User Evaluation of InCoder Based on Statement Completion

Marc Otten
Supervisors: Maliheh Izadi, Arie van Deursen
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

User Evaluation of InCoder Based on Statement Completion

Marc Otten
Computer Science and Engineering
Technical University of Delft
Delft, The Netherlands

***Abstract*—A lot of models have been proposed to automatically complete code with promising evaluation results when tested in isolation on testing sets. This research aims to evaluate the performance of these models when used by developers when programming. Are these models still useful for actual programming and do developers even want this functionality? The model evaluated in this study is the InCoder model by Facebook, specifically the ability to complete code statements for the Python programming language. To evaluate this a plugin called Code4Me was made for PyCharm and VSC that will show code completion suggestions from the model when a keybind is pressed or a trigger point is encountered. If the user is shown a suggestion the plugin will send the actual line of code made by the developer after a delay, this can also be the suggestion itself if the user thought the suggestion was correct. When the user has used the model sufficiently they will also be asked to fill in a survey to gather opinions on the functionality that the model provides. The results show that there is a 21.95% ExactMatch, 52,73% edit similarity, and a BLEU-4 score of 36.05 for the statement completion functionality of InCoder. All users that filled in the survey preferred the automatic suggestions on trigger points but some indicated the keybind functionality was also useful. If the suggestion shown to the user was good they would use it instead of typing it themselves. The users indicated that the suggestions were a lot or a bit better than the default suggestions and using the plugin did save time programming. Overall all users were positive of the performance and thought the statement completion functionality provided useful suggestions.**

***Index Terms*—code completion, InCoder, statement completion, machine learning, integrated development environment, software tools**

I. Introduction

Thinking of a solution to a problem takes time, but then programming it will take even more time. Using current state of the art natural language processing models specifically trained on code instead of natural languages we can significantly reduce the time spent programming by automatically completing tokens, statements, and even entire blocks of code. These models already exist and perform quite well when tested using testing sets but have not been tested by users in actual real life programming environments. Similar studies have tested similar functionality of models such as natural language prompts to code which users thought provided useful functionality. The architecture to collect data from the IDE's and evaluate has also been adapted from earlier studies. This project will evaluate the InCoder model made by Facebook by doing a user study and focus on if users use this kind of functionality and if the performance is still good in real life programming environments [1]. The model is trained to do code infilling but by simply taking the first statement of this infill we are able to evaluate the statement completion functionality of the model. Due to the model being bidirectional model it will use the left and right context when making a completion, which allows it to generate code at arbitrary places in the code. This is ideal for developers since this does not limit the use of the model if the model would only be unidirectional. The version used in this research is the 1B version due to the technical limitations of the server being unable to use the 6B version due to memory constraints. The evaluation will be done only on the Python programming language since it is one of the most used programming languages but InCoder supports a wide range of languages. The model will be used in combination with an IDE plugin to show the user the suggestions from the model when prompted or when a trigger point is typed. The plugin works for VSC and all JetBrains IDE's, but since the study is limited to Python PyCharm and VSC are the main focuses. These suggestions will be evaluated against the actual code the user writes and the usefulness will be evaluated using a survey. The results of this study show that the performance of the InCoder model is still quite good when used for statement completion and users indicated in the survey that it provides useful suggestions and saves time programming. They also indicated the functionality was useful and they would want to keep using the plugin.

The main contributions of this study are:

- User evaluation of InCoder for statement completion
- User opinions on functionality of statement completion for coding
- Trigger point evaluation to optimise statement completion performance
- Open source repository for the VSC and JetBrains Code4Me plugin for code suggestions

The study will look into the performance of automatic code completion models when used by developers by firstly discussing what has already been done in the field of automatic code completion. Afterwards the layout of the experiment and what design choices have been made to accurately evaluate the model and user perception are mentioned. Finally we will show and discuss the results of the research and what conclusions can be concluded from these results.

