Predictive Modeling of Human Developers' Evaluative Judgment of Generated Code as a Decision Process

Sergey Kovalchuk
Chebyshev Research Center
sergey.kovalchuk@huawei.com

Yanyu Li
ITMO University
268904@niuitmo.ru

Dmitry Fedrushkov
Chebyshev Research Center
fedrushkov.dmitriy1@huawei.com

Abstract

The paper presents early results in the development of an approach to predictive modeling of human developer perceiving of code generated in question-answering scenarios with Large Language Model (LLM) applications. The study is focused on building a model for the description and prediction of human implicit behavior during evaluative judgment of generated code through evaluation of its consistency, correctness, and usefulness as subjective perceiving characteristics. We used Markov Decision Process (MDP) as a basic framework to describe the human developer and his/her perceiving. We consider two approaches (regression-based and classification-based) to identify MDP parameters so it can be used as an "artificial" developer for human-centered code evaluation. An experimental evaluation of the proposed approach was performed with survey data previously collected for several code generation LLMs in a question-answering scenario. The results show overall good performance of the proposed model in acceptance rate prediction (accuracy 0.82) and give promising perspectives for further development and application.

1 Introduction

Today, large language models (LLMs) are widely applied in the practice of software development, with both general-purpose solutions like ChatGPT by OpenAI and solutions dedicated to code writing such as CoPilot by Microsoft. One of the important capabilities of LLMs is support of code generation in various scenarios (Lu et al., 2021; Zhong et al., 2022): bug fixing, code completion, question answering with code snippets, and many others. Still, practical implementation of solutions for these tasks reveals several fundamental issues related to the complexity of the software development domain and the specificity of human developers as solution users.

An important problem is the evaluation of the solutions. A straightforward approach for LLM output evaluation is linguistic metrics such as BertScore, BLEU, and others. Complex semantics and non-linearity of code structure lead to the development of code-specific metrics such as CodeBLEU (Ren et al., 2020), RUBY (Tran et al., 2019), and others. Nevertheless, the real-world application of such metrics shows significant limitation in LLM evaluation (Evtikhiev et al., 2023). Another approach is application of test-based evaluation of generated code with such metrics as Pass@k (passing tests with k generated answers). Still, application of such an approach remains limited due to the lack of tests and limited applicability of automatically generated tests to the real-world problem. The problem significantly influences the performance of LLMs in real-world complex projects, which is clearly seen in modern project-based benchmarks like SWE-bench (Jimenez et al., 2024), RepoBench (Liu et al., 2024), CoderEval (Yu et al., 2023), etc. The benchmarks show relatively weak performance even for state-of-the-art solutions. One of the best-known solutions for the evaluation problem is evaluation of LLMs with human-centered metrics like acceptance rate (AR). More complicated approaches may involve value, accuracy, and other human-centered metrics (Dibia et al., 2023). Still, involvement of human developers requires significant time and effort, with the involvement of multiple human users.

Another problem is proper understanding of real human developers roles, needs, intents, and expectations. The practical application and surveys of the developers applying LLM-based solutions in daily tasks reveal several important issues (Bird et al., 2022; Ernst and Bavota, 2022; Liang et al., 2023; Shi et al., 2024). Users often report a lack of personalization, efforts needed to understand generated code, differences between code generated by humans and by LLMs, etc. As a consequence, this

Proceedings of the Fourth Workshop on Bridging Human-Computer Interaction and Natural Language Processing (HCI+NLP), pages 120–128

leads to limited trust of developers (Wang et al., 2023), possible vulnerability in generated code (Risse and Böhme, 2023), weak performance in real-world issues (Jimenez et al., 2024), etc. On the other hand, investigations on interaction with CoPilot show that proper time to show suggestions (Mozannar et al., 2023, 2022) and configuration of interaction patterns (Wang et al., 2023) show possible increases in the acceptance rate of suggestions generated by considered intelligent assistants. A key open problem here is understanding how human developers perceive, comprehend, and evaluate code in proper context (Roehm et al., 2012). Formal structuring of project context is currently approached by many solutions (see, e.g., project-specific benchmarks mentioned above). But the context of the human mind in evaluative judgment of code generated by both humans and machines is weakly investigated.

