JOURNAL OF BASIC SCIENCE AND ENGINEERING

A NOVEL METHODOLOGY OF COMPLETING CODING EXERCISES ON OBJECT ORIENTED PROGRAMMING LANGUAGES BY LLMS TO DEVELOP SOFTWARE

Suvarnalata Hiremath[1] and C M Tavade[2]

[1]Research Scholar, Department of Computer Science and Engineering, Visvesvaraya Technological University (VTU), Belagavi, Karnataka, 590018, India.

[2]Professor, Department of Electronics and Communication Engineering, SGBIT, Belagavi. Visvesvaraya Technological University (VTU), Belagavi, Karnataka, 590018, India.
[1]suvarnahiremat@gmail.com; [2]cmttc1@gmail.com

Abstract - By dynamically proposing the most probable next action within a program's logic, based on the current contextual flow, Integrated Development Environments significantly accelerate the construction of software. The efficacy of three pre-trained language models in tackling multiple-choice assessments pertaining to introductory and intermediate computer science coursework was investigated. Short instructional sequences (instead of "codings snippets") were commonly included in these assessments, offered at the post-secondary level. The emergence of this novel technology has sparked widespread discourse surrounding its potential benefits, such as the generation of practice exercises and explanations of written instructions, in the context of computer science education. Concerns regarding its potential misutilization, such as facilitating fraudulent behaviour, have also been raised within this domain. Despite limited knowledge concerning their effectiveness in analyzing instructional sequences within educational contexts, research into the reasoning capabilities of GPT models remains sparse. An investigation was conducted utilizing various OpenAI models to assess their performance on multiple-choice evaluations (formative and summative) drawn from three introductory computer science courses employing Java, encompassing a total of 530 questions.

I. INTRODUCTION

This study assessed the ability of a specific large language model, txt-vinci-X, to answer MOQs or multiple-option queries that often-included short pieces of code. The MCQs were drawn from introductory and intermediate Python programming courses, and a dataset of 613 MOQs was manually compiled from three existing courses The research employed a combined and mixed approach of basic pattern recognition and manual review to classify the questions into coherent categories based on their nature (e.g., questions posing true/false statements or inquiries regarding the provided code snippet's output). An analysis was conducted on the Generatively Pretrained Transformers models' performance across these categories to discern if specific question types yielded more accurate responses compared to others.Additionally, a comparison was made

between the older InstructingGPT txt-vinci-A model and the newer GPT-3.5 txt-vinci-B and txt-vinci-C models to assess the progress achieved in recent years. This evaluation comes amid heightened public interest in the potential influence of GPT models on education, sparked by the recent release of OpenAI's ChatGPT1.

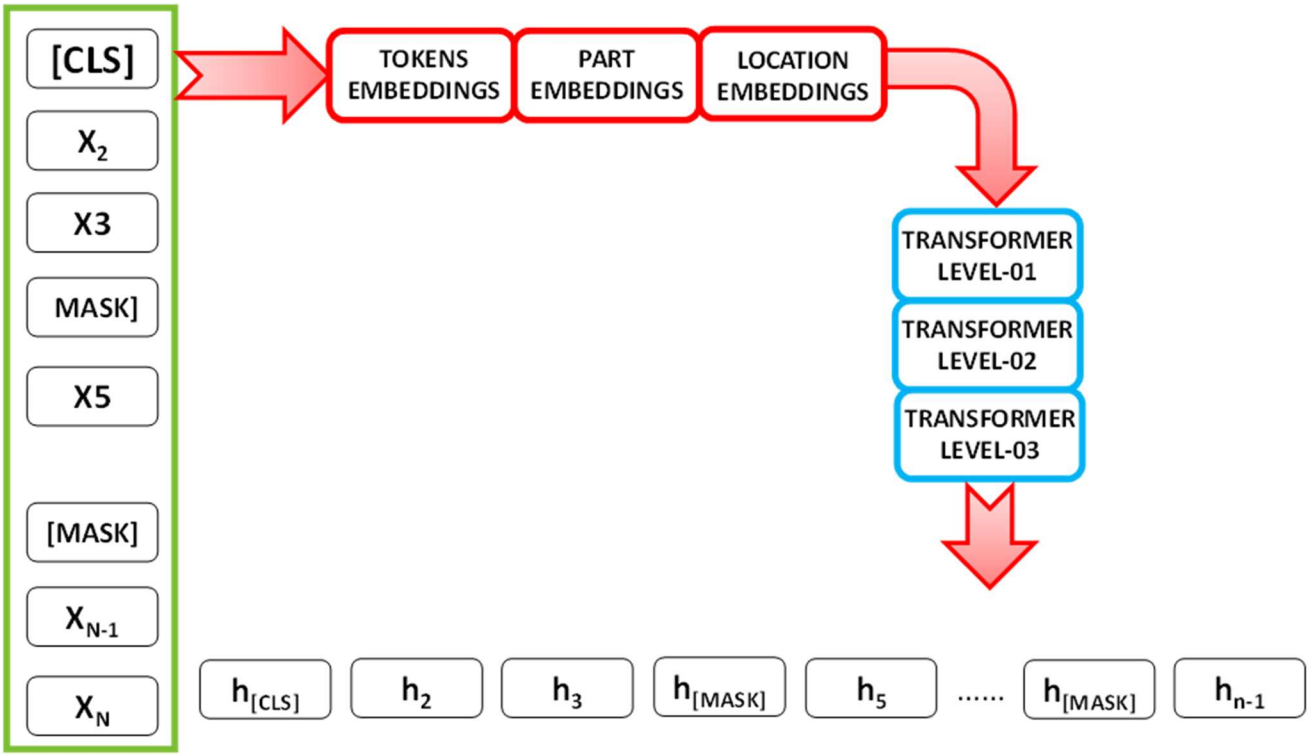

Figure 1 Language Frameworks with sharing variables for comprehension and generating of codes