II. Problem Definition

All the models look promising when used in a testing environment but are they as accurate when used in real programming environments? Most models work by masking a single token in the code and then predicting what this token should be, however this functionality is already built in most IDE's. Most models only use the context from the left side of the mask to predict what the mask should be. However this method can discard useful context on the other side of the token which occurs quite often when actually programming. The new state of the art model by Facebook called InCoder is able to infill code in arbitrary positions by making use of the right context as well[1]. Since this is a new approach it has not been evaluated by users when programming but only on testing data. This evaluation will give critical insight on the performance of the model and if users find the functionality provided when programming useful.

This paper will evaluate the performance of this model by making a plugin that will suggest a statement completion using the model when a keybind is pressed or a trigger point is encountered. The plugin will keep track of how many times a user used the autocomplete function and if the completion was used by the developer. It will compare the suggestion of the model to the actual line of code that the developer ended up with. Surveys will also be used to collect additional data to more accurately evaluate the performance of this new State of The Art model.

III. Background

Natural Language Processing models have been used for a variety of things such as: chat bots, translating, text analysis, and more. With the increase of computing power and available data some models have been designed to specifically work on code instead of text. The structure of code is quite different to languages with abstract syntax trees, inheritance, and unlimited vocabulary. In contrast to languages that have a non-infinite set of words variables and method names can be named anything. By making use of these characteristics and training NLP models using code examples these models have been quite successful on different code completion tasks.

The InCoder model makes use of the left and right context of data. Most state of the art models only use the left context which limits their capabilities since some context can be missing such as entire methods at the bottom of the file. By using the right context aswell the InCoder model was able to improve the accuracy of code infilling and code synthesis [1]. The right context can contain just as much information as the left context so including it gives the model more information to use for predicting.

There are 4 different actions a user can do when a suggestion is shown: explicit select, typed select, explicit cancel, typed cancel [2]. The data stores which prediction was chosen and if a prediction even was chosen which allows us to differentiate between explicit select and typed select. The canceled action is not used in this study. If no suggestions was selected we can still calculate the metrics by comparing the typed code with the suggestion because the suggestion not being chosen does not mean it was a bad suggestion, the user might already know what they want to type.

IV. Related Work

This section discusses other studies that have been used to gather insight in the field of code completion, the different metrics, strategies to represent data for model input, and similar works with regards to IDE's and personalized suggestions.

A. Training Models

There has been a lot of development around pre-trained language models but it is not that clear why these models works so well and what features of the code structure they are able to use [3]. This study by Wan et al. found that these models were able to preserve the syntax structure in intermediate layers and induce these syntaxes from the code. The abstract syntax trees thus play a big role in these models and can be very helpful when pre-training.

When training a model using non-IDE, non-autocompleted, and different-language example code sequences the performance is slightly increased not only when testing but also in the real coding environments [4]. By training a model on these different examples and not only Python code Zhou et al. were able to show increased accuracy when compared to standard pre-training.

These improved accuracies are nice on paper but not all models perform quite as well when used by developers. By training the model on real world data instead of artificial autocompletion examples [5]. Gareth et al. were also able to show an increase in accuracy when the model was tested on real programming environments by developers.

A study by Hadi et al. showed that when pre-trained models were trained using specific annoted api requests these models outperformed generating of api requests by models that treated the data as words without annotation [6]. This shows that these models are very versatile and when trained with the right data can automatically complete statements or even generate correct code blocks.

B. Designing Models

There are a lot of different techniques used for automatic code completion. Such as a the use of abstract syntax trees of the code to better understand the structure. The work of Wang et al. have proposed an even more efficient method to use AST's [7]. By flattening the AST and then transforming it into a graph they were able to better utilize sequential and repetitive patterns which allowed for better accuracy in predicting the next token.

Different developers program with their own unique style and design patterns. This can lead to a model suggesting a good completion of the code but the developer not using this suggestion since it does not fit their style. A new model called PERSONA was made by Nguyen et al. to try and use personal projects from developers as training data for the model [8].

The model is able to recommend the next identifier with top-1 accuracy of 60-65%.

The accuracy of SOTA models is quite high when tested but the compilability of the generated code can be significantly increased as show by a study by Wang et al.[9]. By utilizing compiler feedback and language model fine tuning the code completion compilation rate went from 44% to 89%, this however does not mean that there was also an increase in accuracy for correct code completion.