Resolving the mentioned problems (human-level evaluation of code and human developer internal context representation) is limiting many directions in the application of LLM to software development. The list of directions benefiting from resolving the problem includes training and fine-tuning LLM for better code generation (e.g., with reinforcement learning with human feedback, RLHF); building a complete pipeline for software development (Hong et al., 2023); improving human developer experience through better selection of available actions in AI agents.

In the presented paper, we are focused on approaching the mentioned problems through modeling of human developer perceiving of the code obtained from generative LLM. With the data collected in the previous developers' survey, we've modeled key perceiving characteristics that influence developers' actions in code acceptance evaluation. The approach is based on the idea of sequential decision on accepting or rejecting considered information (code). Thus, the basic idea of human developer perceiving modeling is formulated as a Markov Decision Process (MDP). The paper presents early, still promising results of the ongoing study in human developer perceiving modeling.

The remaining paper is structured as follows. The next section briefly describes problem definition, background for this work, and data collection for the experimental study. Following, Section 3 presents key elements of the proposed approach to human developer modeling. Section 4 presents the results of the experimental evaluation of the proposed approach. Finally, the last sections conclude the paper and discuss possible further directions of research.

Figure 1: Elements of user interface for evaluation of code generated by LLM in question-answering scenario

2 Problem definition and background works

The problem of a human developer perceiving the code generated by an intelligent assistant (IA) such as CoPilot usually considered within some scenario (e.g. code completion, bug fixing, etc.). The developer posts a query to the IA and perceives the answer. In some cases, the query is proactively posted automatically, depending on the developer activity. IA answers with a block of information containing the answer, suggestions, explanation, etc. Within our work, we are narrowing to the scenario where the user asks a question in natural language to AI powered by LLM and expects a piece of code as an answer. According to the classification of the CodeXGLUE benchmark (Lu et al., 2021), the problem is Text-to-Code generation. The goal is to build a model for predicting a human developer's subjective evaluation and final AR for code generated with LLM as an answer.

The examples of practical problems being solved in such a way may be widely found in online forums where users ask questions to resolve some programming issues. One of the most popular forums for the software development domain is StackOverflow[1] (SO) which is also widely used

[1] https://stackoverflow.com/

as an original source for training and evaluation of LLMs. A common pattern for such questions is "how to..." (use API, implement algorithm, fix bug, etc.) where an expected answer is a piece of code. As a result, SO is widely adopted as a source for datasets and benchmarks building in Text-to-Code problem investigation: see, e.g., such datasets as CoNaLa (Yin et al., 2018), StaQC (Yao et al., 2018), SO-DS (Heyman and Cutsem, 2020), etc.

Within our previous study (Kovalchuk et al., 2022), we used the data originating from SO and containing queries to fine-tune and evaluate LLMs. We used two datasets for that purpose. First, we collected 42k pairs of questions (text) and answers (code) from SO in "conceptual" and "API usage" classes (according to (Beyer et al., 2020)) with answers shown as short code snippets in the Python programming language. Second, we used the publicly available CoNaLa dataset (Yin et al., 2018) with 2379 entries of similar structure. We used the datasets for fine-tuning of several LLMs (CodeGen, GPT) for further evaluation. Both queries and answers collected in the dataset were relatively short: the average lengths of queries and answers were 214 and 154 characters, respectively.

Next, we performed a survey with human developers evaluating the code generated by the fine-tuned models. We considered a set of seven different models applied to two datasets. Also, for reference, we considered answers generated by CoPilot as a reference industrial solution.

The survey was structured as a sequential evaluation of randomly selected pairs of questions (text) and answers (code). Figure 1 shows elements of the user web interface developed for this survey. The evaluation was performed with three criteria inspired by the theory of planned behavior (Ajzen, 1991) and includes evaluation of the following subjective perceiving characteristics:

- The general **consistency** of the code (whether the code is readable/understandable). The scale is considered to reflect how well the user understands the answer.
- The subjective correctness of the answer with respect to the proposed question. The scale is considered to reflect the user's **agreement** with the answer.
- The **usefulness** of the provided answer. The scale is considered to reflect the user's expected intention to use.

The selection of metrics reflects key categories of metrics for subjective evaluation of data quality according to (Wang and Strong, 1996): accuracy of data, relevancy of data, and representation of data, except for accessibility of data, which is beyond the considered scenario.

