ContraCLM: Contrastive Learning For Causal Language Model

Nihal Jain[*], **Dejiao Zhang**[*], **Wasi Uddin Ahmad**[*],
Zijian Wang, **Feng Nan**, **Xiaopeng Li**, **Ming Tan**, **Ramesh Nallapati**,
Baishakhi Ray, **Parminder Bhatia**, **Xiaofei Ma**, **Bing Xiang**
AWS AI Labs, USA

Abstract

Despite exciting progress in causal language models, the expressiveness of their representations is largely limited due to poor discrimination ability. To remedy this issue, we present ContraCLM, a novel contrastive learning framework at both the token-level and the sequence-level. We assess ContraCLM on a variety of downstream tasks. We show that ContraCLM enhances the discrimination of representations and bridges the gap with encoder-only models, which makes causal language models better suited for tasks beyond language generation. Specifically, we attain 44% relative improvement on the Semantic Textual Similarity tasks and 34% on Code-to-Code Search tasks. Furthermore, by improving the expressiveness of representations, ContraCLM also boosts the source code generation capability with 9% relative improvement on execution accuracy on the HumanEval benchmark. [1]

1 Introduction

Causal Language Models (CLM) have seen remarkable success in language generation, both in natural language (Radford et al., 2018, 2019; Brown et al., 2020) and programming language (Chen et al., 2021; Nijkamp et al., 2022). However, one limitation at their core is the poor discrimination ability of the representations, which often causes a large performance gap with encoder-only or encoder-decoder models on discriminative tasks (see Appendix D.1), and hence limits the wide usage of CLM beyond language generation.

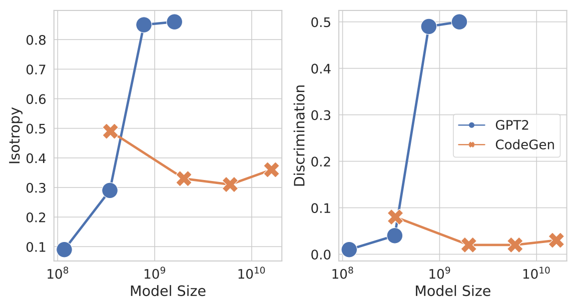
Prior studies posit that the *anisotropy* issue, *i.e.,* representations being squeezed into a tiny cone in the vector space (Ethayarajh, 2019), can be the main cause of the poor discrimination ability of language models across different architectures and objectives. Many efforts have focused on resolving the anisotropy issue on encoder-only or encoder-decoder models, either through post-processing (Su et al., 2021; Li et al., 2020) or integrating different regularization terms into the training objective (Gao et al., 2019; Wang et al., 2020). A recent work (Su and Collier, 2022) shows that the decoder-only CLM does not suffer from the anisotropic problem as long as the model is beyond a certain size. However, we find that such conclusions can vary across domains. As shown in Figure 1a, CLMs pretrained on text, *i.e.,* GPT-2 (Radford et al., 2019), do yield representations with good isotropy and discrimination as long as the model is not smaller than 774M parameters (GPT2-Large), whilst CodeGen (Nijkamp et al., 2022), pretrained on programming language data, consistently suffers from anisotropy and poor discrimination across different model sizes. Therefore, an effective training strategy is still essential for CLMs to improve representation quality with better isotropy and discrimination (Figure 1b). We conjecture that this is essential not only for models suffering from inferior representations, *e.g.,* CodeGen, and GPT2 (124M) but also for those with a good starting point (suffer less *e.g.,* GPT2-Large (774M)).

We argue that an ideal CLM should yield isotropic representations to better leverage the representation space, as well as discriminative representations such that tokens or sequences from the same context are mapped to comparatively closer locations in the vector space compared to those from randomly sampled contexts. To this end, we developed ContraCLM, a novel contrastive learning framework at both the token-level and sequence-level.