C. Evaluating Models

The work by Bibaev et al. proposed ways to use the IDE to collect anonymous logs to train a model [2]. A server was used to collect the logs to have a centralised data storage. They also explained the different types of actions a user can do when a suggestion is shown. The study used these logs to train a model but for this study the architecture and type of actions are the key topics.

A similar study to this one is the work of Xu et al. [10]. A user study was done by evaluation a code generation and a code retrieval model based on a natural language prompt. Similar to this work a plugin was also made to allow users easy usability of the functions. To evaluate the models users had to do specific tasks and fill in a survey. It was shown that users liked the functionality but sometimes struggled writing their conceptualized ideas into text that allowed for a good suggestion. It also limited the tasks the users could use the model for.

The evaluation of code using BLEU is widely used for state of the art models. Some scenarios however require a slightly altered version to more accurately evaluate which can be achieved by using different smoothing functions. Chen et al. proposed and evaluated 7 different smoothing techniques to more accurately represent human judgement [11].

The work by Shi et al. showed how preprocessing affects BLEU score and what smoothing functions are a good fit [12]. It also highlights some issues for some specific smoothing functions. The differences for corpus level and sentence level calculations are also highlighted and discussed.

A plugin like this one is one made by Svyatkovskiy et al. called IntelliCode [13]. Instead of trigger points they showed a suggestion by the model on every keystroke and did not evaluate the model using user data but a testing set. They showed that the plugin was quite succesful when tested on a testing set. The metrics they used were finetuned for this specific task so provide valuable insight which metrics we can use in this study.

This study will contribute evaluation on user provided data when used in programming tasks that they normally encounter when programming. Participation was voluntarily and there was no limits on what tasks the users could perform with the plugin. The strategy of using trigger points to show users a suggestion will also provide valuable information on model performance in different situations.

V. Methodology

To evaluate the performance of the InCoder model we need to gather data from users. This data will be automatically collected when the users make use of the plugin. After the user has used the autocomplete function sufficient times to have enough reference points they are asked to fill in a short survey. The data collection for the paper will approximately take 1 to 2 weeks and the survey will more accurately poll their views of the model and plugin. The plugin and server will keep operating after the research is fnished to collect even more data which can be used for future studies.

The IDE's that the plugin is made for is VSC and JetBrains, specifically the Python version called PyCharm. This is because we have the most experience in these IDE's, are the most used IDE's for Python, and support custom plugins or extensions. The plugin makes a request to a server from TU Delft that allows for faster completion results than running locally on the users computer. This structure removes all the load from user's hardware and makes sure all users experience similar inference time. The suggested completion of the model and the final version of that statement which is what the user meant to program is stored in the database and then evaluated using different metrics. The metrics used are: Levenshtein Distance to compute edit similarity, Exact Match, ROUGE-L, BLEU-4, and METEOR, this is because these metrics are widely accepted and used in the natural language processing field which automatic code completion falls under.

The inference was drastically lowered before releasing the plugin to the users. When first testing the model the inference time was around 5s but after working on the plugin it took a little bit more than 1 second. This was partly due to the increase in computing power by using the server provided by the TU Delft, but also due to some improvements in the code. The model is very powerful and would actually generate multiple lines at once so after adding a custom stopping condition that would stop when one line was completed the inference time went even lower to 80ms. This low inference allows for almost no little waiting time for the user to be shown the suggestion which improves user experience and allowed for an the release of the plugin.

VI. Experiment Design

This section will discuss the research questions and the setup used in the research to answer these. Furthermore the approach of these research question will be evaluated and what metrics will be used. The last subsection will discuss the users and how they are acquired and the distribution of the plugin.

A. Research Questions

The research questions aim to give insight in the usefulness of statement completion which is the main topic.

