1 FEATURES USED IN COPILOT'S FILTER

Based on [4], we investigated the `v1.57.7193` (June 2022) version of Copilot and list the features used in their logistic classifier in Table 1. The weight is the coefficient in the logistic regression model, and the scaling indicates the transformation applied before multiplying with the weight. The language and character maps (last three features) are a one-hot encoding of a fixed set of languages and characters, given in Figure 1 and Figure 2 respectively.

Before a prompt reaches the filter, two hard-coded rules prevent (1) prompts containing fewer than 10 characters, and (2) prompts where the cursor is in the middle of a line, by checking whether there is whitespace after the cursor; except if there is a closing character on that line, such as a closing bracket, quote, or semicolon.

Table 1: Features Used in Copilot's Filter.

Feature	Weight	Input Scaling
Previous filter label	0.997	none
Whitespace after cursor	0.700	none
Time since last label	−0.174	log
Length of last prefix line	−0.230	log
Above, without whitespace	0.134	log
Document length	−0.007	log
Cursor offset	0.005	log
Offset as percentage	0.419	none
Document language (map)	[−0.654, 0.358]	none
Last prefix char (map)	[−1.56, 1.15]	none
Above, without whitespace	[−1.12, 0.85]	none

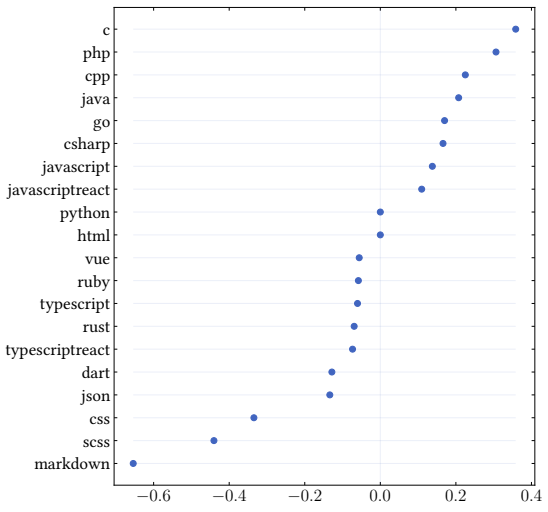

Figure 1: Copilot's Language Map. Higher-scoring languages are more likely to get a completion.

