Topology of Attention: Detecting Hallucinations in Code Generation Models

Anonymous ACL submission

Abstract

While the AI-code assistant tools become widespread, automatic assessment of the correctness of the generated code becomes a significant challenge. Code-generating LLMs are prone to hallucinations, which may lead to code that doesn't solve a required problem or even to code with severe security vulnerabilities. In this paper, we propose a new approach to assessment of code correctness. Our solution is based on topological data analysis (TDA) of attention maps of code LLMs. We carry out experiments with two benchmarks – HumanEval, MBPP and 5 code LLMs: StarCoder2-7B, CodeLlama-7B, DeepSeek-Coder-6.7B, Qwen2.5-Coder-7B, Magicoder-S-DS-6.7B. Experimental results show that the proposed method is better than several baselines. Moreover, the trained classifiers are transferable between coding benchmarks.

1 Introduction

Large Language Models (LLMs) are now widespread and have a great potential to transform natural language processing and artificial intelligence. As far as code generation is concerned, LLMs which are trained on large amounts of code, are capable to generate human-level code for a plethora of simple problems and are expected to revolutionize software engineering. At the same time, code generating LLMs are prone to hallucinations. These hallucination are of various types. Sometimes generated code has syntactic or logical errors, sometimes it is correct but do not solve a required problem. In some cases, the generated code might contain security issues or robustness issues, like a memory leak. While many definitions of hallucinations exist, in this paper we assume that code hallucination is a code which do not pass tests. For a wide adoption of code generating LLMs, there is a high need of automatic assessment of code quality. As for the current state of technologies, a significant time is spent to debugging and rewriting automatically generated code (Liang et al., 2024).

We hypothesize that code quality can be inferred before its execution from an internal state of LLM, in particular its attention maps. Previous studies have shown that attention maps of transformers are useful for artificial text detection (Kushnareva et al., 2021), acceptability judgments (Cherniavskii et al., 2022) and speech classification (Tulchinskii et al., 2022).

Attention maps of LLMs are shown to capture semantically meaningful information and might be a illustration to model's "thought process". The research community actively studies approaches to mitigate hallucinations of LLMs by extenral knowledge bases (Peng et al., 2023) or reduce them to some degree (Elaraby et al., 2023). It is a highly desirable to evaluate to code quality before its execution and a running of tests since the code might contain security vulnerabilities.

The study of hallucinations in LLMs is intrinsically tied to generalization in NLP models. Both challenges stem from how models learn, represent, and apply knowledge. Improving generalization—through robust training, diverse data, and better uncertainty handling—reduces hallucinations by ensuring models produce contextually appropriate, factually grounded outputs. Conversely, analyzing hallucinations provides insights into generalization failures, guiding the development of more reliable NLP systems. This symbiotic relationship underscores the importance of addressing both issues holistically in AI research.

Out contributions are the following:

- We propose a new approach to detection of hallucinations in LLM generated code based on analysing a topology of attention maps;
- We carry out computational experiments with CodeLlama, StarCoder2, DeepSeek-Coder and Qwen2.5-Coder and two benchmarks –

HumanEval and MBPP, and show that the proposed method outperforms baselines;

- We empirically show that proposed classifier of hallucinations is transferable between code benchmarks.

2 Related work

Code generation via Large Language Models (LLMs) is the topic of active research. The popular projects are: CodeLlama (Roziere et al., 2023), StarCoder2 (Lozhkov et al., 2024), DeepSeek-Coder (Guo et al., 2024), Qwen2.5-Coder (Hui et al., 2024), to name a few. Code LLMs differ by data were used for training, by their training and fine-tuning protocols, including RLHF, tokenizers, variants of attention mechanism, etc.

Several works studied attention maps in transformer-based LLMs. (Clark, 2019) studied BERT's attention patterns: attending to delimiter tokens, specific positional offsets, or broadly attending over the whole sentence, with heads in the same layer often exhibiting similar behaviors. (Clark, 2019) further showed that certain attention heads correspond well to linguistic notions of syntax and coreference. (Htut et al., 2019) found that for some universal dependency tree relation types, there exist heads that can recover the dependency type significantly better than baselines on parsed English text, suggesting that some self-attention heads act as a proxy for syntactic structure. (Michel et al., 2019) showed that for downstream tasks, a large proportion of attention heads can be removed at test time without significantly impacting performance, and that some layers can even be reduced to a single head.

The phenomenon of code hallucinations is studied and categorized several papers. (Tian et al., 2024) introduces a categorization of code hallucinations into four main types: mapping, naming, resource, and logic hallucinations, with each category further divided into different subcategories. (Tian et al., 2024) proposed a CodeHalu dataset and studied frequencies of different types of hallucinations in popular code LLMs. (Liu et al., 2024) categorized hallucinations into: intent conflicting, inconsistency, repetition, knowledge conflicting, dead code. (Liu et al., 2024) released a HaluCode benchmark with labeled code hallucinations. (Jiang et al., 2024) proposed Collu-Bench, the benchnark with localization of code hallucinations. (Jiang et al., 2024) found that code LLMs are less confident when hallucinating, as the hallucinated tokens have lower probability and hallucinated generation steps have higher entropy.

In the broader context of NLP, several works introduced methods to hallucination preventing and detection. (Peng et al., 2023) proposed to mitigate hallucination by an LLM-AUGMENTER, a system which makes the LLM generate responses grounded in external knowledge, e.g., stored in task-specific databases. (Zhang et al., 2024b) proposed Self-Eval, a self-evaluation component, to prompt an LLM to validate the factuality of its own generated responses solely based on its internal knowledge. (Feng et al., 2024) proposed two novel approaches for hallucination detection that are based on model collaboration, i.e., LLMs probing other LLMs for knowledge gaps, either cooperatively or competitively. (Zhang et al., 2024a) proposed to improve truthfulness of LLMs by editing their internal representation during inference in the "truthful" space. (Yehuda et al., 2024) introduced InterrogateLLM, a method which prompts the model multiple times to reconstruct the input query using the generated answer. Subsequently, InterrogateLLM quantifies the inconsistency level between the original query and the reconstructed queries.

