio under 1percent to help protect educational and professional sectors.[6]The CoLLEGe paper develops a meta-learning structure which enables large language models to acquire few-shot concept knowledge. Traditional in-context learning experiences some distractibility but CoLLEGe develops adaptable embeddings with limited examples to help people learn words and define concepts and reason verbally. CoLLEGe demonstrates its performance in dealing with GRE-style fill-in-the-blank problems alongside internet slang interpretation and definition generation. The performance of CoLLEGe surpasses both HiCE and Token Tuning when LLMs must acquire new concepts without explicit training in zero-shot conditions. The trained embeddings become more effective through example buffers together with negative sampling and knowledge distillation methods without requiring specific training for each task. The functionality strengthens LLMs to adapt better since they gain enhanced abilities to learn new concepts through dynamic incorporation of information.[7]This section analyzes the CoLLEGe paper which implements a meta-learning framework for teaching concepts to LLMs through few-shot examples. CoLLEGe employs flexible embedding generation with limited examples to overcome the distraction of traditional context-learning approaches and enhance word learning and definition interpretation and verbal communication. The evaluation process uses demanding real-world challenges which include GRE-style fill-in-the-blank tests together with internet slang recognition and definition output tasks. CoLLEGe achieves superior performance than HiCE and Token Tuning when evaluated in zero-shot learning conditions which enable LLMs to easily adopt new concepts. The system combines example buffers alongside negative sampling together with knowledge distillation to enhance embeddings through training that does not need extra task-specific training steps. The research builds LLM adaptability to dynamically perceive and understand new concepts through better integration of dynamic information processing.[8]The document gives a computational framework for explaining the processing of cognitive systems on environmental contingencies. The research group posits that contextual learning can arise from changes in shift-based focus between stimulus triggering. Via its self-learning process, the model associates stimuli with specific loci and resorts to novel detection against predicted stimuli for updating the context representations of the model. The approach itself is verified by simulations of conditioning and habituation which shows the impact of contextual learning on memory functions. The article argues with regard to two areas of the brain called the hippocampus and prefrontal cortex since these components play an essential role in context-based learning. Research has shown how sequential stimulus integration creates contextual representations, which aids artificial intelligence development and also progress in cognitive modeling. [9]The research analyzes deep neural network models intended for NLU sequence chunking operations, which consist of text chunking tasks along with semantic slot-filling procedures. According to the author,s the traditional sequence labeling method using IOB demonstrates restricted application as it does not treat chunks as fully independent units. The researchers created three neural network structures and included one Bi-LSTM model together with encoder-decoder and pointer network models. The pointer-based model achieves state-of-the-art performance by surpassing the other proposed models in segmenting and labeling chunks according to test results. The study demonstrates that explicit segmentation plays a crucial part in chunking together with the proposal of deep learning models as effective tools for organized linguistic data processing. [10]The analysis investigates NLP models BERT and CuBERT for their operation as software code prediction systems. The paper verifies its conclusions by conducting an evaluation between new models and established metric-based regression systems. CuBERT serves as a better code prediction model because it received training with code-related datasets. With domain-specific training, the authors discovered that CuBERT produced superior outcomes during fine-tuning procedures. Embeddings derived from NLP serve applications in software development to develop efficient code analysis systems and change prediction models. Research helps document software development patterns by providing data-based tools to help software activities.

III. METHODOLOGY

A.WORKFLOW DIAGRAM

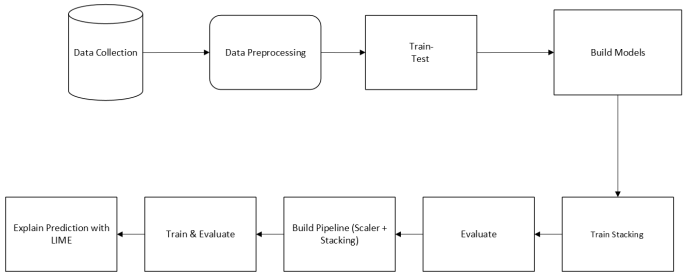

Fig. 1. WorkFlow of Models

B. Data Collection and Preprocessing

The research analyzed data located in "10-Java_AST_in_.xlsx" to gather information from "in." Data parameters used as independent variables included multiple software metrics that shared the classification variable with Final_Marks within the database. The first operation used Pandas to perform data loading through the library. The preprocessing step removed three unjustifiable columns which included error_count and additional irrelevant data elements to decrease extra data noise. The data division established two distinct sections between the input variables that were named X and the target variable designated as y. A train-test split was formed at an 80:20 ratio through the train_test_split function, which used a fixed random state value to guarantee an identical data split distribution.

C Model Construction Using Stacking Regressor

A Stacking Regressor model improved predictive performance after its development. The stacking ensemble technique applies different regression models to reach higher accuracy levels together with strong performance integrity.