The evaluation was performed with a 5-level Likert scale (from -2 to $+2$). We collected the evaluations for 614 question-answer pairs from 42 developers of different levels, including MSc students in computer science, AI, and mathematical modeling, as well as junior, middle, and senior software developers (mainly working in the area of machine learning, data science). More details on dataset collection, methodology, obtained scores, and dataset analysis can be found here (Kovalchuk et al., 2022).

The analysis of the previously collected data showed that the human-centered metrics are weakly correlated with the linguistic metrics (including code-specific metrics) like BertScore, CodeBLEU, Ruby, etc. On the other hand, the collected metrics are well interconnected and may be considered as filters toward code acceptance. Seeing this empirical evidence as a motivation example, we focused on the development of a dynamic model of internal perceiving, evaluative judgment, and acceptance of software code, which is described in the next section.

3 Modeling human developer perceiving process

We propose the following conceptual approach for modeling human developer code perceiving (see Figure 2). The basic interaction loop involves a human developer posting a query to a code generation model, which answers with a code snippet. We use query Q and code context C as arguments that describe the external context of user decisions on answer accepting. In our experiment, Q is a natural language request with a short description of a problem to be resolved with generated code, C is code generated by LLM as an answer to the query and perceived by a user. Next, we consider user-specific information, which may include user profile, code repositories or artifacts authored by the user, personal skills, etc. The idea is to identify robust groups of users with similar perceiving behavior. We can use such information for identification of user personality, groups of similar users (e.g., via clustering (Kovalchuk and Ireddy, 2024)).

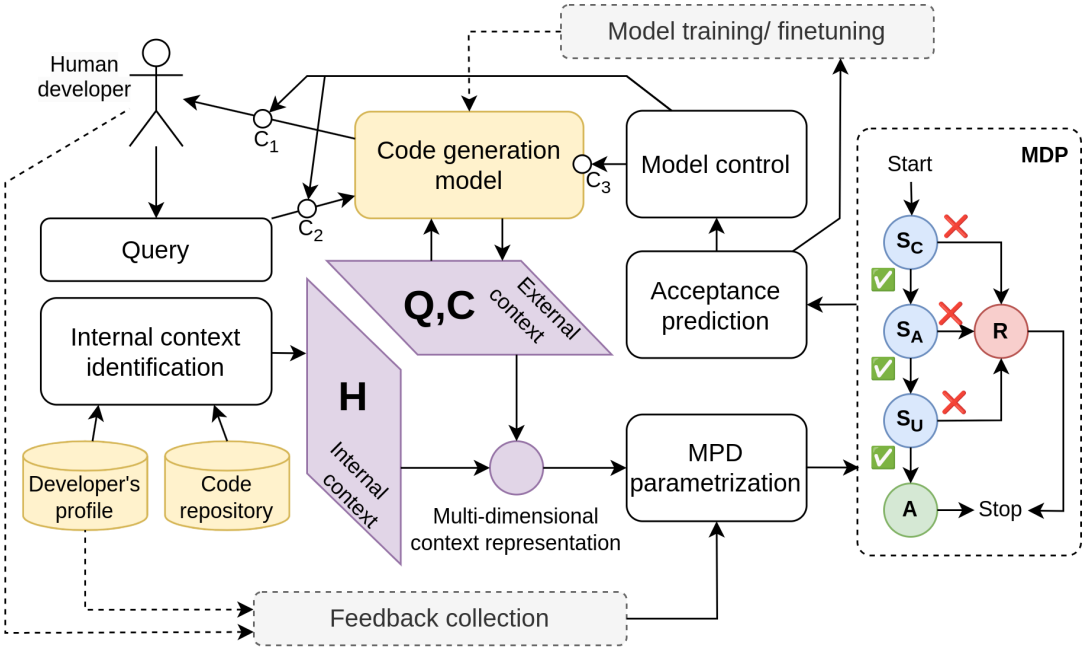

Figure 2: Human developer perceiving modeling with mixed context involvement

The size of such a group can vary from a single user (ultimate personalization) to large communities of groups working in the same area (e.g., "front-end developers"). Here, we see human-related characteristics H as representation of internal context. H can be considered as a way of flexible model personalization, where we can select different levels of model generalization (personal model per each developer, group level developer, or general model for a wider group of developers).