3 Background

3.1 Transformer-based LLMs

All of the state-of-the art LLMs for code generation networks are based on different variants of the transformer architecture (Vaswani, 2017). A transformer architecture comprises L layers of multi-head self-attention blocks each of them having H heads. Each attention head takes $X \in \mathbb{R}^{n\times d}$ matrix as an input, and an output of attention head in X^{out}:

$$X^{out} = A(XW^v),$$

$$A = \mathrm{softmax}\left(\frac{(XW^Q)(XW^K)^T}{\sqrt{d}}\right),$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d\times d}$ are projection matrices and $A \in [0,1]^{n\times n}$ is an **attention map**. In self-attention block, the attention map shows how each token in the input sequence "interacts" to every other token in the same sequence. A token might attend more to other tokens that are contextually related. We interpret each element $a_{i,j}$ of

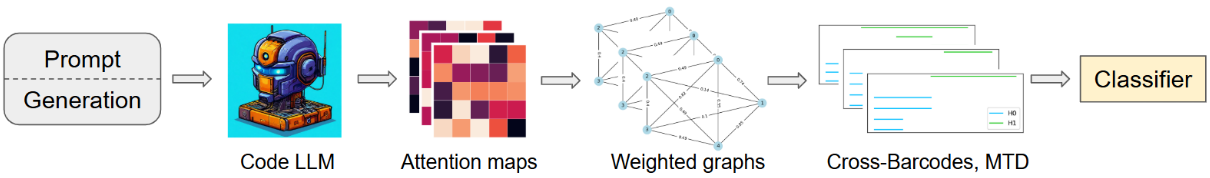


Figure 1: A pipeline of the proposed method for hallucination detection.

an attention map as an "interaction force" between tokens i and j.

3.2 Representing attention map by a weighted graph

While attention map is typically presented as a matrix, we treat it as a weighted graph. For n tokens in a sequence, we consider a fully-connected weighted graph with n vertices, where weights of edges are related to the "interaction force" between tokens (vertices). The natural idea is to leave only the most interacting tokens, that is, attending to each other higher than some threshold. However, the optimal threshold is not known in advance. Moreover, topology of such graph changes discontinuously with the change of a threshold (or weights). Topological Data Analysis (TDA) (Chazal and Michel, 2017) introduces a principled way to access topology of such graphs for all thresholds simultaneously.

3.3 Manifold Topology Divergence

MTD (Manifold Topology Divergence) (Barannikov et al., 2021) is a tool of TDA which can be used to evaluate the "dissimilarity" between two sets of vertices in a weighted graph $\mathcal{G} = (V, E, W)$ or, in other words, to which degree one set of vertices is covered by another set.

Let a set of vertices $V = P \sqcup G$, be split into disjoint sets P, G. We consider a nested sequence of graphs $\mathcal{G}_0 \subset \ldots \subset \mathcal{G}_i \subset \mathcal{G}_{i+1} \subset \ldots \subset \mathcal{G}$ in the following way. $\mathcal{G}_0$ has all the vertices P, G and all the edges connecting vertices from P. The sequence $\mathcal{G}_i$ is obtained by adding the rest of edges one by one in an ascending order by their weights, see Figure 2. During this process, graphs' topology naturally changes: connected components are merged, cycles appear and disappear, etc. This process is rigorously described by the persistence barcodes theory (Barannikov, 1994; Chazal and Michel, 2017). Each topological feature like connected component or cycle has "birth time" and "death time", by a corresponding edge weight. The multi-set of these birth-death pairs (intervals) altogether is called a Cross-Barcode$_k$, see Figure 3. Here k is an index of a *persistence homology*, each of them reflects a kind of topological feature: 0 - connected components, 1 - cycles, 2 - voids, etc. MTD$_k$ is an integral characteristic of a Cross-Barcode$_k$ and it is defined as a sum of birth-death intervals' lengths. The higher MTD$_k$ is, the bigger is a "dissimilarity" between sets of tokens. Note, that according to a definition, MTD$_k$ is not symmetric. Also, MTD$_k$, as a kind of persistence barcode, enjoys stability w.r.t. small perturbations of weights (Cohen-Steiner et al., 2005).

4 Methods

In the context or code generation, we naturally have two sets of tokens – a prompt and a generation. Intuitively, hallucination happens when code LLM *doesn't pay much attention* to the prompt. As was pointed in Section 3.2, attention matrices can be analyzed as weighted graphs. Specifically, for n tokens in a sequence, we consider a fully-connected weighted graph with n vertices, where weights of edges are obtained by a symmetrization of an attention map: $w_{i,j} = 1 - \max(a_{i,j}, a_{j,i})$, for $i \neq j$. Then, Cross-Barcode and MTD for a weighted "attention graph" can be calculated. To predict code hallucinations, we use the following set of features:

- $\mathrm{MTD}_0(P, G)/|P|$, $\mathrm{MTD}_0(G, P)/|G|$
- $\mathrm{MTD}_1(P, G)/|P|$, $\mathrm{MTD}_1(G, P)/|G|$
- $\sum_{i \in P} a_{i,i}/|P|$, $\sum_{i \in G} a_{i,i}/|G|$

Here all the features are normalized by a size of corresponding vertices set for better transferability. Additionally, sums of diagonal values of attention matrices which are not directly present in edge weights are included. These features are calculated for every layer and head of a code LLM. At the top of the proposed topological features, we applied XGBoost (Chen and Guestrin, 2016) for a classification. The high-level pipeline of the proposed method is shown in Figure 1.