As the sophistication and size of software development balloon, readily available collections of open-source projects offer an avenue to explore machine learning applications in source code designing[1]. This exploration often hinges on the recognition that source code shares characteristics with natural language, as it is constructed by and intended for human understanding, hence exhibiting similar statistical patterns [19].Leveraging statistical language models for source code modeling has proven beneficial for various software engineering tasks [19, 42, 46]. These include tasks like condensing code into summaries [23, 49], identifying identical or similar code segments [51, 52], and automatically fixing code errors [15, 47]. Notably, code completion, which suggests likely continuations based on existing code, has seen significant progress through this approach [17, 19, 27, 46].Concerns regarding potential misuse have led to restrictions on the tool's use in certain contexts. For instance, New York City public schools blocked it due to plagiarism and inappropriate content risks (Elsen-Rooney, 2023). Universities have responded by adapting assignments (Huang, 2023) and adopting detection tools like GPTZero (Bowman, 2023). Notably, OpenAI developed a similar tool. However, the effectiveness of these detection methods remains

unproven.A crucial feature of Integrated Development Environments (IDEs) is code completion. This feature accelerates software development by anticipating the most likely next element (token) based on the existing code. Recent advancements in deep learning have seen the application of Recurrent Neural Network (RNN)-based language models to understand and predict code patterns, contributing to this helpful feature [3, 27].

A) ***Masked bidirectional Language Modelling:*** Existing code completion methods struggle to accurately predict identifiers (meaningful names for variables or functions) because they hold valuable information for understanding the program. Therefore, creating representations for all program tokens, particularly identifiers, that capture both their context and general meaning would benefit both source code modelling and code completion. To achieve this, identifiers are masked within the programs, and the model is tasked with predicting these masked tokens based solely on the surrounding context.

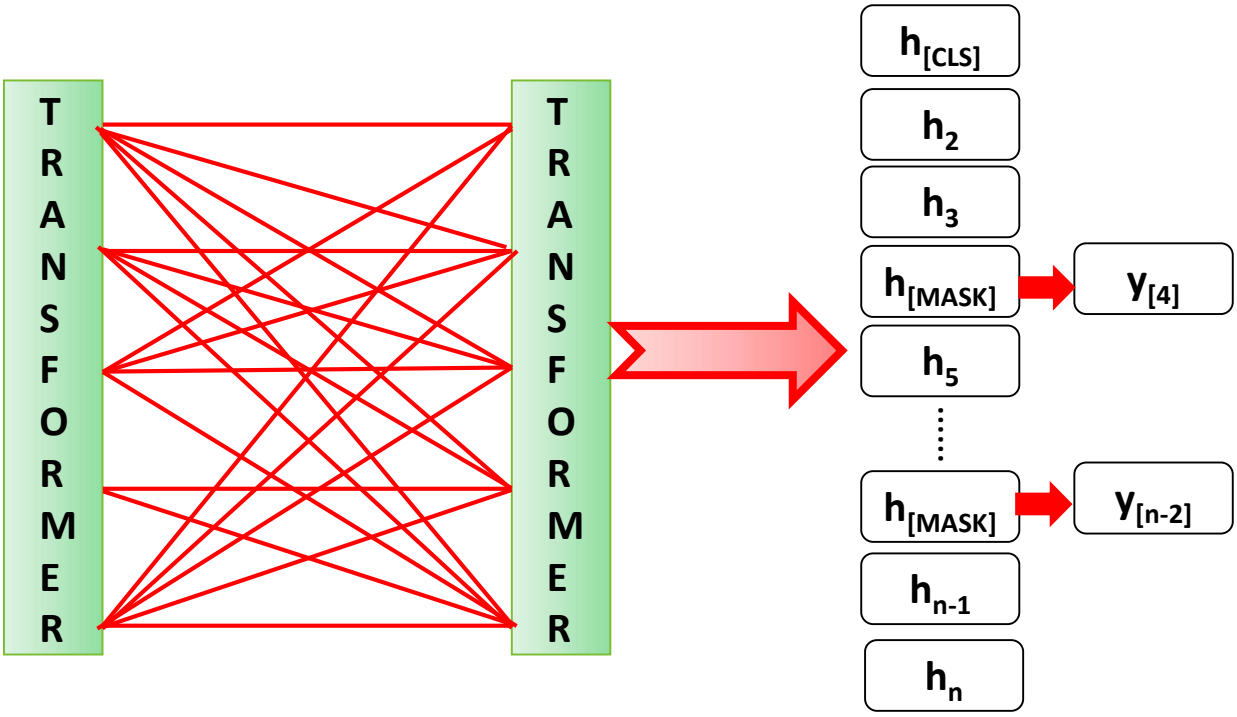

Figure 2 Two-directional Language Model (Masked)

B) ***Next Code segment Predicting:***This study posits that comprehending connections between code sections is crucial for effective source code modeling. To facilitate this, a pre-training process is implemented using a binary "next code segment prediction" task. This task involves determining whether two sequences of code tokens follow each other within a given code snippet.