Next, we use $Q \times C \times H$ as an input to the perceiving model identification. The model can be used to predict AR, which, in turn, can be applied to control the code generation model in different ways. We can consider at least three such ways: filtering of the code generation model c_1 can be used to block unwanted and weak suggestions, improving overall user satisfaction of IA use; filtering of queries c_2 can additionally reduce the computational resource load, as running LLMs multiple times may be costly; internal code generation control c_3 may be applied directly during sequential code generation by blocking or re-weighting candidate tokens.

Additionally, we can consider offline procedures involved in the approach. First, we consider human feedback as an important source for model identification and parametrization. Second, the result model can be used for evaluation of generated code as an "artificial developer" assessing code and providing human feedback to the model (e.g., in the RLHF framework), which may enable significant scaling of the training/fine-tuning process with limited involvement of real human developers.

3.1 Decision process modeling

Within the presented work, we propose considering the human developer's perceiving of the code as a sequential MDP. In particular, we define three states where decisions are made $\{S_C, S_A, S_U\}$, two terminating states $\{R, A\}$ for rejecting the proposed code and accepting it, respectively, and two service states $\{Start, Finish\}$. The action space $A = stop, cont$ at each decision state includes two options with deterministic consequent transitions, namely, stopping the evaluation with following rejection and continuation of evaluation. The order of decision states is selected according to analysis of

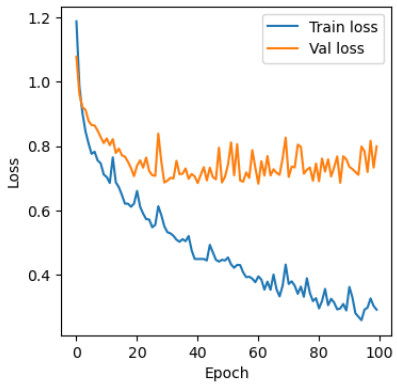

Figure 3: RGR model training

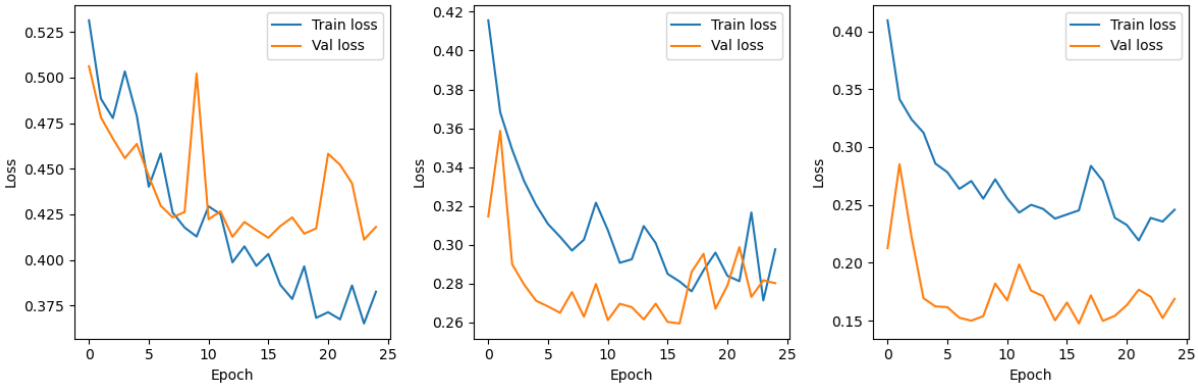

Figure 4: CLS models training for states $\{S_C, S_A, S_U\}$

previously collected data evaluation code for consistency/understanding (S_C), agreeing/correctness (S_A), and intention to use (S_U).

In this study, we are focused on the identification of the model that can be applied to the evaluation of generated code. We can consider the problem of learning from the demonstrated behavior of a human expert, which is widely resolved with the inverse reinforcement learning (IRL) approach (Arora and Doshi, 2021) aimed at the identification of reward function $\hat{R}_E$ from expert demonstration. In our case, we want $\hat{R}_E$ (and corresponding policy $\hat{\pi}$ inferred from the obtained reward) to be context-dependent, i.e., defined over the parameter space $Q \times C \times H$.

To identify the process with Likert scale based surveys, we define the switching threshold Th such that observed action is with $L(s)$ evaluation for state s will interpret as:

$$a_O(s, L) = \begin{cases} a_{stop} & \text{if } L(s) < Th \\ a_{cont} & \text{otherwise} \end{cases}$$

With this approach, we can transfer survey results into trajectories available for identification of reward function and corresponding policy in IRL fashion.