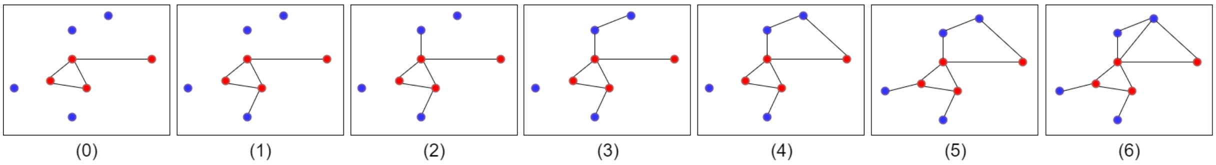


Figure 2: An example of MTD evaluation for a graph having two groups of vertices – red and blue. (0): initially, only edges connecting red vertices are present. (1)-(6): the rest of edges are added sequentially in an ascending order by their weights. While adding edges, connected components merge with each other. These moments are depicted by H_0 bars in Fig. 3. At moment (4) a cycle appears, at moment (6) this cycle disappears. These moments are depicted by the H_1 bar in Fig. 3.

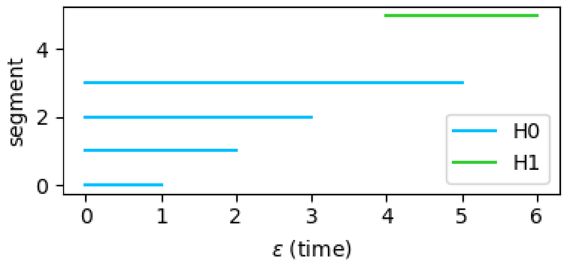


Figure 3: Cross-Barcode for a filtration from Fig. 2.

Model	Pass@1	#Pos.	#Neg.
HumanEval			
StarCoder2-7B	28.9	1186	2914
CodeLlama-7B	25.9	1064	3036
DeepSeek-Coder-6.7B	40.3	1653	2447
Qwen2.5-Coder-7B	47.8	1961	2139
Magicoder-S-DS-6.7B	65.5	2689	1411
MBPP			
StarCoder2-7B	42.8	1071	1429
CodeLlama-7B	35.2	879	1621
DeepSeek-Coder-6.7B	52.6	1315	1185
Qwen2.5-Coder-7B	52.1	1302	1198
Magicoder-S-DS-6.7B	61.3	1533	967

Table 1: Characteristics of generated data: Pass@1, number of correct (#Pos.) and incorrect (#Neg.) solutions for each of the selected code LLMs.

5 Experiments

5.1 Generation of datasets

To assess the efficacy of the proposed method for hallucination prediction, we carry out a set of computational experiments. We use the following popular code LLMs: StarCoder2-7B (Lozhkov et al., 2024), CodeLlama-7B (Roziere et al., 2023), DeepSeek-Coder-6.7B (Guo et al., 2024), Qwen2.5-Coder-7B (Hui et al., 2024), Magicoder-S-DS-6.7B (Wei et al., 2024). We adapted two public benchmarks for evaluation of code generation: HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021)[1]. In order to account for various possible code generations, for each of the coding problems several solutions were generated by each of the selected code LLMs: we obtained 25 generations per task for HumanEval and 5 generations per task for MBPP. To address the quality of the proposed approach in different LLM prompting regimes, we used 0-shot prompt for the HumanEval dataset and 1-shot prompt for the MBPP dataset. To enable diversity of generated solutions, a sampling with non-zero temperature of was done. Thus, we obtain 4100 samples for HumanEval and 2500 samples for MBPP for each code LLM. See Appendix A for further details. Table 1 presents a summary of generated code solutions. The correctness of code is evaluated via tests provided together with the coding benchmarks. Incorrect code is considered a "hallucination"; prediction of code's correctness is a binary classification problem. The pass@1 metric is slightly lower that reported in original papers, mostly because we have used sampling with non-zero temperature instead of greedy search. Before moving further, note that there is a strong negative dependency between prompt and generation lengths and code quality, see Figure 5, 9. The longer the prompt (i.e. task description) and generation (i.e. task solution) are, the lower is the probability of code's correctness. This dependency is more pronounced for HumanEval than MBPP, because MBPP employed more complicated 1-shot prompts. These attributes are natural baselines for hallucination's prediction.

5.2 Analyzing method's classification quality

Using the generated data, we estimated the classification quality of the proposed approach. We applied 5-fold stratified group cross-validation where different solutions of the same coding problem

[1]Licenses of pretrained models and benchmarks permit use for research purposes.

Method	ROC-AUC	F1-Score
	StarCoder2-7B	
Prompt Len.	54.5 ± 6.6	24.6 ± 11.2
Gen. Len.	57.7 ± 5.6	13.5 ± 4.8
Mean Log. Prob.	70.9 ± 1.3	32.4 ± 4.8
CodeT5-base ft.	70.1 ± 7.1	33.3 ± 10.1
Attn. Feat. (ours)	$\mathbf{82.9 \pm 2.7}$	$\mathbf{54.2 \pm 6.9}$
	CodeLlama-7B	
Prompt Len.	61.6 ± 4.4	25.7 ± 15.0
Gen. Len.	60.1 ± 5.3	10.6 ± 7.0
Mean Log. Prob.	64.1 ± 2.0	25.4 ± 6.2
CodeT5-base ft.	74.5 ± 6.3	43.6 ± 13.2
Attn. Feat. (ours)	$\mathbf{85.6 \pm 3.9}$	$\mathbf{56.4 \pm 7.2}$
	DeepSeek-Coder-6.7B	
Prompt Len.	56.2 ± 4.6	44.4 ± 4.3
Gen. Len.	57.9 ± 2.4	34.4 ± 4.9
Mean Log. Prob.	69.8 ± 2.5	51.1 ± 3.4
CodeT5-base ft.	69.1 ± 4.2	52.6 ± 6.5
Attn. Feat. (ours)	$\mathbf{85.6 \pm 2.8}$	$\mathbf{68.9 \pm 5.5}$
	Qwen2.5-Coder-7B	
Prompt Len.	54.3 ± 8.7	51.0 ± 5.7
Gen. Len.	57.6 ± 3.6	48.9 ± 5.1
Mean Log. Prob.	63.1 ± 2.4	55.6 ± 5.5
CodeT5-base ft.	65.9 ± 3.7	58.2 ± 4.5
Attn. Feat. (ours)	$\mathbf{81.7 \pm 2.8}$	$\mathbf{70.2 \pm 4.2}$
	Magicoder-S-DS-6.7B	
Prompt Len.	57.3 ± 5.4	70.4 ± 7.0
Gen. Len.	52.5 ± 2.1	76.3 ± 2.6
Mean Log. Prob.	71.0 ± 5.3	78.4 ± 2.9
CodeT5-base ft.	64.7 ± 2.7	77.5 ± 2.7
Attn. Feat. (ours)	$\mathbf{82.3 \pm 4.9}$	$\mathbf{80.7 \pm 3.6}$