C) ***Framework for Single-directional Language:*** This study incorporates a left-to-right language modeling task, where each token's internal representation reflects only the

information preceding it in the sequence. This training objective is crucial because tasks like code completion only access information from the left side of the current token.Once pre-trained, the model undergoes fine-tuning, where the pre-trained weights are adapted to the specific task of code completion. Instead of directly predicting the next token, a multi-task learning approach is employed. This approach involves first predicting the type of token (e.g., variable, function), and then using this predicted type to inform the subsequent token prediction.

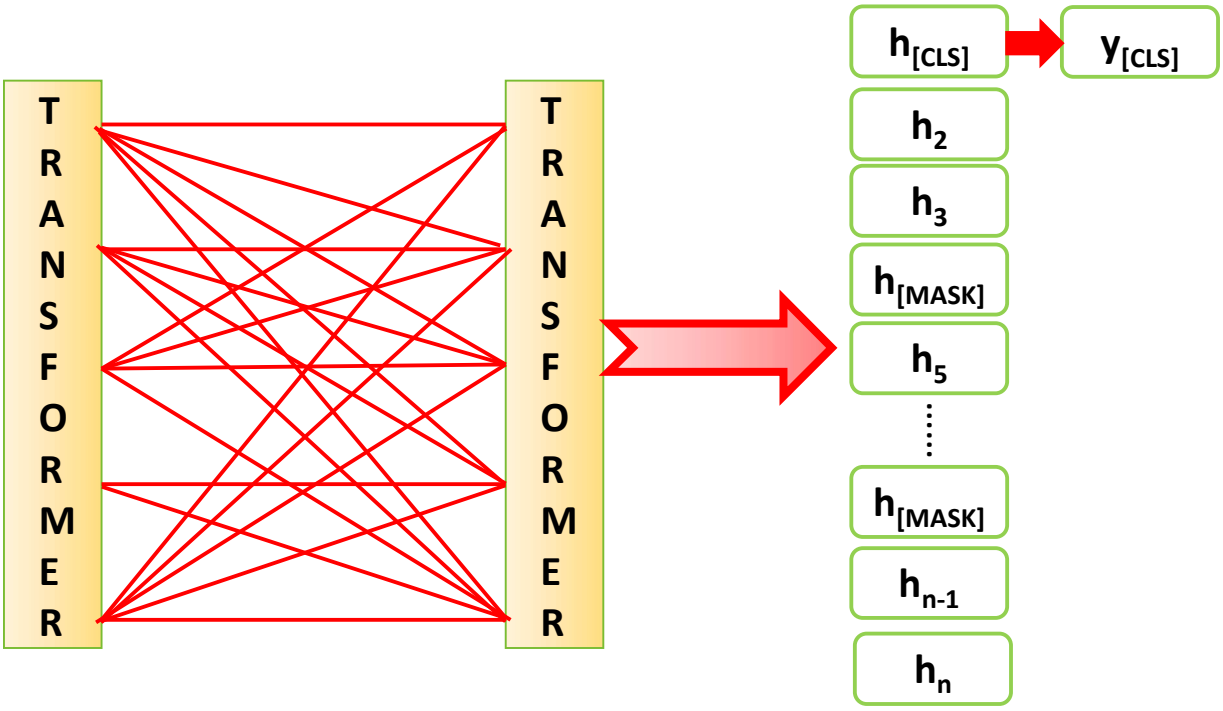

Figure 3 Predicting framework for following code portion

II. RELATED WORKS

A previous study (Savelka et al., 2023) assessed the ability of a specific GPT model (text-davinci-003) to answer various question formats, including multiple-choice questions (MCQs), within the context of real-world programming courses. The study revealed that, while not capable of excelling across the entire range of assessments typically encountered in introductory and intermediate Python courses (scoring below 70% even in basic modules), these GPT models, when used directly, could still assist learners in achieving a significant portion of the total available score (over 55%). Multi-task learning is a technique that promotes better performance on multiple related tasks by sharing what they have in common. It does this by drawing valuable clues from the training data of each task, leveraging their similarities [4]. Think of it like studying for multiple related exams together: you learn the core concepts shared

between them, leading to a deeper understanding overall. This shared understanding, represented by layers hidden within the learning model, helps identify common features across all tasks. Additionally, by focusing on solutions that benefit all tasks simultaneously, the risk of overfitting to specific details of one task is reduced, making the model more adaptable to new tasks in the future. This approach has proven successful in various fields, including natural language processing (understanding and generating human language), speech recognition (converting spoken words to text), and computer vision (analysing and interpreting images and videos) [10, 14, 31, 9, 32, 34]. A previous study (Savelka et al., 2023) identified a key weakness in GPT models: difficulty handling tasks requiring multiple logical steps. Additionally, the study noted a discrepancy in performance between multiple-choice questions (MCQs) with and without code snippets. This research delves deeper into this phenomenon, specifically seeking to pinpoint finer details of MCQ characteristics that pose challenges for GPT models.