RQ$_1$: *What is the acceptance rate of suggestions from the model when used by developers?*

This question will give insight in how the functionality of the plugin is used and if users even want it. The acceptance rate can be calculated by checking if the user chose the suggestion

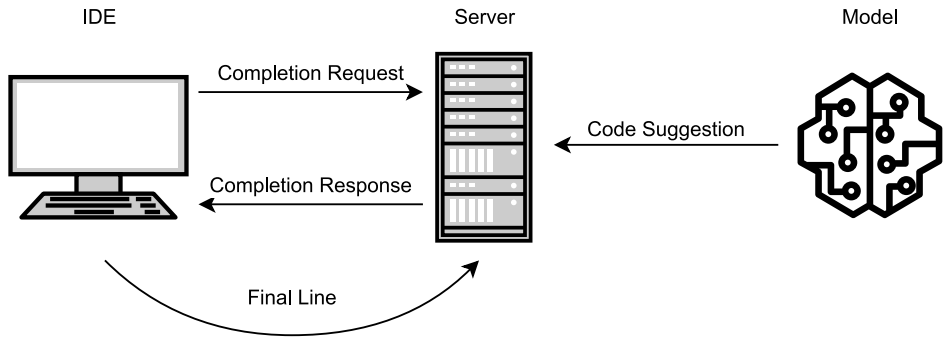

Fig. 1. Diagram of communication between plugin and server.

or not. It is also interesting to look if there is a significant difference in the accuracy of the model when a suggestion is accepted or not to know if the quality of the suggestion or something else is an important factor for acceptance rate. Another interesting thing is if the acceptance rate is different for different trigger points, some might be easier to complete for the model than others.

RQ$_2$: *How useful is the model from the users' perspective?*

The usefulness of the model will show how users used the functionality and if desired functionality might be lacking. The usefulness will be calculated using the different accuracy metrics but also using the survey. The survey will give insight into the perceived usefulness by the users. The accuracy might be quite low but users can still think the few good suggestions are very useful and outweigh the bad suggestions.

RQ$_3$: *How can the acceptance rate of the model be improved for everyday coding use?*

This information will be useful for further research. Some trigger points could be removed to increase user experience, users might have very good suggestions for the use cases in the survey, a model that works better on less context, sending more context, the keybind might not be a useful addition in the eye of of users, and other ideas can be explored here using the available data. The more users participate the more data the better the functionality of these models in the future.

B. Prediction Setup

Since this study focuses on statement completion and InCoder is able to do infilling of arbitrary length we added the stopping condition to stop completing on a newline character to improve inference time and make sure one line is completed. The model can handle up to 2048 tokens and to make sure there is room to infill tokens we limited the left and right context to 1000 tokens each. The the average token length of a python token is 3.8 characters [1]. We used this to estimate how much characters to send to the server and then server side limited the tokens before feeding it to the model. Since left and right context were both capped at 1000 tokens the model was always able to fill in at least 48 tokens.

[1]This can be shown by calculating the token frequencies on the raw files used in the P1K-22 dataset

The trigger points used to automatically prompt a suggestion are shown below, these were the most beneficial tokens to complete on [14]. The plugin checks after every insertion or deletion if the token before the cursor is a keyword and shows a suggestion if that is the case. If a suggestion is shown location in the file is tracked and the final line is send to the server to compare against the suggestion.

> DOT, AWAIT, ASSERT, RAISE, DEL, LAMBDA, YIELD, RETURN, EXCEPT, WHILE, FOR, IF, ELIF, ELSE, GLOBAL, IN, AND, NOT, OR, IS, BINOP, WITH, ;, ,, [, (, {, ~, =

C. Evaluation Metrics

Since the suggestion made by the model and the ground truth is stored we can easily evaluate the performance of the model. To do this we make use of several widely used metrics in the natural language processing field. For edit similarity we make use of the Levenshtein distance. The Levenshtein distance calculates the amount of single character insertions, deletions, or modifications. The Exact Match metric is a binary metric that checks if the things to compare are exactly the same. For accuracy, precision, and f1 score we use the ROUGE metric, specifically the ROUGE-L metric which takes the Longest Common Subsequence to calculate the score[15]. By using the LCS in the calculation it puts emphasis on the biggest overlapping part which indicates a higher similarity. This version is better for checking the similarity of code due to the nature of tokens [13]. The main metric that is used for the comparison of code in similar papers is the BLEU-4 metric, this metric makes use of tokens and calculates the score based on overlapping n-grams, the weights for the ratio of overlapping n-grams are uni formally distributed [16]. Another metric used is the METEOR score which makes use of unigram precision and recall and puts extra weight on the recall [17]. The last metric used is the inference time of the model which can be used to track any anomalies that could happen if the server is having issues. The inference time should not be very different for users but will increase slightly the more context is sent to the model.