Table 2: Code hallucination detection for HumanEval dataset.

Method	ROC-AUC	F1-Score
	StarCoder2-7B	
Prompt Len.	51.2 ± 2.3	40.0 ± 3.8
Gen. Len.	57.7 ± 0.9	45.4 ± 3.5
Mean Log. Prob.	62.0 ± 2.0	47.5 ± 3.4
CodeT5-base ft.	58.5 ± 3.5	43.3 ± 9.0
Attn. Feat. (ours)	$\mathbf{81.9 \pm 2.4}$	$\mathbf{68.4 \pm 5.3}$
	CodeLlama-7B	
Prompt. Len.	59.1 ± 4.2	35.4 ± 2.9
Gen. Len.	60.8 ± 2.5	24.2 ± 5.3
Mean Log. Prob.	61.0 ± 3.7	27.2 ± 1.5
CodeT5-base ft.	61.7 ± 3.0	19.1 ± 7.1
Attn. Feat. (ours)	$\mathbf{83.4 \pm 3.3}$	$\mathbf{64.0 \pm 4.4}$
	DeepSeek-Coder-6.7B	
Prompt Len.	52.5 ± 2.5	56.4 ± 3.6
Gen. Len.	54.6 ± 1.9	59.4 ± 1.3
Mean Log. Prob.	61.0 ± 1.9	62.3 ± 1.6
CodeT5-base ft.	55.7 ± 3.0	64.8 ± 2.7
Attn. Feat. (ours)	$\mathbf{82.6 \pm 1.9}$	$\mathbf{76.5 \pm 2.7}$
	Qwen2.5-Coder-7B	
Prompt Len.	51.8 ± 3.6	56.2 ± 4.4
Gen. Len.	55.6 ± 2.1	59.7 ± 4.8
Mean Log. Prob.	61.5 ± 1.3	60.4 ± 1.8
CodeT5-base ft.	56.0 ± 1.3	65.2 ± 2.0
Attn. Feat. (ours)	$\mathbf{82.2 \pm 2.2}$	$\mathbf{75.4 \pm 1.7}$
	Magicoder-S-DS-6.7B	
Prompt Len.	52.5 ± 2.5	56.4 ± 3.6
Gen. Len.	58.7 ± 1.1	60.7 ± 2.1
Mean Log. Prob.	60.6 ± 3.7	72.1 ± 1.4
CodeT5-base ft.	61.0 ± 3.7	$\mathbf{74.8 \pm 1.4}$
Attn. Feat. (ours)	$\mathbf{77.8 \pm 2.5}$	73.4 ± 3.4

Table 3: Code hallucination detection for MBPP dataset.

belonged to the same group. In this way, training and testing were performed always at non-overlapping coding problems (prompts). The reported results are the mean and standard deviation estimated over the 5 folds. As baselines for comparison, we used XGBoost classifier trained on simple features: tokenized prompt length, tokenized generation length, and mean log-probability of generated tokens (Chen et al., 2021). Also, we trained a linear classification head on top of a frozen CodeT5-base (Wang et al., 2021) encoder. Training details are provided in Appendix B. Tables 2, 3 present results. In the majority of cases, the proposed classifier based on features of attention maps performed significantly better than the baselines and demonstrated stable results for all models and datasets as measured by ROC-AUC score. Further analysis revealed that some features made a high contribution to the classification quality, see Figure 4.

5.3 Analyzing method's ranking quality

Next, we assess the usefulness of the proposed code hallucination classifier for ranking of code generations. For each problem, all generations were ranked via probability of correctness predicted by the classifier and one with the highest probability was selected. A baseline was random picking of a code generation. The usage of a classifier is always significantly better by a pass@1 score, see Table 4.

5.4 Method's transferability between benchmarks

We study further the transferability of the classifiers, based on topological features. In this setting, hallucination classifiers for a fixed code LLM are trained on data for one benchmark (HumanEval, MBPP) and evaluated on another, then repeated

Model	Random	Clf. Prob.
HumanEval		
StarCoder2-7B	28.6 ± 5.5	43.3 ± 9.0
CodeLlama-7B	26.0 ± 5.1	39.7 ± 7.2
DeepSeek-Coder-6.7B	39.1 ± 4.9	56.7 ± 7.4
Qwen2.5-Coder-7B	51.8 ± 8.0	64.0 ± 7.3
Magicoder-S-DS-6.7B	72.5 ± 10.0	74.3 ± 6.1
MBPP		
StarCoder2-7B	43.0 ± 3.6	49.6 ± 4.6
CodeLlama-7B	35.2 ± 3.3	43.6 ± 3.4
DeepSeek-Coder-6.7B	53.0 ± 2.5	61.4 ± 2.3
Qwen2.5-Coder-7B	52.6 ± 3.6	62.0 ± 2.4
Magicoder-S-DS-6.7B	61.4 ± 3.4	63.8 ± 2.0