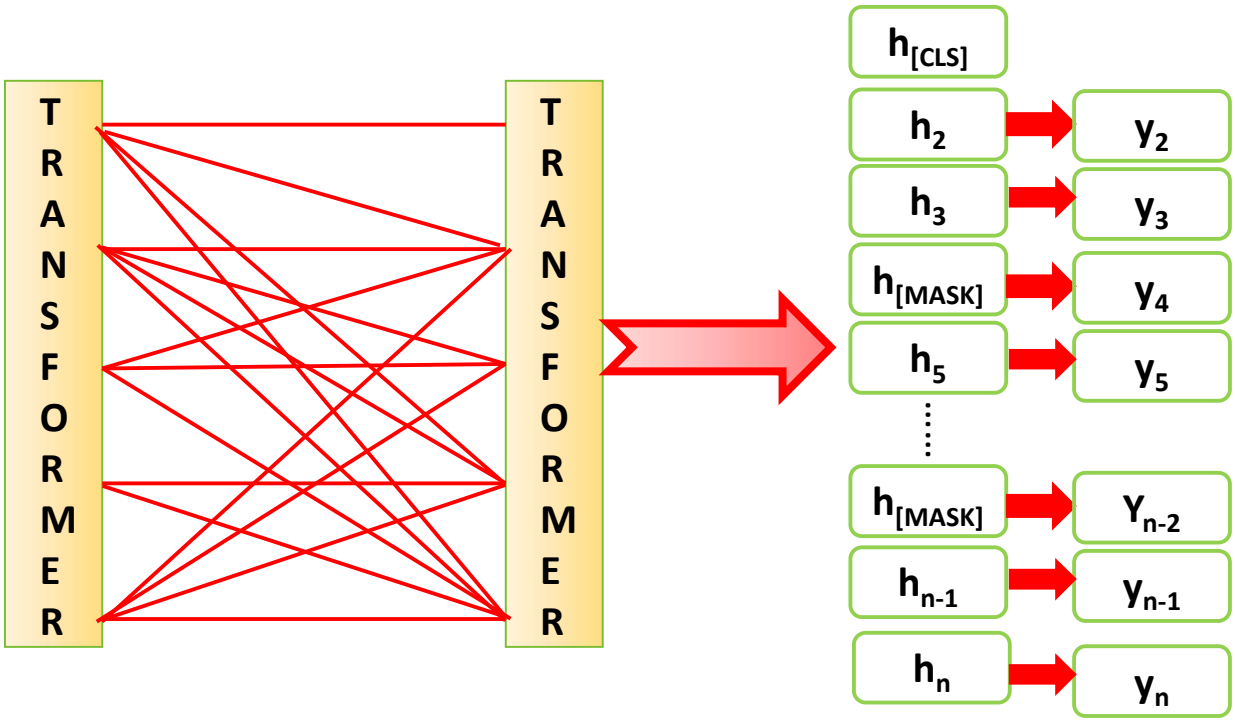

Figure 4 Language framework - single dimensional

This study investigates the performance of GPT models on programming-related multiple-choice questions (MCQs), a previously unexplored area. While past research evaluated GPT performance on MCQs in other domains, achieving varying success rates (including passing scores), no prior work specifically examined programming MCQs.For instance, Robinson et al. (2022) explored InstructGPT and Codex models on datasets requiring multi-step reasoning, memory, and reading comprehension (OpenBookQA, StoryCloze, RACE-m) and reported accuracy between 77.4% and 89.2%. Additionally, GPT models have demonstrated the ability

Vol. 21, No. 1, (2024)
ISSN: 1005-0930

to generate code for programming assignments in higher education, with Drori and Verma (2021) utilizing Codex to solve computational linear algebra MCQs in Python with 100% accuracy. Furthermore, studies have shown GPT models achieving around 50% accuracy on various MCQ-based exams like the USMLE, MBE, and AICPA REG exam (Kung et al., 2022; Gilson et al., 2022; Li´evin et al., 2022; Bommarito II and Katz, 2022; Bommarito et al., 2023).

III. MOTIVATION AND IDENTIFIED PROBLEM

A. **Language Framework (statistical):** These models analyze word patterns in languages, assigning likelihood scores to sequences based on how often they appear together. Sequences judged "natural" by native speakers receive high scores, while unnatural or incorrect sentence structures receive lower scores. Interestingly, programming languages also exhibit predictable patterns, making them suitable for analysis using these same models [19].

B. **Multiple-work Learning:**Multi-task learning combines training on several similar tasks to boost overall performance. Sharing information between these tasks helps the model identify common patterns and features, leading to better generalization on unseen data [4]. Think of it like studying for multiple related exams together - you learn the core concepts shared between them, leading to a deeper understanding overall.This shared understanding, represented by "hidden layers" within the model, allows it to focus on solutions that benefit all tasks simultaneously. This reduces the risk of overfitting to specific details of one task, making the model more adaptable to new tasks in the future. This approach has proven successful in various fields, like understanding and generating human language (natural language processing), converting spoken words to text (speech recognition), and analysing and interpreting images and videos (computer vision) [10, 14, 31, 9, 32, 34].