D. Participants & Distribution

Participants of this study are acquired through our own professional networks and through online communities. Since the more data points the better conclusions can be drawn we decided to also advertise on Reddit and Discord to gather users. We also advertised on Instagram and Facebook using advertisements but these did not really increase the amount of users so this approach was stopped after a few days. The users can download the plugin for free on the JetBrains Marketplace and the VSC Marketplace. This simple process allows for a low barrier of entry which allows for more users. The participants will mostly consist of people interested in helping with the study which will be people with at least some programming experience and probably interested in machine learning and language processing.

VII. Results

The plugins for JetBrains and VSC have gotten over 450 downloads combined, with 200 of these users actually having used the plugin. Since some people found the plugin through the marketplaces of the IDE's and not through our advertising the amount of users that have used the plugin for Python and is 86. However since half of the users were assigned to another model for another study half of these users generated data for this study. There were 13 users that filled in the survey this is due not all users reaching the required threshold to have used the plugin enough to be able to accurately judge the performance of the model. The data collection period for this study ran from June 7 2022 till June 18 2022. The data collection will keep going for a bit longer for future studies. The rest of this chapter will show the results for the research questions and compare them using different categories to give clear insight in the usage of the plugin and capabilities of the model.

A. Acceptance Rate

The goal of the first research question is to evaluate the acceptance rate and accuracy of the InCoder model. Table I shows all the metrics for the all the data points combined, if the shown suggestion was explicitly chosen by the user, if the user decided to not click it and manually complete the statement, when the manual keybind is used, and the results on automatic trigger points suggestions. Using the Exact Match metric we can conclude that 21,95% the model had a perfect prediction and the line the user ended up with was the same as the model suggested. The table shows that in 16% of all cases the user chose the prediction of the model which on average had an edit similarity of 83.03% which shows that if the prediction was good users will in most cases use it. In some cases the tokens needed to finish the statement are so simple or short that the developer still prefers to type them instead of selecting the completion. The keybind performance is lower than automatic trigger points and also consists of less data points. The average inference time during the data collection was 230ms.

Table II shows the different performances of the model when a trigger point occurs in the code and an automatic suggestion is done by the plugin. The edit similarity metric ranges from 39.21% all the way up to 68.91%. Different trigger points tell the model more accurately what to do to complete the statement. The "in" keyword for example has high evaluation scores because the what comes after this keyword is limited to check if a value is present in a list and other classes or to loop over all elements like a for loop. This limits the options for the models and increases accuracy. The '=' and '.' are by far the most used trigger points accounting for almost half the data from all automatic suggestions.

B. Usefulness

Looking at Table I again shows that the average BLEU-4 score of the suggestions is 36.05 which is not that high when also looking at the edit similarity of 52.73%, Exact Match of 21.95%, and F1 Score of 42.8. The BLEU score still indicates that the model performs well and is not suggesting random code. Now taking a look at the survey to see if users find the performance of the model useful or annoying. One user rated the the perceived accuracy of the model as 5 (very good), nine users rated it as 4 (good) and the other three users rated it as 3 (neutral). Looking at the time saved we have a very similar result, one user answered they saved a lot of time by using the plugin, one user answered they saved no time but also no extra time was needed, and all the other answers being little time was saved. Half of the responses said the completions were a bit better than the default completions and the other half answered that they were a lot better.