Table 4: pass@1 for random choice vs argmax of classifier probability

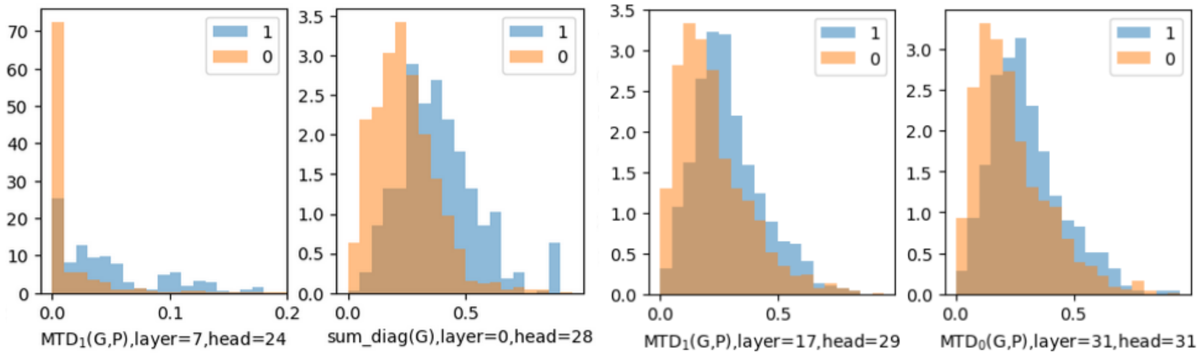
vise versa. Tables 5, 6 shows results: the proposed classifiers are transferable, albeit the performance is lower when training and testing is done on the same benchmark.

5.5 Ablation study

The proposed approach is based on the two types of attention features: the diagonal elements of attention maps corresponding to the prompt and generation and topological features computed for the corresponding "attention graph" (see Section 4 for details). In this Section, we provide an ablation study to estimate the contribution of each type of attention features. For this purpose, we trained the XGBoost classifier using only MTD features (i.e. without the diagonal elements of attention maps) or using only diagonal attention values (i.e. without MTD features) and compared its performance with the initial setup where both types of attention features were used. As demonstrated with Tables 7, 8, the DeepSeek-Coder-6.7B and Qwen2.5-Coder-7B achieved the best performance when both types of attention features were used for both HumanEval and MBPP datasets. In contrast, the best performance of StarCoder2-7B and Magicoder-S-DS-6.7B was achieved with different sets of attention features dependent on dataset and metric choices. In order to account for various information available via attention maps, we propose to use both types of features as the most universal choice. Nevertheless, we note that for some code LLM one certain type of attention features may result in better performance than combination of both types.

Model	ROC-AUC	F1-Score
StarCoder2-7B		
Prompt Len.	48.6	0.0
Gen. Len.	56.0	14.6
Mean Log. Prob.	63.7	36.2
CodeT5-base ft.	53.7	0.0
Attn. Features	67.5	0.14
CodeLlama-7B		
Prompt Len.	51.7	0.0
Gen. Len.	61.5	4.2
Mean Log. Prob.	57.7	15.2
CodeT5-base ft.	54.9	0.0
Attn. Features	69.5	0.2
DeepSeek-Coder-6.7B		
Prompt Len.	48.0	15.6
Gen. Len.	55.3	41.4
Mean Log. Prob.	62.5	56.3
CodeT5-base ft.	53.4	0.0
Attn. Features	67.7	70.4
Qwen2.5-Coder-7B		
Prompt Len.	49.9	34.1
Gen. Len.	51.6	46.1
Mean Log. Prob.	60.3	60.4
CodeT5-base ft.	49.1	52.3
Attn. Features	70.6	63.3
Magicoder-S-DS-6.7B		
Prompt Len.	48.1	56.3
Gen. Len.	54.8	75.2
Mean Log. Prob.	63.7	74.9
CodeT5-base ft.	49.3	76.0
Attn. Features	73.5	78.4

Table 5: Transferability of code hallucination detectors. Each classifier was trained on HumanEval (HE) dataset and tested on MBPP dataset.

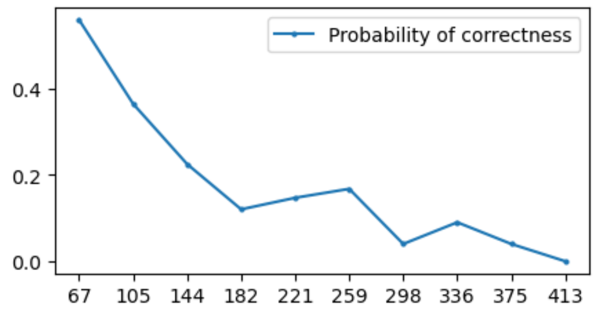


(a) HumanEval (b) MBPP

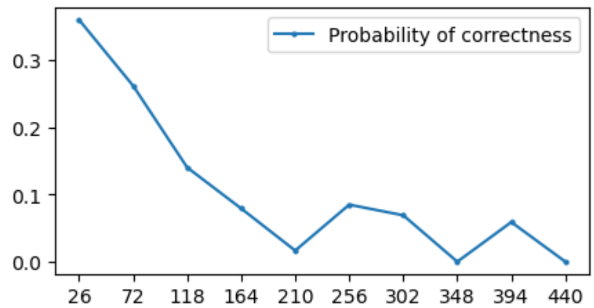
Figure 4: Distribution of classes (0-code is not correct, hallucination, 1-code is correct) vs. features from attention maps. Some of the most discriminative features are presented. Features are normalized with MinMaxScaler. CodeLlama.