We are considering the effectiveness of two basic approaches to parameterizing MDP with obtained trajectories. The first approach is regression-based learning (RGR). We consider a task of learning a regression function $\hat{R} : Q \times C \times H \to \mathbb{R}$ such that $\hat{R}(s) \sim L(s)$. The inferred policy will be the selection of actions according to the rules $a_O(s, \hat{R}(s))$. The second approach is classification-based policy as a classification problem at each decision state (CLS). Here we learn a classification function $\hat{\pi}(s) : Q \times C \times H \to A$ with direct inference of action probability as a class.

3.2 Model identification with available data

First, we need to select a proper threshold Th to interpret our data. Table 1 shows the relative number of actions A_{cont} in the observed trajectories depending on the threshold. We consider $Th = 0$ as the main scenario also showing the most balanced action representation over trajectories. Still, we also consider other options $Th \in \{-1, 0, 1, 2\}$ (here $Th = -2$ is omitted as no a_{stop} actions were observed).

Table 1: Portion of a_{cont} decision depending on threshold Th

Th	S_C	S_A	S_U
-1	0.7818	0.6059	0.5863
0	**0.6808**	**0.5000**	**0.5049**
1	0.3453	0.2199	0.2248
2	0.2036	0.1156	0.0993

For both RGR and CLS approaches, we've implemented the machine learning solutions with a simple artificial neural network with one dense layer (128 neurons). In the RGR model, the output layer consists of 3 values for $\{L(S_C), L(S_A), L(S_U)\}$. In the CLS model, output layer depicts probabilities over $\{a_{stop}, a_{cont}\}$ set (models were trained separately for each decision state).

An important aspect of this experimental study is the influence of extended context with consideration of human personality. In the experiments, we consider three context spaces defined as embeddings in space $\mathbb{R}^N$. Q and C were defined as with embeddings obtained using the CodeBERT model by Microsoft (Feng et al., 2020) ($N = 768$). H

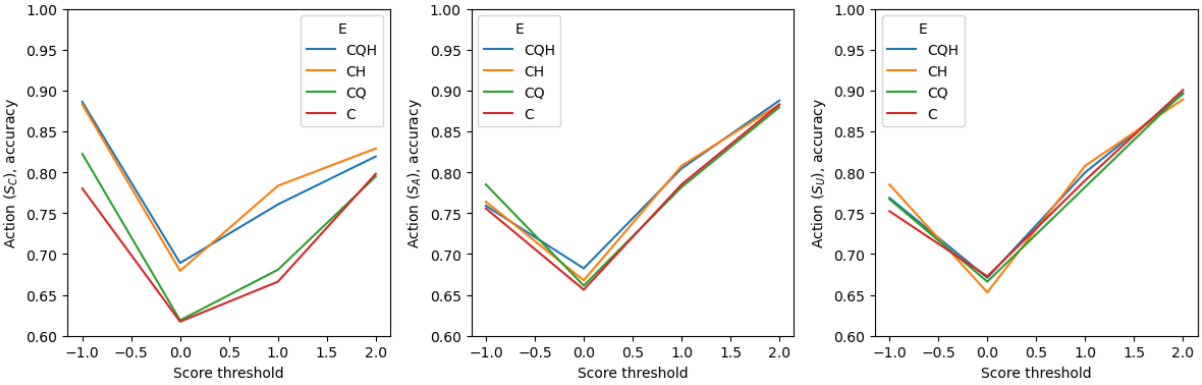

Figure 5: RGR model evaluation with different context embedding space E for states $\{S_C, S_A, S_U\}$