C. Survey Results

All users that filled in the survey answered they would like to keep using the plugin. All the users made use of the automatic suggestion feature and some also used the keybind in conjunction with automatic suggestions. For use cases most users reported that the current way of showing suggestions the same way and appending them to the default suggestion window was the best way. Some however would also like inline completions, showing the suggestion greyed out where it would be inserted and either continue typing or pressing tab to use that suggestion, like GitHub Copilot does. There were quite some suggestions on how to improve the usefulness of the plugin and model. Some code statements can span multiple lines which the plugin would simply cut off due to the functionality being aimed on statement completion. The InCoder model does support this but due to the aim of the study being statement completion this functionality was not added. Other feedback was a bit more generic and consisted of reducing inference time, which sometimes could get a bit high due to the server also being used for other things, and improving suggestions made by the model.

By taking a look at Table III we see the performance of the model based on the length of tokens that were in the prediction. According to the BLEU metric there is a trend of the scores decreasing the more tokens are present in the prediction but there are also some outliers. 63% of the

TABLE I
EVALUATION OF INCODER WITH CHOSEN & WRITTEN AND AUTOMATIC & MANUAL CATEGORIES

		Total	Explicit Select	Not Selected	Keybind	Trigger Point
	Occurrences	4164	663	3501	468	3696
	Exact Match	21.95	62.14	14.34	14.32	22.92
	Edit Similarity	52.73	83.03	47.00	30.78	55.51
	BLEU-4	36.05	65.76	30.43	21.04	37.96
	METEOR	42.33	72.63	36.59	22.90	44.79
ROUGE-L	Precision	45.03	82.66	37.90	17.73	48.48
ROUGE-L	Recall	44.98	80.69	38.22	18.86	48.29
ROUGE-L	F1 Score	42.87	80.14	35.81	17.86	46.04

TABLE II
EVALUATION OF INCODER WITH TOP 10 OCCURRING TRIGGER POINTS

		=	.	(	,	if	return	in	+	[	-
	Occurrences	889	710	468	218	172	111	104	84	79	62
	Exact Match	22.16	26.76	21.37	32.11	24.42	45.05	47.12	15.48	5.06	3.23
	Edit Similarity	52.73	59.88	51.85	60.52	61.98	63.58	68.91	54.97	39.21	45.70
	BLEU-4	36.27	41.72	39.64	39.92	42.28	42.95	48.75	40.89	28.36	27.64
	METEOR	42.16	49.97	47.95	42.98	51.29	46.83	59.74	48.80	35.70	36.06
ROUGE-L	Precision	46.58	53.71	42.01	53.31	61.03	60.11	60.60	58.74	22.63	29.76
ROUGE-L	Recall	45.87	51.92	42.98	52.68	58.72	68.26	59.94	62.28	25.68	33.52
ROUGE-L	F1 Score	44.31	50.08	40.46	50.75	55.91	60.93	58.44	56.01	22.78	30.17

TABLE III
EVALUATION OF INCODER PREDICTIONS OF DIFFERENT TOKEN LENGTH

		1	2	3	4	5	6	7	8	9	10	>10
	Occurrences	638	614	502	590	369	285	285	201	107	107	466
	Exact Match	29.62	27.52	34.46	19.66	18.70	17.89	13.33	14.43	13.08	24.30	8.58
	Edit Similarity	45.22	56.18	62.83	55.82	55.34	53.72	50.92	49.35	48.29	56.29	43.81
	BLEU-4	18.97	37.28	45.20	44.41	42.79	39.13	40.12	35.14	34.22	41.98	27.14
	METEOR	17.72	42.73	52.45	46.62	48.70	45.98	50.23	46.19	45.59	52.84	42.21
ROUGE-L	Precision	45.43	43.56	56.45	47.42	48.18	48.00	40.42	38.74	38.92	46.03	33.45
ROUGE-L	Recall	39.97	37.81	52.63	46.14	47.09	45.83	45.27	41.67	46.16	56.19	47.79
ROUGE-L	F1 Score	41.22	38.87	52.89	45.44	45.47	44.76	40.85	38.27	39.80	48.02	35.89

predictions were 4 tokens or less and 87% were 10 tokens or less to finish the line.