Model	ROC-AUC	F1-Score
StarCoder2-7B		
Prompt Len.	52.1	45.0
Gen. Len.	52.4	38.3
Mean Log. Prob.	71.8	45.4
CodeT5-base ft.	59.1	0.0
Attn. Features	67.2	25.5
CodeLlama-7B		
Prompt Len.	53.4	42.9
Gen. Len.	50.0	41.0
Mean Log. Prob.	65.0	34.9
CodeT5-base ft.	62.4	0.0
Attn. Features	80.3	34.1
DeepSeek-Coder-6.7B		
Prompt Len.	52.2	58.2
Gen. Len.	54.0	51.3
Mean Log. Prob.	69.1	58.3
CodeT5-base ft.	55.9	57.4
Attn. Features	72.4	20.4
Qwen2.5-Coder-7B		
Prompt Len.	51.1	64.1
Gen. Len.	54.5	54.3
Mean Log. Prob.	64.7	60.8
CodeT5-base ft.	51.6	65.6
Attn. Features	64.2	54.3
Magicoder-S-DS-6.7B		
Prompt Len.	54.5	79.5
Gen. Len.	56.1	74.6
Mean Log. Prob.	69.8	76.5
CodeT5-base ft.	45.9	79.2
Attn. Features	56.9	37.6

Table 6: Transferability of code hallucination detectors. Each classifier was trained on MBPP dataset and tested on HumanEval (HE) dataset.



(a) Prompt length, tokens. HumanEval



(b) Generation length, tokens. HumanEval

Figure 5: The individual conditional expectations for prompt and generation lengths, CodeLlama.

Method	ROC-AUC	F1-Score
StarCoder2-7B		
Attn. Feat. (ours)	82.9 ± 2.7	54.2 ± 6.9
- w/o Diag. Feat.	82.2 ± 4.5	$\mathbf{56.1 \pm 9.7}$
- w/o MTD Feat.	$\mathbf{83.8 \pm 2.7}$	52.5 ± 8.4
CodeLlama-7B		
Attn. Feat. (ours)	$\mathbf{85.6 \pm 3.9}$	56.4 ± 7.2
- w/o Diag. Feat.	83.5 ± 4.8	50.0 ± 6.5
- w/o MTD Feat.	85.5 ± 4.4	$\mathbf{58.3 \pm 10.1}$
DeepSeek-Coder-6.7B		
Attn. Feat. (ours)	$\mathbf{85.6 \pm 2.8}$	$\mathbf{68.9 \pm 5.5}$
- w/o Diag. Feat.	85.1 ± 2.2	67.0 ± 5.9
- w/o MTD Feat.	84.4 ± 2.2	67.1 ± 3.8
Qwen2.5-Coder-7B		
Attn. Feat. (ours)	$\mathbf{81.7 \pm 2.8}$	$\mathbf{70.2 \pm 4.2}$
- w/o Diag. Feat.	80.6 ± 2.3	68.9 ± 3.9
- w/o MTD Feat.	78.9 ± 1.9	66.4 ± 1.4
Magicoder-S-DS-6.7B		
Attn. Feat. (ours)	$\mathbf{82.3 \pm 4.9}$	80.7 ± 3.6
- w/o Diag. Feat.	79.8 ± 2.7	81.1 ± 1.8
- w/o MTD Feat.	82.1 ± 3.4	$\mathbf{81.6 \pm 2.6}$

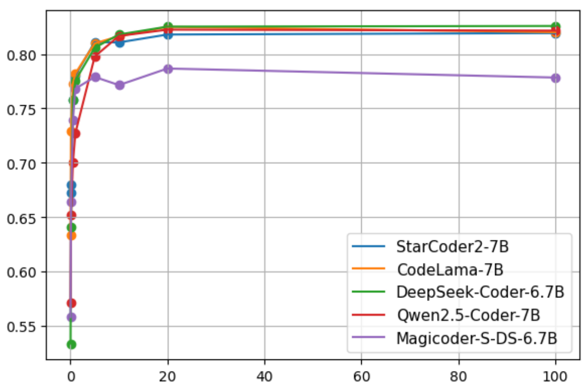
Table 7: HumanEval features ablation

Method	ROC-AUC	F1-Score
StarCoder2-7B		
Attn. Feat. (ours)	$\mathbf{81.9 \pm 2.4}$	$\mathbf{68.4 \pm 5.3}$
- w/o Diag. Feat.	80.5 ± 2.8	66.3 ± 5.3
- w/o MTD Feat.	81.1 ± 2.6	67.7 ± 5.0
CodeLlama-7B		
Attn. Feat. (ours)	83.4 ± 2.2	$\mathbf{64.0 \pm 4.4}$
- w/o Diag. Feat.	81.5 ± 2.6	60.2 ± 4.2
- w/o MTD Feat.	$\mathbf{83.5 \pm 1.8}$	63.9 ± 4.3
DeepSeek-Coder-6.7B		
Attn. Feat. (ours)	$\mathbf{82.6 \pm 1.9}$	$\mathbf{76.5 \pm 2.7}$
- w/o Diag. Feat.	81.3 ± 2.6	74.9 ± 3.2
- w/o MTD Feat.	82.2 ± 1.7	75.9 ± 1.7
Qwen2.5-Coder-7B		
Attn. Feat. (ours)	$\mathbf{82.2 \pm 2.2}$	$\mathbf{75.4 \pm 1.7}$
- w/o Diag. Feat.	80.8 ± 2.1	$\mathbf{75.4 \pm 0.4}$
- w/o MTD Feat.	76.9 ± 2.2	71.6 ± 1.7
Magicoder-S-DS-6.7B		
Attn. Feat. (ours)	77.8 ± 2.5	$\mathbf{73.4 \pm 3.4}$
- w/o Diag. Feat.	77.8 ± 3.2	72.4 ± 2.9
- w/o MTD Feat.	$\mathbf{78.4 \pm 2.6}$	73.2 ± 2.1

Table 8: MBPP features ablation

5.6 Analyzing method's pruning ability

In its base setup, the proposed approach requires computation of attention features from attention maps for all layers and heads. However, we explored that the trained XGBoost classifier experienced a natural sparsity with only about 25% of meaningful features as measured by classifiers' feature importance. To explore further the pruning ability of our approach, we followed the two-stage pipeline. First, for a given sparsity level, we selected the most important features as measured by feature importance of the classifier trained on all attention features simultaneously. Second, we trained a new XGBoost classifier on the selected set of attention features. As indicated by Figure 6, the proposed feature selection procedure could retain only 5% of all attention features without significant loss of classification quality highlighting that only a limited number of all attention heads is relevant hallucination detection.