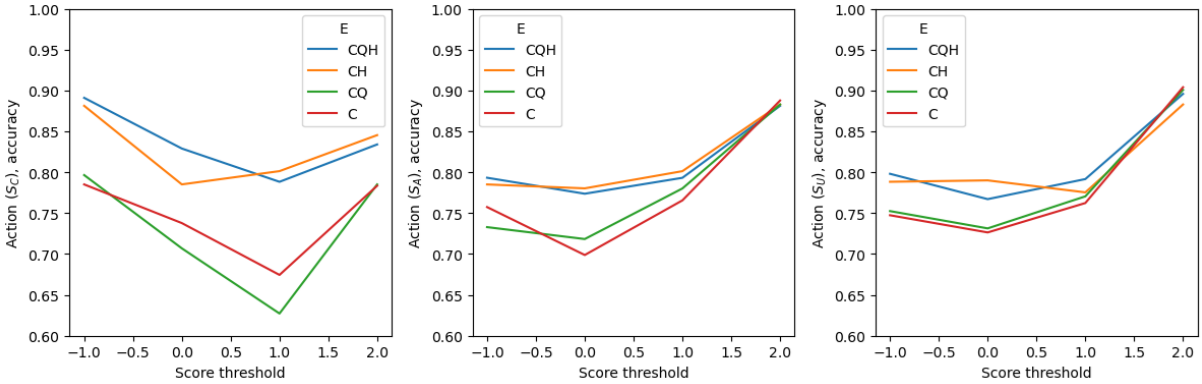

Figure 6: CLS model evaluation with different context embedding space E for states $\{S_C, S_A, S_U\}$

was encoded as one-hot embeddings for the users who participated in the survey ($N = 42$). We run the experiments with different combinations of embeddings E, namely $C \times Q \times H$ (all of them), $C \times H$, $C \times Q$, C (denoted as experiments CQH, CH, CQ, and C, respectively).

The loss function was selected as mean absolute error (MAE) for the RGR model and cross-entropy for CLS model. Figure 3 and Figure 4 shows training process for RGR and CLS models correspondingly. We selected the number of epochs for model training as 50 and 20 for RGR and CLS models, respectively, to get relatively stable validation loss without further decreasing.

For evaluation of MDP model performance, we performed 5-fold cross-validation with available survey data. The following two metrics were selected with averaging across the folds: accuracy of action prediction $Acc(s)$ in each decision state according to the classifier in the CLS model and according to $a_O(s, L)$ rules for the RGR model; accuracy of complete AR prediction $Acc(AR)$ estimated as correct prediction of reaching the terminating state in $\{R, A\}$ set.

4 Experimental evaluation results

4.1 Context influence analysis

The evaluation results for $Th = 0$ (the main scenario) are shown in Table 2. It can be seen that inclusion of H significantly increases the performance of both models in most of the states. The most significant increase is observed in action prediction in the consistency state (S_C), which can be interpreted as the state most influenced by the personal view of the user to the idea of code "consistency". For example, some users reported that the generated code included a correct answer but also contained syntactic errors, which leads to confusion in consistency evaluation. Comparison of RGR and CLS models shows significant outperforming of CLS compared to the RGR model. Although the RGR model provides more information (continuous space referring to the Likert scale), the provided accuracy is lower. E.g., it leads to an increase in AR prediction ($Acc(AR)$) by 20% (from 0.6889 to 0.8289).

Figures 5, 6, and 7 show evaluation of the developed models with different values of Th. Although the achieved accuracy is even higher, the main rea-

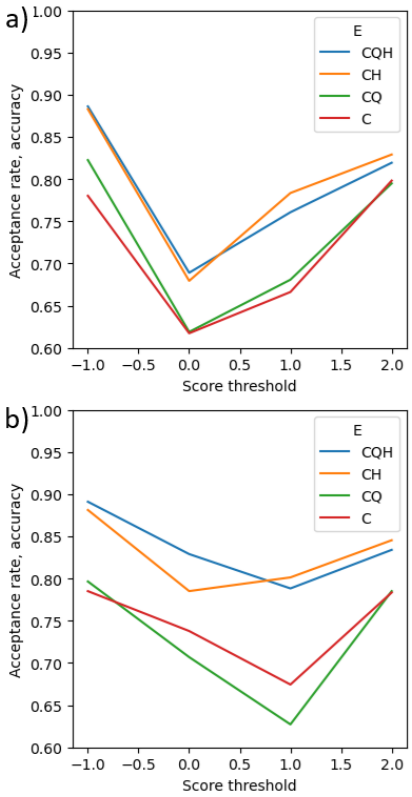

Figure 7: RGR (a) and CLS (b) model evaluation with different context embedding space E for AR

Table 2: Performance of the models with $Th = 0$

E	$Acc(\cdot)$			
	S_C	S_A	S_U	AR
Model: RGR				
CQH	**0.6889**	**0.6824**	**0.6709**	**0.6889**
CH	0.6794	0.6678	0.6531	0.6794
CQ	0.6189	0.6613	0.6662	0.6189
C	0.6173	0.6564	0.6727	0.6173
Model: CLS				
CQH	**0.8289**	0.7737	0.7670	**0.8289**
CH	0.7851	**0.7802**	**0.7899**	0.7850
CQ	0.7069	0.7182	0.7312	0.7069
C	0.7378	0.6987	0.7263	0.7378

son may be class imbalance. Also, all the observed tendencies are kept as well.