VIII. DISCUSSION

The results for the most part are in line with the expectations are there are not huge anomalies that are immediately obvious. In the Methodology it was claimed the inference time was around 80ms but the average inference time was almost 3 times as high. This is a significant slowdown and can influence the result due to users already skipping the completion. Unfortunately this is due the server also being used for other projects when the data collection was running. The data is still valid and does not contain errors if this happened but a data point that could have been selected by the user could be classified as not selected due to it not appearing in time.

Due to the limited amount of time the data collection ran the amount of data is less then hoped for but still enough to accurately calculate most metrics, some categories that do not have a high enough amount of data points are the trigger points below the top 10 and predictions with a token length higher than 10. The group of token length 10 also has a significantly higher score then expected which is also due to the limited number of data points for that category.

Some users contributed a significant amount of data points more than other users. All these data points are treated equally which can allow some users to skew the scores. However enough users contributed enough data so this effect is minimal in this study. Another aspect with regards to the users is that some code faster than others. The evaluation was done by comparing the suggestion of the model and how the user actually programmed that line. This line by the user is collected 30 seconds after the suggestion is shown to the user. In some edge cases this might not be enough time but looking at our data these 30 seconds were enough for almost all data points.

Since we do not choose who uses the plugin there is a slight

risk of people participating in the study that have malicious intentions but these are the very minority if they even occur and the server handled this. There were no abnormal data points from users or any noticeable intent to skew or dilute the data.

The metrics used for this study are natural language processing metrics. Since code completion is a sub category of this and similar studies also used these metrics we also decided to use these. There does exist a variant of BLEU for code evaluation called CodeBLEU but this metric does not work for non compiling code due to it comparing the different abstract syntax trees of the code snippets. To tokenize the code, which is what these metrics use to calculate the scores, we made a custom tokenizer. It behaves exactly the same as the built in Python tokenizer but does not error on non compiling code. If another tokenizer was used the scores for the metrics would have changed.

The context given to the model is quite limited and not optimised. Since both the left and right context are limited to 1000 tokens if one context has a short length the other context could have been longer since there is space. The context also does not include code from imports so the model is unable to always correctly predict what methods can be used on an object. This issue is also apparent when trying to read data from a JSON file or a map and the context not including the keys that can be used.

Due to some users testing out the functionality of the plugin and model by using a new file and pressing the keybind to check it the scores for the keybind category are a bit lower. This is due the limited context in this scenario and the limited amount of data points. If there were more data points the impact of users testing the plugin out would be limited but this is not the case and has some influence on the results.

IX. Conclusions & Future Work

The overall results of the performance of the model are quite promising. With an Exact Match of 21.95%, an edit similarity of 52.73%, and a BLEU-4 score of 36.05 show that the suggestions from InCoder are quite accurate and contain a lot of correct tokens. The users also indicated that the functionality is desired and helps them save time due to useful completions or part of the completion. If the completion is good users will select it from the list unless it is a small simple completion that is easier to type then select. The automatic suggestions on trigger points are the most desired functionality for the users, this also increases the accuracy of the model since the model has more context to complete that line. Some users would like the keybind as well but removing it would not harm them.

For future work an important aspect to check is how the performance of the model changes if more context, such as filename and other files, or smarter context is sent. The smarter context could be filtering out some parts that do not seem useful for the model. Some user suggestions also seem promising such as evaluating code block completion and more suggestions to choose from. More data will also allow to draw better conclusions on less used trigger points to adjust the list of trigger points and increase performance via this approach.

X. Responsible Research

This study adheres to the GDPR regulations, every user has a random id which does not correspond to anything and can not be traced to an individual. This id is solely used to know which data points belong to which user so that we can track the amount of completions a user is shown to send the survey. Taking part in this study was voluntarily and users were clearly informed which minimal part of data collection is stored and how it is used. Since all the code used for this study is open source, both the plugins and the server, users can check that the data collection and storage is as how it was described. The data is stored on TU Delft servers and not shared with any third parties.