(a) ROC-AUC vs. percentage of retained features, MBPP.

Figure 6: Pruning ability of the proposed method.

6 Conclusions

In this paper, we have proposed a new approach to hallucination detection is code generating LLMs. Our approach is based on the introspection of a LLM: we get attention maps for a prompt and generation and study their topology after transforming to weighed graphs. The proposed topological features of these graphs are empirically shown to be relevant to detection of code hallucinations. A classifier built on top of these features outperformed several baselines. These classifiers are transferable across coding benchmarks. The natural extension of our research is detection of specific places of code with bugs, we leave it for a further research. We believe that our work may lead to a wider application of code generating LLMs by making them more reliable. In a wider context, our work contributes to study of interpretation and generalization in NLP models since hallucinations and generalization ability are intrinsically tied.

7 Limitations

Although we have achieved good experimental results, we realize that our research have several limitations. First of all, we explored only code LLMs having no more than 7B parameters. Information in larger models are more distributed in attention heads and results might differ. Also, processing more attention heads is computationally costly. Next, the proposed classifiers of hallucinations are based on the attention maps of the same code LLMs as for code generations. We leave more general setting to a further research. Finally, our approach can predict whether a code is correct as a whole but can't point to a specific place with a bug.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

S. Barannikov. 1994. Framed Morse complexes and its invariants. *Adv. Soviet Math.*, 22:93–115.

Serguei Barannikov, Ilya Trofimov, Grigorii Sotnikov, Ekaterina Trimbach, Alexander Korotin, Alexander Filippov, and Evgeny Burnaev. 2021. Manifold topology divergence: a framework for comparing data manifolds. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 7294–7305.

Frédéric Chazal and Bertrand Michel. 2017. An introduction to topological data analysis: fundamental and practical aspects for data scientists. *CoRR*, abs/1710.04019.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Daniil Cherniavskii, Eduard Tulchinskii, Vladislav Mikhailov, Irina Proskurina, Laida Kushnareva, Ekaterina Artemova, Serguei Barannikov, Irina Piontkovskaya, Dmitri Piontkovski, and Evgeny Burnaev. 2022. Acceptability judgements via examining the topology of attention maps. *arXiv preprint arXiv:2205.09630*.

Kevin Clark. 2019. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*.

David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. 2005. Stability of persistence diagrams. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 263–271.

Mohamed Elaraby, Mengyin Lu, Jacob Dunn, Xueying Zhang, Yu Wang, Shizhu Liu, Pingchuan Tian, Yuping Wang, and Yuxuan Wang. 2023. Halo: Estimation and reduction of hallucinations in open-source weak large language models. *arXiv preprint arXiv:2308.11764*.

Shangbin Feng, Weijia Shi, Yike Wang, Wenxuan Ding, Vidhisha Balachandran, and Yulia Tsvetkov. 2024. Don't hallucinate, abstain: Identifying llm knowledge gaps via multi-llm collaboration. *arXiv preprint arXiv:2402.00367*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R Bowman. 2019. Do attention heads in bert track syntactic dependencies? *arXiv preprint arXiv:1911.12246*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Nan Jiang, Qi Li, Lin Tan, and Tianyi Zhang. 2024. Collu-bench: A benchmark for predicting language model hallucinations in code. *arXiv preprint arXiv:2410.09997*.

Laida Kushnareva, Daniil Cherniavskii, Vladislav Mikhailov, Ekaterina Artemova, Serguei Barannikov, Alexander Bernstein, Irina Piontkovskaya, Dmitri Piontkovski, and Evgeny Burnaev. 2021. Artificial text detection via examining the topology of attention maps. *arXiv preprint arXiv:2109.04825*.

Jenny T Liang, Chenyang Yang, and Brad A Myers. 2024. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13.

Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. 2024. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, et al. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Yuchen Tian, Weixiang Yan, Qian Yang, Qian Chen, Wen Wang, Ziyang Luo, and Lei Ma. 2024. Codehalu: Code hallucinations in llms driven by execution-based verification. *arXiv preprint arXiv:2405.00253*.

Eduard Tulchinskii, Kristian Kuznetsov, Laida Kushnareva, Daniil Cherniavskii, Serguei Barannikov, Irina Piontkovskaya, Sergey Nikolenko, and Evgeny Burnaev. 2022. Topological data analysis for speech processing. *arXiv preprint arXiv:2211.17223*.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *Preprint*, arXiv:2109.00859.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with OSS-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 52632–52657. PMLR.

Yakir Yehuda, Itzik Malkiel, Oren Barkan, Jonathan Weill, Royi Ronen, and Noam Koenigstein. 2024. Interrogatellm: Zero-resource hallucination detection in llm-generated answers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9333–9347.

Shaolei Zhang, Tian Yu, and Yang Feng. 2024a. Truthx: Alleviating hallucinations by editing large language models in truthful space. *arXiv preprint arXiv:2402.17811*.