4.2 Code generation

One of the important parts of the model application is controlling code generation and model fine-tuning in order to increase result AR. While the previous experiments show good performance and can be further applied in filtering of LLM output (mentioned as c_1 control in Figure 2), deeper model control requires evaluation of generated information in advance (see control c_3 in Figure 2). For preliminary analysis of the applicability of our approach, we performed an experiment in the evaluation of code during the generation process. We used the CodeGen model by SalesForces (Nijkamp et al., 2023) and evaluated the proposed model with a reduced number of tokens. Figure 8 shows the prediction of the perceiving characteristics. We see that the MAE and STDe (standard deviation of error) of the prediction error obtained by the RGR model reached stable values approximately with 50 tokens, while the generated code in our examples may reach 100 or more tokens (with a considered limit of generation of 256 tokens). This evidence allows us to evaluate positively the applicability of the model in early detection of possible result rejecting by the human developer and stop or re-run generation.

5 Conclusion and future works

The paper presents early results in the investigation of human developer perceiving of code generated by LLM as an answer to an explicit or implicit query. With the MDP-based model, we showed higher performance in predicting acceptance of

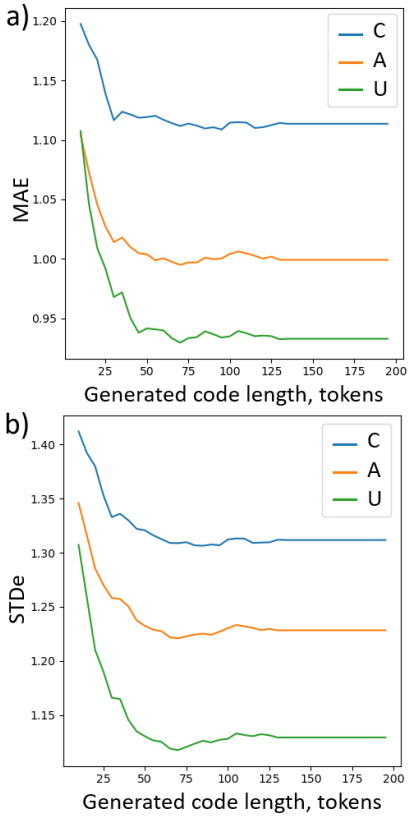

Figure 8: Prediction performance with a reduced number of tokens for states $\{S_C, S_A, S_U\}$

generated code by human developers. We see the results as promising evidence in the prospective application of structured human perceiving models with implicit internal context involved in the modeling. Moreover, the development of such models may be actively involved in practical application for LLM control. Additionally, we believe that the approach is generalizable and could be applied to different scenarios and various problem domains where the implicit internal context of an expert plays a role.

We see several directions for further development of the model and approach in general. First, we consider further development of the proposed approach and detailed investigation of methodological basis we used. For example, we are planning to extend and reconfigure the set of metrics we are using for more detailed representation of diverse metrics considered in the area of subjective information evaluation (Wang and Strong, 1996; Pipino et al., 2002). Also, we are aimed at the development of more detailed and structured representation of cognitive state and transfers between those states to extend the proposed basic sequential model. Within the experimental study, we are going to consider more realistic scenarios of human developer behavior with newly collected datasets or existing project-level datasets like (Mozannar et al., 2024; Chi et al., 2025). We are planning to perform more detailed analysis of internal context embedding space with possible dimensional reduction. For example, we can assess similarity between human developers and try to train a model for unobserved developers with a certain level of personalization. Next, we want to implement the mentioned control scenarios in order to increase LLM human-centered performance. In particular, the developed model can be considered as a "critic" model in the actor-critic machine learning approach in LLM (Gorbatovski and Kovalchuk, 2024). Finally, we want to evaluate more existing methods from the IRL field in order to identify parameters of the proposed MDP.