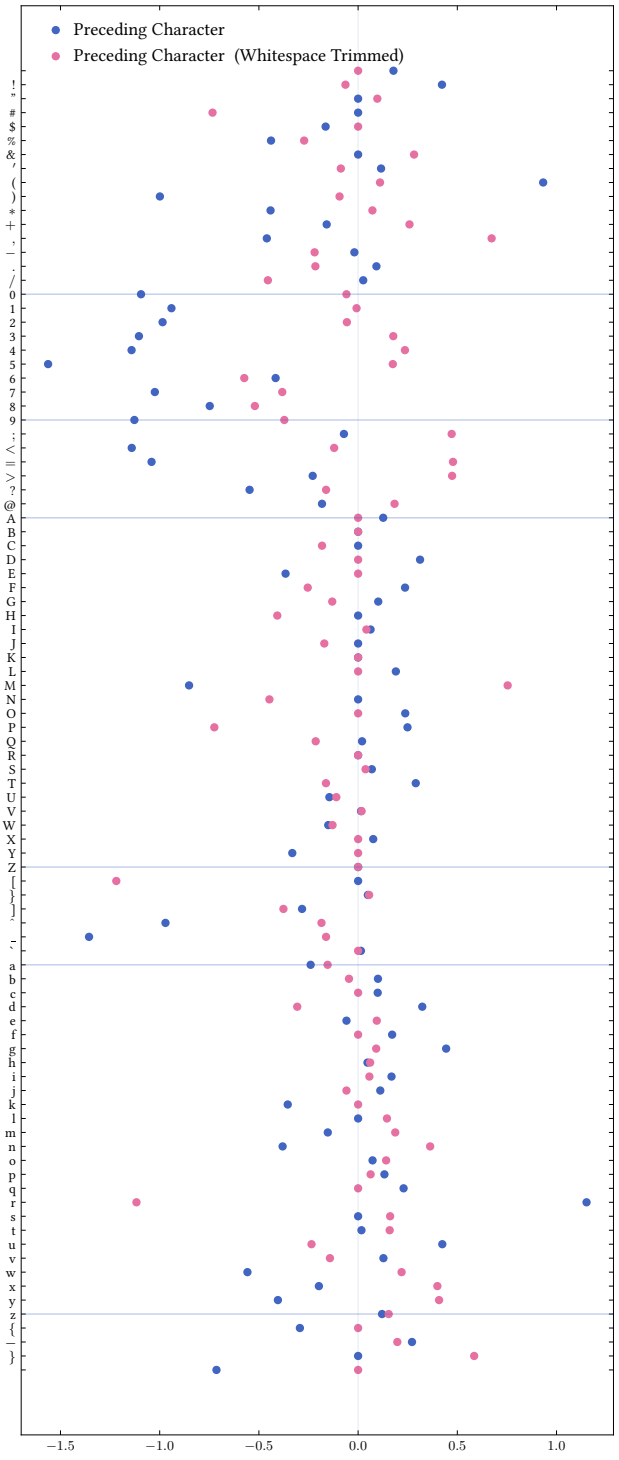

Figure 2: Copilot's Prefix Character Map. Higher-scoring characters (directly before the cursor) are more likely to get a completion.

AIWARE '24, July 15–16, 2024, Porto de Galinhas, Brazil
2024. ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

The languages in Figure 1 show that more verbose languages like C and Java are biased to receive more completions, while design-oriented languages like CSS, SCSS, and TypeScript-React are less likely to receive completions. Notably, Markdown is least likely to receive completions, likely because the LLMs that generate completions are primarily trained on code; and, it may be especially annoying to developers to receive incorrect completions when they are in a natural-language flow.

The characters in Figure 2 reveal some specific usage patterns too. Many letters fall close to 0, indicating that they do not have much influence on the filtering; with the exception of 'r', maybe due to the common print-statement completions which tend to be easier to infer. Furthermore, if the developer is currently typing a number, this has a negative weight. Presumably, programmers don't want completions while they are writing some uninferrable number. Lastly, perhaps surprisingly, the full-stop '.' has near-zero weight, while a comma ',' followed by whitespace has positive weight, implying developers may rely on Copilot to complete list/object structures.

2 COMPARISON OF TOKENISATION STRATEGIES

To investigate the effectiveness of our joint prefix-and-suffix tokenisation strategy, we compare it to the two alternatives: prefix-only and suffix-only tokenisation. To this end, we train RoBERTa models from the `huggingface/CodeBERTa-small-v1` checkpoint [1]. We use the same hyperparameters for all models: a learning rate of 2×10^{-5}, a batch size of 8, and train for 3 epochs.

Table 2: Comparison of tokenisation Strategies.

Tokenisation	Manual	Auto/acc.	Auto/rej.	**Average**
Suffix-only	89.3	**93.0**	24.1	68.8
Prefix-only	97.6	61.0	**74.7**	77.8
Joint	**98.6**	73.5	71.7	**81.3**

3 THE EFFECT OF DATASET DISTRIBUTION ON MODEL PERFORMANCE

Our code completion dataset distribution does not equally represent the invocation types we want to classify, yet they should be weighed roughly equally. To remedy this, we experiment with several data distributions (Figure 3) for training our models. As shown in Table 3 for CodeBERTa, and Table 4 for the Logistic Regression model, find that the biased distribution, giving roughly equal weight to all sub-classes *and* undersampling to better represent manual, accepted invocations, yields the best macro average performance across the sub-classes. We observe a similar pattern for all models detailed in Section 6. Thus, we use the *biased* distribution as our training/validation dataset.

For CodeBERTa in Table 3, the superior performance on the biased balancing may also be due to our training hyperparameters. We fine-tune for 8 epochs (as opposed to 6 in the rest of this paper), and our models are prone to overfitting due to the small training set size. Future work could perhaps study a data curriculum, going from biased to unbalanced (real-world distribution) to help the model learn classes quickly, and then generalise to data more diverse in textual content.

[1] https://huggingface.co/huggingface/CodeBERTa-small-v1

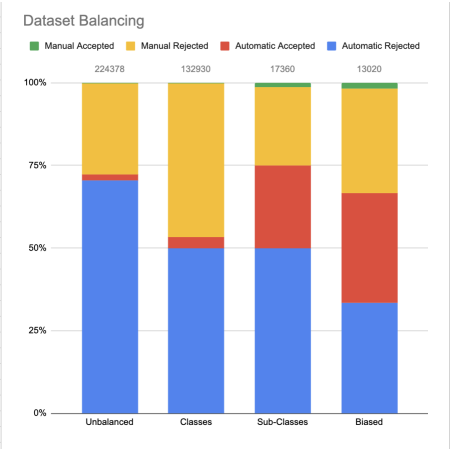

Figure 3: Dataset Distributions

Table 3: CodeBERTa Models under Different Training Data Distributions.