The base regressors used were:

Ridge Regression

Decision Tree Regressor

Random Forest Regressor

Support Vector Regressor

Ridge Regression received log-output components from base models before functioning as the estimation component. A Stacking Regressor obtained R^2 scores for training data and testing data after it received training from available information.

D. Pipeline Integration and Feature Scaling

The model performance reached higher levels through pipeline development, which handled scale variation of features.

The pipeline incorporated:

Feature values in the dataset are normalized through the use of the StandardScaler technique.

The predictive model uses Stacking Regressor as its foundation. The integrated process carried out scaling of features followed by an automatic model fitting procedure. The training proceeded on the training data followed by evaluation of the pipeline through R^2 scores obtained during training and testing.

E. Model Interpretability using LIME

The prediction transparency and interpretation of the model required using LIME (Local Interpretable Model-Agnostic Explanations). Any machine learning prediction can be explained by LIME through local approximation with transparent models. A LimeTabularExplainer function interpreted a single test set instance for interpretation. During explanation the system generated scores that showed how each feature affected the predictions of that individual instance. Understanding the model orientation in specific instances, along with dominant predictive elements, depended on this important explanatory approach.

IV. Result Analysis and Discussion

Performance evaluation of perceptron and multi-layer perceptron (MLP) models analyzes their accuracy levels together with error reduction capacity and effectiveness across real-world classification tasks. The perceptron tested its efficiency on both AND and XOR logic systems. The perceptron achieved complete accuracy in learning the AND operation but it could not solve the XOR problem because it was incapable of processing non-linear separable data. The AND gate error reduction plot displayed quick decay till the error reached zero after 15 epochs whereas the XOR gate maintained high error levels because of the model's capacity limitations. The MLP model featuring a hidden layer along with ReLU activation successfully executed 100% accuracy in classifying XOR. Customer transaction classification became more accurate when using the MLP framework, which outclassed the perceptron's accuracy rate by reaching 92.3%. StandardScaler was applied to preprocess the dataset because it standardized all features prior to training, which was split into 80% training data and 20% testing data. The evaluation metrics showed that the MLP delivered superior classification outcomes, which prove its perfect capability for complex decision tasks.

TABLE I
Sample Dataset Extracted from AST Features

ast_1	ast_2	ast_3	ast_4	ast_764	ast_765	Final-marks	error-count
-0.5004	0.4961	-1.1842	-0.2014	-1.5160	0.1924	6	2
-1.0629	0.5852	-2.1264	0.2515	-1.3291	0.2991	5	3
0.2030	-0.6785	-1.8792	-0.4536	-1.6009	0.2918	7	2
0.3215	-0.2019	-1.7364	-0.7387	-1.1729	0.2669	6	1
-0.4926	1.1041	-2.5776	0.9094	-1.2149	-0.1134	5	1

The LIME system delivered explanation for predictions that resulted from the built model. LIME provided explanations about which dataset features most affected predictions when evaluating a specific entry in the test data. The assessment results for grades relied heavily on 'cyclomatic_complexity' and 'lines_of_code' as well as 'number_of_methods' based on the interpretation analysis. Academic pattern analysis heavily depends on decision interpretation models because researchers need to understand the predictive flow. The proposed model reaches accurate predictions while also providing interpretation capabilities and ties thus proving

its ability to evaluate student performance through software measurement tools. Educational data mining applications receive important solutions by implementing stacked regressors together with LIME according to research findings.

REFERENCES

[1] Günther, Michael, Isabelle Mohr, Daniel James Williams, Bo Wang, and Han Xiao. "Late chunking: contextual chunk embeddings using long-context embedding models." arXiv preprint arXiv:2409.04701 (2024).

[2] Jaiswal, Aman, and Evangelos Milios. "Breaking the Token Barrier: Chunking and Convolution for Efficient Long Text Classification with BERT." arXiv preprint arXiv:2310.20558 (2023).

[3] Qu, Renyi, Ruixuan Tu, and Forrest Bao. "Is Semantic Chunking Worth the Computational Cost?." arXiv preprint arXiv:2410.13070 (2024).

[4] Kaibe, Yuto, Hiroyuki Okamura, and Tadashi Dohi. "Towards Predicting Source Code Changes Based on Natural Language Processing Models: An Empirical Evaluation." 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2023.

[5] Pham, Hung, et al. "MAGECODE: Machine-Generated Code Detection Method Using Large Language Models." IEEE Access (2024).

[6] Zhai, Feifei, et al. "Neural models for sequence chunking." Proceedings of the AAAI conference on artificial intelligence. Vol. 31. No. 1. 2017.

[7] Schmidt, Craig W., et al. "Tokenization Is More Than Compression." arXiv preprint arXiv:2402.18376 (2024).

[8] Balkenius, Christian, and Jan Morén. "A computational model of context processing." 6th international conference on the simulation of adaptive behaviour. 2000.

[9] Teehan, Ryan, Brenden Lake, and Mengye Ren. "CoLLEGe: Concept Embedding Generation for Large Language Models." arXiv preprint arXiv:2403.15362 (2024).