- **Previously-Trained Language Framework:** Prior training on massive datasets (pre-training) has proven effective in natural language processing (NLP), achieving top performance across various tasks [citations]. This success can be attributed to several benefits: (a) Universal understanding: Pre-training exposes the model to a vast amount of text, allowing it to grasp general language patterns applicable to various NLP tasks.(b) Strong starting point: The pre-trained model provides a solid foundation for learning specific tasks, leading to better overall performance.(c) Reduced overfitting: Pre-training acts like a safeguard against overfitting to small datasets, ensuring the model generalizes well to unseen data.(d) Two main approaches utilize pre-trained language representations for specific tasks:Feature-based: These methods leverage pre-trained representations as additional information for the task-specific model.Fine-tuning: This approach directly adapts the pre-trained model to the specific task by adjusting its parameters.

IV. PROPOSED SOLUTION

When given a program as a sequence of words (tokens), like "x1, x2, ..., xn", CugLM creates a unique representation for each word based on its context in the program.A sequence of tokens has been presented as $x = x_1, x_2, \ldots\ldots, x_n$. For every one of these tokens CugLMs are designed that gets vectors based representations that has a definite context in case of every token. A framework of models has been demonstrated over Fig. 2. An adoption of a N-level transformers like a framework of languages for encoding of vector of inputs like $y = y_1, y_2, y_3, \ldots\ldots, y_m$ in representing of context of differing layers like in $J^n = [j_1^n, j_2^n, j_3^n, \ldots\ldots\ldots, j_m^n]$ in which $J^n = Transformers[J^{n-1}]$, with $n \in [1, N]$. Over fig. 2 with following parts the omission of the superscripted L because of vectors that are hidden from ultimate levels j_k^N for the making of the pictures of reduced complication. In cases of every level of the transformer, multiple heads of attention have been utilized for aggregation of results of the penultimate levels. Along with this the result of self-attentions headings B_m are evaluated in the manner provided below:

$$R = J^{N-1}X_n^R, L = J^{N-1}X_n^L, \;\; W = J^{N-1}X_n^W$$

$$P_{jk} = \begin{cases} 0, permission\ of\ attending \\ -\infty, stop\ attendance \end{cases}$$

$$B_m = softMax\left(\frac{RL^U}{\sqrt{f^l}} + N\right)W$$

In which $J^j = R^{|y| \times f_i}$ demonstrates the j[th] layer output. Questions Q, key like L along with magnitudes like W have been evaluated through linear projections of the last level's results J^{n-1} utilizing variable matrix like X_n^R, X_n^L, X_n^W. $P \in R^{|y| \times |y|}$ represents matrices of masks which determine in case of a token pair might be sufficed by one another. In cases of varying before-training aims matrices of different masks like M for controlling of token of related contexts migh be attended by tokens in cases of computation of their demonstrations that are done over a particular context like shown in Figure 2. In cases of Language Models that have 2 directions contents of the matrices of mask have been initialized by zeros, that have the meaning of every token having connections with one another. In cases of 1 direction Language Model, above triangle like portion of masks are initialized to negative infinity showing every one of the tokens may connect the the token on the left with correct context and with thyself. Results of CugLM involves (a) for every one of the tokens that have been inputted a representation of vectors. (b) the presentations in [CLS] that represents the abbreviation of sequential demonstrations and might be utilized in cases of works of classifications. At the times of before-trainings, variables of a model

goes through the process of sharing and optimization for multiple aims like 2-directional Language Model, predictions of the part of Following Codes and 1 directionals Language Models. Following the before-training of these frameworks, fine-tuning might be performed in case of works that are in the lower streams. For the purpose of completing the codes fine-tuning of Cug Language Model is done by the authors.

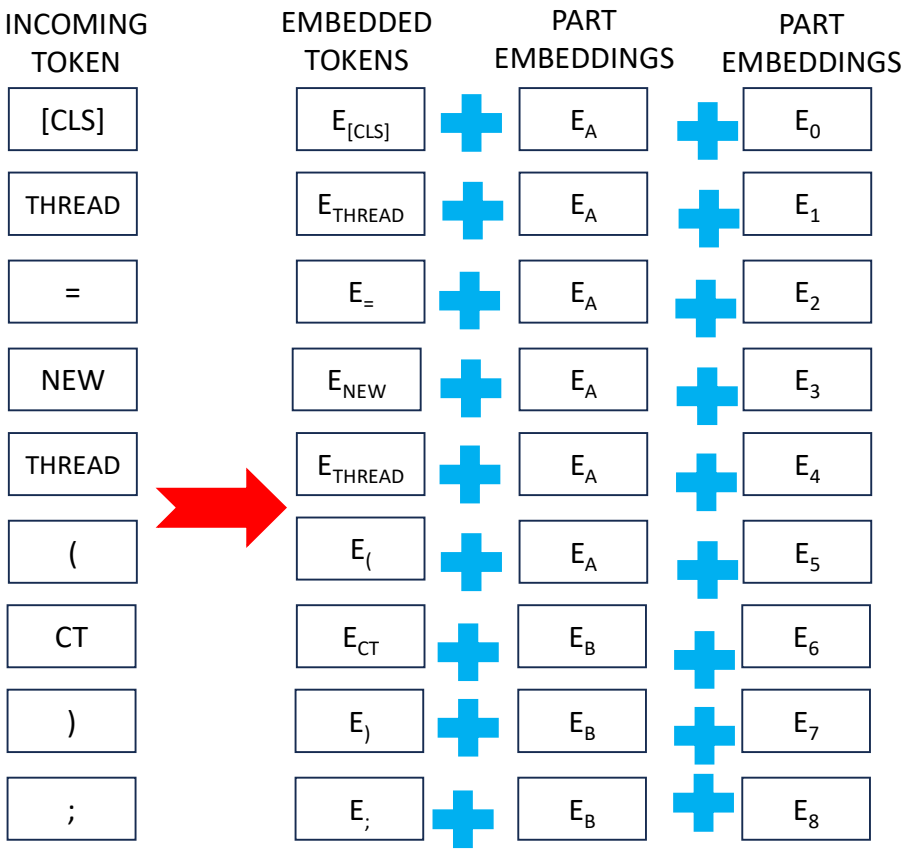

Figure 5 Demonstration of input information. Embedded inputs are addition of Embedded tokens and Part embeddings and embedded locations.