	Manual	Auto/acc.	Auto/rej.	**Average**
Unbalanced	98.9	19.4	98.1	72.1
Classes	99.2	32.2	95.7	75.7
Sub-Classes	98.2	71.6	74.4	81.3
Biased	98.6	78.9	67.3	**81.6**

Table 4: Logistic Regression Models (with full telemetry), under Different Training Data Distributions.

	Manual	Auto/acc.	Auto/rej.	**Average**
Unbalanced	97.4	0.0	99.8	65.8
Classes	97.8	0.0	99.7	65.8
Sub-Classes	97.8	35.7	88.4	74.0
Biased	98.5	66.1	65.0	**76.5**

4 JONBERTA-HEAD ARCHITECTURE EXPERIMENTS

We list the 3 JonBERTa-head variants below, with the 2 options for whether to reinitialise the module or not. We hypothesise that the dense variant performs best because dense layers allow the model to learn low-rank embeddings, and speed up training at this scale.

5 JONBERTA-ATTN ARCHITECTURE EXPERIMENTS

It may seem natural to consider telemetry data as simply another modality that can be fed to the transformer akin to multimodal

Table 5: JonBERTa-head Variants. R denotes re-initialised modules.

	Manual	Auto/acc	Auto/rej	**Avg.**
dense-R	98.6 ± 0.4	78.0 ± 6.2	71.4 ± 5.1	**82.7**
proj-R	98.6 ± 0.6	76.1 ± 11.0	72.4 ± 3.6	82.3
dense	98.6 ± 0.5	75.6 ± 6.8	72.6 ± 4.4	82.3
dense-proj-	98.6 ± 0.3	73.5 ± 6.7	74.1 ± 1.8	82.1
dense-proj-R	98.5 ± 0.5	72.2 ± 5.8	75.4 ± 1.9	82.0
proj	98.4 ± 0.5	72.9 ± 7.0	74.7 ± 5.0	82.0

large language models [3]. We choose not to explore this approach as it introduces too many new parameters for the model to learn in our limited-data setting. Furthermore, as far as we know, there are no studies investigating whether an existing natural-language transformer can be extended with additional modalities without reinitialising (all) its weights. We train a variety of layer configurations, from individual layers to triplets of layers; starting from two checkpoints: CodeBERTa-small-v1, provided by HuggingFace[2] **??**; and CodeBERTa as referenced in the paper, which is fine-tuned on our collected code completion dataset **??**.

Table 6: Performance of JonBERTa-attn Variants initialised from RoBERTa-small-v1.

Layers	Manual	Auto/acc	Auto/rej	**Avg.**
1, 4, 5	98.4 ± 0.4	75.9 ± 6.4	69.9 ± 5.8	**81.4**
2, 4, 5	98.4 ± 0.5	77.1 ± 6.9	68.5 ± 6.4	81.3
0, 4, 5	98.2 ± 0.7	76.4 ± 8.0	69.1 ± 5.0	81.3
0, 1, 4	98.3 ± 0.4	77.3 ± 5.8	68.1 ± 2.7	81.2
0, 1, 5	98.3 ± 0.4	77.3 ± 5.0	68.1 ± 2.8	81.2
0, 2, 4	98.3 ± 0.7	77.5 ± 7.1	67.9 ± 5.6	81.2
1, 2, 3	98.2 ± 0.5	77.6 ± 5.0	67.9 ± 2.1	81.2
0, 1, 3	98.4 ± 0.4	78.0 ± 6.1	67.2 ± 6.1	81.2
3, 4, 5	98.4 ± 0.5	76.3 ± 6.2	68.7 ± 3.3	81.1
0, 2, 5	98.3 ± 0.4	77.4 ± 5.8	67.6 ± 3.1	81.1
1, 2, 4	98.4 ± 0.5	77.3 ± 7.0	67.6 ± 5.0	81.1
0, 1, 2	98.5 ± 0.6	80.6 ± 5.2	64.1 ± 6.2	81.1
0, 3, 5	98.3 ± 0.5	76.0 ± 7.5	68.7 ± 6.4	81.0
2, 3, 4	98.3 ± 0.5	76.2 ± 6.7	68.4 ± 8.2	81.0
2	98.0 ± 0.6	77.1 ± 5.7	67.4 ± 4.0	80.8
2, 3, 5	98.3 ± 0.5	77.5 ± 7.2	66.6 ± 5.2	80.8
0, 3, 4	98.4 ± 0.5	76.6 ± 5.9	67.2 ± 4.3	80.7
0, 2, 3	98.3 ± 0.7	74.9 ± 10.6	68.8 ± 5.9	80.7
1, 2, 5	98.3 ± 0.3	76.8 ± 6.3	66.8 ± 6.1	80.6
1	98.2 ± 0.5	76.6 ± 7.7	66.9 ± 3.4	80.5
0	98.1 ± 0.7	76.5 ± 10.0	66.8 ± 5.8	80.5
1, 3, 5	98.2 ± 0.6	74.7 ± 10.2	68.3 ± 7.2	80.4
3	98.1 ± 0.8	77.5 ± 7.8	65.3 ± 5.7	80.3
1, 3, 4	98.3 ± 0.6	74.1 ± 7.6	68.5 ± 6.6	80.3
4	98.2 ± 0.4	77.7 ± 6.4	64.5 ± 5.3	80.1
5	98.2 ± 0.5	78.7 ± 8.2	63.4 ± 5.9	80.1