References

Icek Ajzen. 1991. The theory of planned behavior. *Organizational Behavior and Human Decision Processes*, 50(2):179–211. Theories of Cognitive Self-Regulation.

Saurabh Arora and Prashant Doshi. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500.

Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 2020. What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. *Empirical Software Engineering*, 25:2258–2301.

Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2022. Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools. *Queue*, 20(6):35–57.

Wayne Chi, Valerie Chen, Anastasios Nikolas Angelopoulos, Wei-Lin Chiang, Aditya Mittal, Naman Jain, Tianjun Zhang, Ion Stoica, Chris Donahue, and Ameet Talwalkar. 2025. Copilot arena: A platform for code llm evaluation in the wild. *Preprint*, arXiv:2502.09328.

Victor Dibia, Adam Fourney, Gagan Bansal, Forough Poursabzi-Sangdeh, Han Liu, and Saleema Amershi. 2023. Aligning offline metrics and human judgments of value for code generation models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8516–8528, Toronto, Canada. Association for Computational Linguistics.

Neil A. Ernst and Gabriele Bavota. 2022. Ai-driven development is here: Should you worry? *IEEE Software*, 39(2):106–110.

Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. 2023. Out of the bleu: how should we assess quality of the code generation models? *Journal of Systems and Software*, 203:111741.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint*.

Alexey Gorbatovski and Sergey Kovalchuk. 2024. Reinforcement learning for question answering in programming domain using public community scoring as a human feedback. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, page 2294–2296, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Geert Heyman and Tom Van Cutsem. 2020. Neural code search revisited: Enhancing code snippet retrieval through natural language intent. *Preprint*, arXiv:2008.12193.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Sergey Kovalchuk and Ashish Tara Shivakumar Ireddy. 2024. Prediction of users perceptional state for human-centric decision support systems in complex domains through implicit cognitive state modeling. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 46, pages 3257–3264.

Sergey Kovalchuk, Vadim Lomshakov, and Artem Aliev. 2022. Human perceiving behavior modeling in evaluation of code generation models. In *Proceedings of the Second Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 287–294, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Jenny T Liang, Chenyang Yang, and Brad A Myers. 2023. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pages 605–617. IEEE Computer Society.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024. Repobench: Benchmarking repository-level code auto-completion systems.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, and 1 others. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2022. Reading between the lines: Modeling user behavior and costs in ai-assisted programming. *arXiv preprint*.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2023. When to show a suggestion? integrating human feedback in ai-assisted programming. *arXiv preprint*.

Hussein Mozannar, Valerie Chen, Mohammed Alsobay, Subhro Das, Sebastian Zhao, Dennis Wei, Manish Nagireddy, Prasanna Sattigeri, Ameet Talwalkar, and David Sontag. 2024. The realhumaneval: Evaluating large language models' abilities to support programmers. *Preprint*, arXiv:2404.02806.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. *ICLR*.

Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. 2002. Data quality assessment. *Communications of the ACM*, 45(4):211–218.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Niklas Risse and Marcel Böhme. 2023. Limits of machine learning for automatic vulnerability detection. *arXiv preprint arXiv:2306.17193*.

Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How do professional developers comprehend software? In *2012 34th International Conference on Software Engineering (ICSE)*, pages 255–265. ISSN: 1558-1225.

Yuling Shi, Hongyu Zhang, Chengcheng Wan, and Xiaodong Gu. 2024. Between lines of code: Unraveling the distinct patterns of machine and human programmers. *Preprint*, arXiv:2401.06461.

Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. 2019. Does BLEU Score Work for Code Migration? In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 165–176, Montreal, QC, Canada. IEEE.

Richard Y. Wang and Diane M. Strong. 1996. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33.

Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. 2023. Investigating and designing for trust in ai-powered code generation tools. *arXiv preprint arXiv:2305.11248*.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, WWW '18, page 1693–1703. ACM Press.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*, pages 476–486.

Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2023. CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models. ArXiv:2302.00288 [cs].

Maosheng Zhong, Gen Liu, Hongwei Li, Jiangling Kuang, Jinshan Zeng, and Mingwen Wang. 2022. Codegen-test: An automatic code generation model integrating program test information. *arXiv preprint arXiv:2202.07612*.