V. RESULTS AND DISCUSSIONS

The experimental outcomes are presented in Table 2.The GPT models' performance clearly outstrips that of the Jaccard similarity baseline, as expected.Among the models tested, txt-vinci-C emerged as the leader, achieving an impressive 65.5% overall accuracy. Following closely behind was txt-vinci-B, with a score of 64.5%. txt-vinci-A's overall accuracy (mention the specific score) falls short of the other two models, suggesting further investigation is needed.As txt-vinci-B and txt-vinci-C share a closer developmental background, their performance similarity makes sense.

However, txt-vinci-A's independent development trajectory likely contributes to its distinct performance level.OpenAI's Codex, released in 2021 by Chen et al., marked a significant leap forward in GPT-3's ability to handle computer code.

Query Sort	J-similarity baselines	txt-vinci-A	Txt-vinci-B	Txt-vinci-C
Without Coding				
Right/Wrong	**23 out of 50 (46 percent)**	**27/50 (54 percent)**	**29/50 (58 percent)**	**42/50 (84 percent)**
Recognize Right/Wrong statements	**11 out of 100 (11 percent)**	**15 out of 100 (15 percent)**	**48 out of 100 (48 percent)**	**62 out of 100 (62 percent)**
Completion Statements	**8 out of 37 (21.6 percent)**	**24 out of 37 (64.8 percent)**	**31 out of 37 (83.7 percent)**	**33 out of 37 (89.2 percent)**
Others	**15 out of 54 (27.8 percent)**	**25 out of 54 (46.3 percent)**	**43 out of 54 (79.6 percent)**	**51 out of 54 (94.4 percent)**
Net	**57 out of 241 (23.6 percent)**	**91 out of 241 (37.7 percent)**	**151 out of 241 (62.6 percent)**	**188 Out of 241 (78 percent)**

This groundbreaking model directly paved the way for the development of the subsequent txt-vinci-B.Txt-vinci-C tackles non-code MCQs with an impressive 77.9% accuracy, but its score drops to 59.5% when faced with code snippets, indicating a notable difference in its capabilities.With a 77.9% success rate on non-code MCQs, txt-vinci-C demonstrates its strength. However, its accuracy falls to 59.5% when encountering code snippets, revealing a potential area for improvement.We can be extremely confident that this observed difference isn't just random noise due to the statistically significant p-value of less than 0.0001.The presence of both code and natural language likely increases the information density and complexity of the input compared to natural language alone.There's a chance that, specifically in our setup, the average difficulty level of questions with code is higher compared to questions without code.Completion-oriented MCQs seem to be txt-vinci-C's strength, where it excels with a remarkable 87.1% accuracy. However, its broader capabilities, as measured by other question types, score 60.1%, indicating a potential area for further development. This difference is statistically significant ($p < 0.0001$). The observed disparity is demonstrably statistically significant, with a p-value well below 0.0001.Given their

specialization in prompt completion, it's unsurprising that GPT models perform well in this specific task, as seen in this finding.

Query Sort	J-similarity baselines	txt-vinci-A	Txt-vinci-B	Txt-vinci-C
Coding				
Right/Wrong	**9 out of 25 (36 percent)**	**23/50 (46 percent)**	**26/50 (52 percent)**	**31/50 (62 percent)**
Recognize Right/Wrong statements	**17out of 100 (17 percent)**	**19 out of 100 (19 percent)**	**45 out of 100 (45 percent)**	**41 out of 100 (41 percent)**
Filled-in	**7 out of 37 (18.9 percent)**	**12 out of 37 (32.4 percent)**	**24 out of 37 (64.8 percent)**	**35 out of 37 (94.6 percent)**
Completion Statements	**14 out of 54 (25.9 percent)**	**19 out of 54 (35.2 percent)**	**42 out of 54 (77.7 percent)**	**42 out of 54 (77.7 percent)**
Net	**47 out of 216 (21.8 percent)**	**73 out of 216 (33.8 percent)**	**137 out of 216 (63.5 percent)**	**149 Out of 216 (68.9 percent)**

VI. CONCLUSION

The assesments of workings of txt-vinci-A variants of GPTs utilizing vast collections of five hundred thirty multi-choice queries an important parts of those codingssnippets taken from 3 differing programming-based Python curriculum. The models of txt-vinci-C exhibits largest capabilities and exhibits a totality of accuracies of 65.5% compared with similarity baselines of Jaccard's of 23.4%. Although the achieved performance is commendable, there seem to be evident constraints.Initially, it seems that the multi-choice queries featuring code excerpts posed a slightly greater difficulty (59.5%) for the model compared to those lacking code (77.9%).Furthermore, multi-choice queries prompting completion of sentences or filling blanks seem to be addressed with significantly higher efficacy (87.1%) in contrast to alternative question formats (60.1%).Hence, the capacities of GPT models appear constrained when confronted with multi-choice queries related to computer code that demand reasoning extending beyond simple completing tasks (56.6%).Although our examination of GPT models' effectiveness across various MCQ types provided many valuable observations, it is bound by numerous constraints and offers ample opportunity for enhancement.