Xiaoying Zhang, Baolin Peng, Ye Tian, Jingyan Zhou, Lifeng Jin, Linfeng Song, Haitao Mi, and Helen Meng. 2024b. Self-alignment for factuality: Mitigating hallucinations in llms via self-evaluation. *arXiv preprint arXiv:2402.09267*.


```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
```

problem description

```
    for i in range(len(numbers) - 1):
        for j in range(i+1, len(numbers)):
            if abs(numbers[i] - numbers[j]) <= threshold:
                return True
    return False
```

generation

Figure 7: Example of prompt (problem description) and model generation for the HumanEval dataset.

```
You are an expert Python programmer, and here is your task: Write a function to find the
similar elements from the given two tuple lists. Your code should pass these tests:

assert similar_elements((3, 4, 5, 6),(5, 7, 4, 10)) == (4, 5)
assert similar_elements((1, 2, 3, 4),(5, 4, 3, 7)) == (3, 4)
assert similar_elements((11, 12, 14, 13),(17, 15, 14, 13)) == (13, 14)
[BEGIN]
def similar_elements(test_tup1, test_tup2):
  res = tuple(set(test_tup1) & set(test_tup2))
  return (res)
[DONE]
```

one-shot

```
You are an expert Python programmer, and here is your task: Write a python function to
remove first and last occurrence of a given character from the string. Your code should
pass these tests:

assert remove_Occ("hello","l") == "heo"
assert remove_Occ("abcda","a") == "bcd"
assert remove_Occ("PHP","P") == "H"
[BEGIN]
```

problem description

```
def remove_Occ(s,c):
  return s.replace(c,'',s.count(c)-1)
[DONE]
```

generation

Figure 8: Example of prompt (one-shot example and problem description) and model generation for the MBPP dataset.

A Details on generation procedure

We generated solutions for the coding problems with temperature of 0.8. For the HumanEval dataset, the maximum length of model output (i.e. input prompt + generation) was limited to 512 tokens. For the MBPP dataset, the maximum number of new tokens to generate was set to 256. Figures 7, 8 provide examples of prompt and generation for HumanEval and MBPP datasets. We followed the guidelines[2] to post process the model output and extract the valid problem solution. To compute attention features according to the proposed method in Section 4, we used the attention submatrix corresponding to input prompt and valid problem solution. For computational experiments we used NVIDIA TITAN RTX.

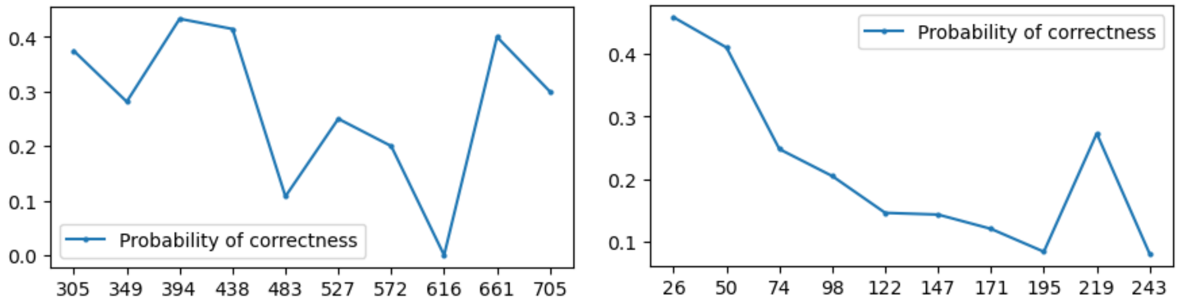
B Details on training procedure

For the code hallucination detectors, based on the XGBoost classifier training, we utilized the XGBClassifier with an approximation tree method "hist" from the XGBoost library [3]. For the code hallucination detector based on the embeddings

[2] https://github.com/bigcode-project/bigcode-evaluation-harness

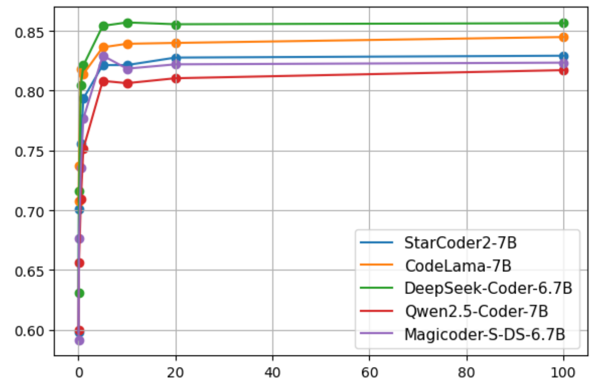
[3] https://xgboost.readthedocs.io/en/latest/index.html

from CodeT5-base, we used the pretrained frozen CodeT5-base encoder with trainable classification head consisting of 2 linear layers with hidden dimensionality 768. The classification head was trained for 100 epochs with batch size 32 and learning rate $3e-5$.

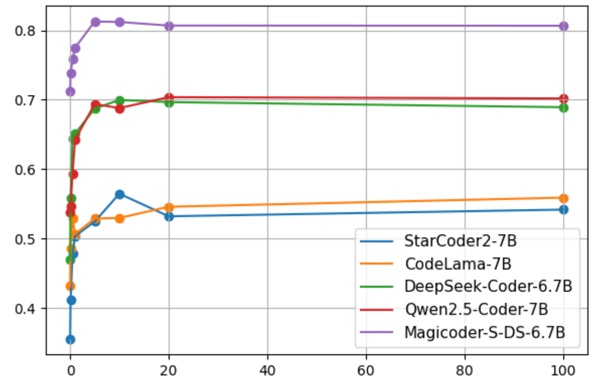


(a) Prompt length, tokens. MBPP (b) Generation length, tokens. MBPP

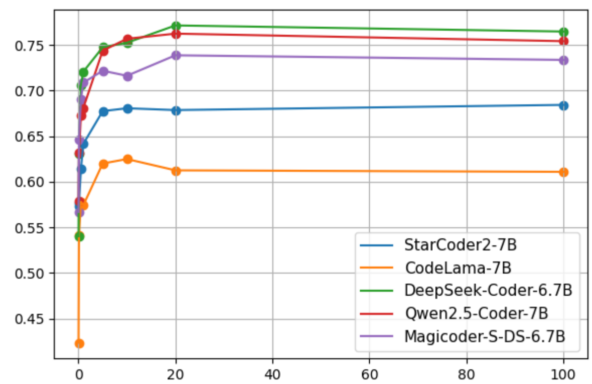
Figure 9: The individual conditional expectations for prompt and generation lengths, CodeLlama.



(a) ROC-AUC vs. percentage of retained features, HumanEval.



(b) F1, HumanEval



(c) F1, MBPP

Figure 10: Pruning ability of the proposed method.