Since the study involved users and was affiliated with the TU Delft we had to request permission from the Human Resource Ethics Committee. This request contained the details of the data collection process, the targeted users, and risks involved. The user space was limited to the EU and people over 18 years old to minimise the risks. To minimise user bias the study was advertised as much as possible to get a diverse set of users.

XI. Acknowledgements

I would like to start of by thanking Maliheh Izadi for guiding me through the process of this study and providing valuable feedback. I would also like to thank Georgios Gousios for providing a powerful server where the models could be hosted and advising me. Lastly I want to thank my fellow students. Frank van der Heijden for working on the JetBrains plugin and server. Jorit de Weerdt for making the VSC plugin and helping wherever he could. Tim van Dam and Mika Turk for providing feedback and other helpful tips.

References

[1] D. Fried *et al.*, *Incoder: A generative model for code infilling and synthesis*, 2022. arXiv: 2204.05999 [cs.SE].

[2] V. Bibaev *et al.*, *All you need is logs: Improving code completion by learning from anonymous ide usage logs*, 2022. DOI: 10.48550/ARXIV.2205.10692. [Online]. Available: https://arxiv.org/abs/2205.10692.

[3] Y. Wan, W. Zhao, H. Zhang, Y. Sui, G. Xu, and H. Jin, *What do they capture? – a structural analysis of pre-trained language models for source code*, 2022. DOI: 10.48550/ARXIV.2202.06840. [Online]. Available: https://arxiv.org/abs/2202.06840.

[4] W. Zhou, S. Kim, V. Murali, and G. A. Aye, "Improving code autocompletion with transfer learning," *ArXiv*, vol. abs/2105.05991, 2021. [Online]. Available: https://arxiv.org/pdf/2105.05991.pdf.

[5] G. Aye, S. Kim, and H. Li, "Learning autocompletion from real-world datasets," Nov. 2020. [Online]. Available: https://arxiv.org/pdf/2011.04542.pdf.

[6] M. A. Hadi *et al.*, "On the effectiveness of pre-trained models for api learning," *arXiv preprint arXiv:2204.03498*, 2022.

[7] Y. Wang and H. Li, "Code completion by modeling flattened abstract syntax trees as graphs," Feb. 2021. [Online]. Available: https://arxiv.org/pdf/2103.09499.pdf.

[8] T. Nguyen and T. Nguyen, "Persona: A personalized model for code recommendation," *PLOS ONE*, vol. 16, e0259834, Nov. 2021. DOI: 10.1371/journal.pone.0259834.

[9] X. Wang *et al.*, *Compilable neural code generation with compiler feedback*, 2022. DOI: 10.48550/ARXIV.2203.05132. [Online]. Available: https://arxiv.org/abs/2203.05132.

[10] F. F. Xu, B. Vasilescu, and G. Neubig, *In-ide code generation from natural language: Promise and challenges*, 2021. DOI: 10.48550/ARXIV.2101.11149. [Online]. Available: https://arxiv.org/abs/2101.11149.

[11] B. Chen and C. Cherry, "A systematic comparison of smoothing techniques for sentence-level BLEU," in *Proceedings of the Ninth Workshop on Statistical Machine Translation*, Baltimore, Maryland, USA: Association for Computational Linguistics, Jun. 2014, pp. 362–367. DOI: 10.3115/v1/W14-3346. [Online]. Available: https://aclanthology.org/W14-3346.

[12] E. Shi *et al.*, "On the evaluation of neural code summarization," in *International Conference on Software Engineering (ICSE'22)*, Feb. 2022. [Online]. Available: https://www.microsoft.com/en-us/research/publication/on-the-evaluation-of-neural-code-summarization/.

[13] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, *Intellicode compose: Code generation using transformer*, 2020. DOI: 10.48550/ARXIV.2005.08025. [Online]. Available: https://arxiv.org/abs/2005.08025.

[14] M. Izadi, R. Gismondi, and G. Gousios, "Codefill: Multi-token code completion by jointly learning from structure and naming sequences," Feb. 2022.

[15] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013.

[16] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. [Online]. Available: https://aclanthology.org/P02-1040.

[17] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72. [Online]. Available: https://aclanthology.org/W05-0909.