REFERENCES:

[1] MiltiadisAllamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018.A survey of machine learning for big code and naturalness. ACM ComputingSurveys (CSUR) 51, 4 (2018), 1–37.
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural MachineTranslation by Jointly Learning to Align and Translate. (2015).
[3] Avishkar Bhoopchand, Tim Rocktäschel, Earl Barr, and Sebastian Riedel. 2016.Learning python code suggestion with a sparse pointer network. arXiv preprintarXiv:1611.08307 (2016).
[4] Rich Caruana. 1997. Multitask Learning. Machine Learning 28, 1 (1997), 41–75.
[5] Hao Chen, Triet Huynh Minh Le, and Muhammad Ali Babar. 2020. Deep Learningfor Source Code Modeling and Generation: Models, Applications and Challenges.ACM Computing Surveys (CSUR) (2020).
[6] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio.2014. On the Properties of Neural Machine Translation: Encoder-DecoderApproaches. (2014), 103–111.
[7] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. InAdvances in neural information processing systems. 3079–3087.
[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, andRuslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyonda Fixed-Length Context. In Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019,Volume 1: Long Papers. 2978–2988.
[9] Li Deng, Geoffrey E. Hinton, and Brian Kingsbury. 2013. New types of deep neuralnetwork learning for speech recognition and related applications: an overview. InIEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013. IEEE, 8599–8603.
[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT:Pre-training of Deep Bidirectional Transformers for Language Understanding. InNAACL-HLT (1). Association for Computational Linguistics, 4171–4186.
[11] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, JianfengGao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pretrainingfor natural language understanding and generation. In Advances inNeuralInformation Processing Systems. 13042–13054.
[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, MingGong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT:A Pre-Trained Model for Programming and Natural Languages. arXiv preprintarXiv:2002.08155 (2020).
[13] Philip Gage. 1994. A new algorithm for data compression. C Users Journal 12, 2(1994), 23–38.
[14] Han Guo, Ramakanth Pasunuru, and Mohit Bansal. 2018. Soft Layer-SpecificMulti-Task Summarization with Entailment and Question Generation. In Proceedingsof the 56th Annual

Meeting of the Association for Computational Linguistics,ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers. Associationfor Computational Linguistics, 687–697.
[15] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. Deepfix: Fixingcommon c language errors by deep learning. In Thirty-First AAAI Conferenceon Artificial Intelligence.
[16] Vincent J Hellendoorn, Christian Bird, Earl T Barr, and MiltiadisAllamanis. 2018.Deep learning type inference. In Proceedings of the 2018 26th acm joint meetingon european software engineering conference and symposium on the foundations ofsoftware engineering. 152–162.
[17] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networksthe best choice for modeling source code?. In Proceedings of the 2017 11th JointMeeting on Foundations of Software Engineering. 763–773.
[18] Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochasticregularizers with gaussian error linear units. (2016).
[19] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar T. Devanbu.2012. On the naturalness of software. In 34th International Conferenceon Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland. IEEE Computer Society, 837–847.
[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory.Neural Computation 9, 8 (1997), 1735–1780.
[21] Daqing Hou and DavidMPletcher. 2010. Towards a better code completion systemby API grouping, filtering, and popularity-based ranking. In Proceedings of the2nd International Workshop on Recommendation Systems for Software Engineering.26–30.
[22] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Finetuningfor Text Classification. In ACL (1). Association for Computational Linguistics,328–339.
[23] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code commentgeneration. In Proceedings of the 26th Conference on Program Comprehension.200–210.
[24] Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2019. PretrainedContextual Embedding of Source Code. arXiv preprint arXiv:2001.00059(2019).
[25] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, andAndrea Janes. 2020. Big Code!= Big Vocabulary: Open-Vocabulary Models forSource Code. ICSE.
[26] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. 2020. Code Predictionby Feeding Trees to Transformers. arXiv preprint arXiv:2003.13848 (2020).
[27] Jian Li, Yue Wang, Michael R. Lyu, and Irwin King. 2018. Code Completion withNeural Attention and Pointer Networks. In Proceedings of the Twenty-SeventhInternational Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19,2018, Stockholm, Sweden. 4159–4165.
[28] Chang Liu, Xin Wang, Richard Shin, Joseph E Gonzalez, and Dawn Song. 2016.Neural Code Completion. (2016).