[2]https://huggingface.co/huggingface/CodeBERTa-small-v1

We observe better performance by initialising from our checkpoint that was fine tuned for 3 epochs on the code completion data we collected. We then fine-tune an additional 3 epochs with the additional telemetry features.

Table 7: JonBERTa-attn variants initialised from our fine-tuned CodeBERTa.

Layers	Manual	Auto/acc	Auto/rej	**Avg.**
4, 5	98.6 ± 0.5	76.6 ± 5.7	72.1 ± 4.2	82.4
2, 3, 4	98.5 ± 0.4	74.8 ± 5.9	73.8 ± 2.4	82.4
0, 1, 4	98.5 ± 0.4	75.2 ± 7.6	73.1 ± 4.1	82.3
0, 1, 3	98.5 ± 0.4	74.8 ± 4.6	73.4 ± 2.0	82.3
2, 5	98.5 ± 0.4	74.9 ± 4.6	73.2 ± 3.0	82.2
0, 5	98.6 ± 0.3	74.6 ± 7.5	73.3 ± 6.1	82.2
1, 2, 5	98.6 ± 0.3	75.5 ± 7.2	72.4 ± 2.7	82.2
2	98.6 ± 0.4	75.8 ± 5.9	71.9 ± 3.1	82.1
3, 4	98.6 ± 0.4	75.7 ± 5.3	72.0 ± 2.5	82.1
3, 4, 5	98.4 ± 0.4	74.4 ± 5.0	73.5 ± 2.3	82.1
3, 5	98.5 ± 0.3	74.2 ± 6.3	73.6 ± 2.7	82.1
0, 4, 5	98.5 ± 0.4	75.0 ± 4.9	72.8 ± 2.3	82.1
0, 2	98.7 ± 0.3	77.5 ± 4.6	70.1 ± 1.9	82.1
0, 3, 5	98.6 ± 0.4	75.4 ± 6.2	72.3 ± 2.7	82.1
2, 3, 5	98.4 ± 0.4	72.4 ± 6.2	75.4 ± 3.4	82.1
0, 2, 5	98.5 ± 0.4	74.1 ± 4.3	73.5 ± 1.5	82.1
0, 1	98.7 ± 0.4	75.4 ± 5.6	72.1 ± 2.1	82.0
0, 3	98.7 ± 0.3	76.7 ± 5.0	70.7 ± 3.0	82.0
0	98.6 ± 0.4	75.0 ± 6.8	72.5 ± 3.5	82.0
1, 2, 3	98.6 ± 0.4	74.4 ± 4.4	73.1 ± 1.3	82.0
1, 2, 4	98.5 ± 0.4	73.1 ± 7.8	74.4 ± 2.9	82.0
1, 5	98.6 ± 0.6	75.3 ± 9.1	72.1 ± 4.3	82.0
1, 4	98.6 ± 0.5	75.8 ± 7.1	71.6 ± 3.0	82.0
0, 1, 2	98.6 ± 0.5	74.3 ± 6.5	73.1 ± 6.0	82.0
0, 2, 4	98.5 ± 0.4	73.6 ± 4.6	73.7 ± 3.5	81.9
0, 2, 3	98.5 ± 0.5	74.3 ± 5.4	73.0 ± 5.1	81.9
0, 4	98.5 ± 0.4	75.0 ± 5.3	72.2 ± 1.4	81.9
0, 3, 4	98.6 ± 0.4	76.6 ± 7.9	70.5 ± 4.2	81.9
1, 2	98.6 ± 0.4	74.3 ± 8.4	72.6 ± 4.2	81.9
1, 3, 4	98.5 ± 0.4	74.6 ± 6.8	72.6 ± 3.4	81.9
0, 1, 5	98.6 ± 0.5	76.4 ± 10.1	70.6 ± 4.9	81.9
1, 4, 5	98.6 ± 0.4	73.7 ± 6.5	73.1 ± 2.2	81.8
1	98.5 ± 0.4	73.8 ± 7.3	73.0 ± 7.5	81.8
2, 3	98.5 ± 0.5	72.4 ± 7.6	74.4 ± 2.9	81.8
1, 3, 5	98.6 ± 0.5	73.0 ± 8.1	73.7 ± 4.8	81.7
2, 4, 5	98.5 ± 0.5	73.9 ± 5.5	72.8 ± 4.9	81.7
4	98.5 ± 0.5	75.5 ± 9.9	71.0 ± 4.4	81.7
2, 4	98.5 ± 0.4	73.7 ± 6.5	72.9 ± 6.7	81.7
3	98.5 ± 0.3	75.0 ± 8.6	71.4 ± 6.5	81.6
5	98.4 ± 0.5	74.6 ± 6.9	71.8 ± 5.6	81.6
1, 3	98.5 ± 0.6	73.8 ± 7.3	71.9 ± 5.8	81.4

6 COMPARISON OF MODELS

We compare by encoding n prefix and n suffix words/tokens/lines surrounding the cursor caret. It is not fully fair to compare every method with this granularity, as some methods may perform better

with e.g. 3 prefix and the entire suffix encoded, but we do so for consistency. If we had any remarkable results, this would not be in the appendix in the first place.

Autosklearn and AutoSklearn2 are ensembles consisting of 22 and 3 models respectively. We omit these models from the body of our study as we prefer simpler solutions over complex ones. We further want to highlight the potential of integrating pre-trained transformers' semantical understanding and contextual capabilities with additional modalities like telemetry feature data.

Table 8: Filter accuracy for Logistic Regression classification models with different code-context encodings, given per invocation sub-class. Macro denotes the macro average of the three sub-classes.

Logistic Regr. w/	Manual	Auto/acc.	Auto/rej.	**Macro**
One-Hot Tokens[3] (dim = 50821)				
3 Tokens	99.0	77.7	54.0	77.1
2 Tokens	99.1	77.5	53.0	76.7
1 Token	99.1	75.2	50.7	73.9
Tok2Vec[15] (dim = 100)				
3 tokens	97.7	74.0	58.8	76.0
2 tokens	98.3	75.4	56.2	76.3
1 token	98.5	76.3	52.7	75.5
Word2Vec (dim = 100)				
3 words	74.9	60.3	58.2	63.7
2 words	77.5	60.6	56.3	64.2
1 words	85.1	63.3	51.7	66.7
SetFit[4] (dim = 768)				
1 line	78.6	58.3	66.9	70.7
2 lines	67.4	57.6	62.1	61.4
3 lines	61.3	57.3	60.8	59.9
TF.IDF (dim $\in [15000, 18500]$)				
3 tokens	93.0	84.5	41.6	72.9
2 tokens	95.0	84.0	41.4	72.4
1 token	97.0	80.5	38.4	69.4
1 line	89.6	91.9	24.0	69.0
AutoSklearn [2]	98.9	75.9	64.6	79.4
AutoSklearn2 [1]	98.7	73.5	65.7	78.9

REFERENCES

[1] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *arXiv:2007.04074 [cs.LG]* (2020).

[2] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28 (2015)*. 2962–2970.

[3] Brandon McKinzie, Zhe Gan, Jean-Philippe Fauconnier, Sam Dodge, Bowen Zhang, Philipp Dufter, Dhruti Shah, Xianzhi Du, Futang Peng, Floris Weers, Anton Belyi, Haotian Zhang, Karanjeet Singh, Doug Kang, Ankur Jain, Hongyu Hè, Max Schwarzer, Tom Gunter, Xiang Kong, Aonan Zhang, Jianyu Wang, Chong Wang, Nan Du, Tao Lei, Sam Wiseman, Mark Lee, Zirui Wang, Ruoming Pang, Peter Grasch, Alexander Toshev, and Yinfei Yang. 2024. MM1: Methods, Analysis & Insights from Multimodal LLM Pre-training. http://arxiv.org/abs/2403.09611 arXiv:2403.09611 [cs].

[4] Parth Thakkar. 2023. Copilot Internals. https://thakkarparth007.github.io/copilot-explorer/posts/copilot-internals Publication Title: Copilot-Explorer.