[29] Fang Liu, Ge Li, Bolin Wei, Xin Xia, Ming Li, Zhiyi Fu, and Zhi Jin. 2019. A Self-Attentional Neural Architecture for Code Completion with Multi-Task Learning.arXiv preprint arXiv:1909.06983 (2019).
[30] Fang Liu, Lu Zhang, and Zhi Jin. 2020. Modeling programs hierarchically withstack-augmented LSTM. Journal of Systems and Software 164 (2020), 110547.
[31] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang.2015. Representation Learning Using Multi-Task Deep Neural Networks forSemantic Classification and Information Retrieval. In NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for ComputationalLinguistics: Human Language Technologies, Denver, Colorado, USA, May 31 – June5, 2015. The Association for Computational Linguistics, 912–921.
[32] Mingsheng Long and Jianmin Wang. 2015. Learning Multiple Tasks with DeepRelationship Networks. CoRR abs/1506.02117 (2015). arXiv:1506.02117 http://arxiv.org/abs/1506.02117
[33] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretrainingtask-agnostic visiolinguistic representations for vision-and-language tasks. InAdvances in Neural Information Processing Systems. 13–23.
[34] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, andRogério Schmidt Feris. 2017. Fully-Adaptive Feature Sharing in Multi-TaskNetworks with Applications in Person Attribute Classification. In 2017 IEEEConference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI,USA, July 21-26, 2017. IEEE Computer Society, 1131–1140.
[35] Rabee Sohail Malik, Jibesh Patra, and Michael Pradel. 2019. NL2Type: inferringJavaScript function types from natural language information. In 2019 IEEE/ACM41st International Conference on Software Engineering (ICSE). IEEE, 304–315.
[36] Robert C. Martin. 2009. Clean Code - a Handbook of Agile Software Craftsmanship.Prentice Hall.
[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013.Distributed representations of words and phrases and their compositionality. InAdvances in neural information processing systems. 3111–3119.
[38] Tung Thanh Nguyen, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen.2013. A statistical semantic language model for source code. In Joint Meeting ofthe European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, RussianFederation, August 18-26, 2013. ACM, 532–542.
[39] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, ChristopherClark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized WordRepresentations. In NAACL-HLT. Association for Computational Linguistics,2227–2237.
[40] Alec Radford, Karthik Narasimhan, Tim Salimans, and IlyaSutskever. 2018. Improving language understanding by generativepre-training. URL https://s3-us-west-2. amazonaws.

com/openaiassets/researchcovers/languageunsupervised/language understanding paper.pdf (2018).
[41] Veselin Raychev, Pavol Bielik, and Martin T. Vechev. 2016. Probabilistic model forcode with decision trees. In Proceedings of the 2016 ACM SIGPLAN InternationalConference on Object-Oriented Programming, Systems, Languages, and Applications,OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 -November 4, 2016. ACM, 731–747.
[42] Veselin Raychev, Martin T. Vechev, and Eran Yahav. 2014. Code completionwith statistical language models. In ACM SIGPLAN Conference on ProgrammingLanguage Design and Implementation, PLDI '14, Edinburgh, United Kingdom – June 09 - 11, 2014. ACM, 419–428.
[43] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid.2019. Videobert: A joint model for video and language representation learning. InProceedings of the IEEE International Conference on Computer Vision. 7464–7473.
[44] Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. 2019. Pythia:AI-assisted Code Completion System. In Proceedings of the 25th ACM SIGKDDInternational Conference on Knowledge Discovery & Data Mining. 2727–2735.
[45] Alexey Svyatkovskoy, Sebastian Lee, Anna Hadjitofi, Maik Riechert, JulianaFranco, and MiltiadisAllamanis. 2020. Fast and Memory-Efficient Neural CodeCompletion. arXiv preprint arXiv:2004.13651 (2020).
[46] Zhaopeng Tu, Zhendong Su, and Premkumar T. Devanbu. 2014. On the localnessof software. In Proceedings of the 22nd ACM SIGSOFT International Symposium onFoundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 -22, 2014. ACM, 269–280.
[47] Marko Vasic, Aditya Kanade, Petros Maniatis, David Bieber, and Rishabh Singh.2019. Neural program repair by jointly learning to localize and repair. arXivpreprint arXiv:1904.01720 (2019).
[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is allyou need. In Advances in neural information processing systems. 5998–6008.
[49] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, andPhilip S Yu. 2018. Improving automatic source code summarization via deep reinforcementlearning. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. 397–407.
[50] Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as adual task of code summarization. In Advances in Neural Information ProcessingSystems. 6563–6573.
[51] HuihuiWei and Ming Li. 2017. Supervised Deep Features for Software FunctionalClone Detection by Exploiting Lexical and Syntactical Information in SourceCode.. In IJCAI. 3034–3040.
[52] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and XudongLiu. 2019. A novel neural source code representation based on abstract syntaxtree. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 783–794.

[53] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J.Wu. The neuro-symbolic concept learner: Interpretingscenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.
[54] G. Marcus. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprintarXiv:2002.06177*, 2020.
[55] G. Marcus and E. Davis. *Rebooting AI: Building Artificial Intelligence We Can Trust*. Ballantine BooksInc., 2019.
[56] J. Menick, M. Trebacz, V. Mikulik, J. Aslanides, F. Song, M. Chadwick, M. Glaese, S. Young, L. Campbell-Gillingham, G. Irving, et al. Teaching language models to support answers with verified quotes.*arXiv preprint arXiv:2203.11147*, 2022.
[57] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking therole of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*,2022.
[58] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders,et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprintarXiv:2112.09332*, 2021.
[59] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz,M. Bosma, D. Luan, et al. Show your work: Scratchpads for intermediate computation withlanguage models. *arXiv preprint arXiv:2112.00114*, 2021a.

Vol. 21, No. 1, (2024)
ISSN: 1005-0930