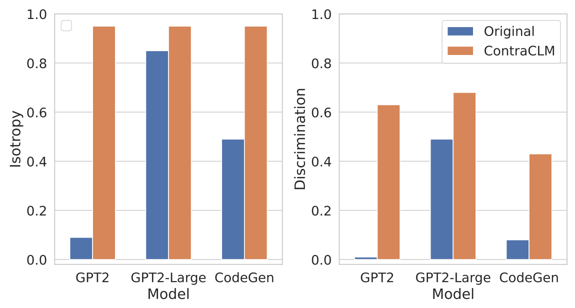
ContraCLM is able to promote more uniformly distributed and hence isotropic representations by separating the instances at different semantic levels, *e.g.,* tokens or sequences, apart from each

[*]Equal Contribution. Correspondence to Dejiao Zhang <dejiaoz@amazon.com>.

[1]We release our code at `https://github.com/amazon-science/ContraCLM`.



(a) The isotropy and discrimination abilities of the representations yielded by a model do not always improve with increase in model sizes.



(b) ContraCLM can effectively enhance both isotropy and discrimination, regardless of whether the original models suffer from degenerate representations or not.

Figure 1: Evaluating the representation quality with respect to both <u>isotropy</u> $:= 1 - \textit{intra-similarity}$ and <u>discrimination</u> $:= 1 - \textit{intra-similarity} / \textit{inter-similarity}$, where *intra-similarity* refers to the average cosine similarity between tokens from the same sequence and inter-similarity is defined with respect to tokens from two randomly sampled sequences. We use WIT (Srinivasan et al., 2021) and code search dataset (Guo et al., 2022) to evaluate GPT2 and CodeGen models, respectively.

other. ContraCLM improves the discrimination of representations due to the implicit grouping effect on semantically similar instances, yielded by pulling together the variations that preserve semantics or positive pairs, of the same instance (Wang and Isola, 2020; Wang and Liu, 2021; Zhang et al., 2021).

A natural question arises as to how would the improved representations affect the generation ability of CLMs. Towards addressing this, we assess ContraCLM on language generation tasks in different domains, where we achieve better MAUVE (Pillutla et al., 2021) on text generation and 9% relative improvement on pass@1 accuracy on HumanEval (Chen et al., 2021). The improvement in code completion is indeed significant as it reflects that more model-generated programs pass a suite of test cases. On the discriminative tasks, ContraCLM attains 44% relative improvement on Semantic Textual Similarity tasks and 34% on Code-to-Code Search tasks, which largely bridges the gap with the encoder-only or encoder-decoder models (see Section 4.4 and Appendix D.1). Such improvements allow us to boost the performance of decoder-only models on a wide range of discriminative tasks where encoder-only models are currently the workhorse.

2 Related Work

Anisotropic Representation of Language Models Despite the remarkable success achieved by language models (Devlin et al., 2019; Radford et al., 2019; Yang et al., 2019; Raffel et al., 2020; Lewis et al., 2020), they suffer from the *anisotropy* issue where the representations are distributed into a tiny cone in the vector space (Gao et al., 2019; Ethayarajh, 2019; Li et al., 2020; Wang et al., 2020). In particular, Ethayarajh (2019) shows that the degeneration is severer on CLM, where the average cosine similarity between two words sampled from randomly selected sequences is almost at one when evaluating the outputs from the last hidden layer of GPT-2 (Radford et al., 2019). However, Su and Collier (2022) show that CLMs (Radford et al., 2019) are indeed coherent as long as the model is larger than a certain size. We find such conclusions can vary across domains, *e.g.,* when pretraining on code, CodeGen (Nijkamp et al., 2022) consistently suffers from the anisotropy issue over a wide range of model sizes. On the bright side, Figure 1b shows that ContraCLM can effectively improve the representation quality when we continue to train the existing CLMs with our proposed objectives, regardless of whether the CLMs suffer from inferior representations initially.

Contrastive Learning Contrastive learning (Chen et al., 2020; He et al., 2020) has seen remarkable successes in Natural Language Processing (NLP). A large amount of research has focused on sentence representation learning for encoder-only models, with the main differences lying in how the augmentations are generated (Fang and Xie, 2020; Giorgi et al., 2021; Wu et al., 2020; Meng et al., 2021; Yan et al., 2021; Kim et al., 2021; Gao et al., 2021; Zhang et al., 2022). Recently there is an emerging interest in developing effective contrastive learning approaches for text generation models. However, most existing work mainly focuses on the encoder-decoder structure (Dong et al., 2019; Raffel et al., 2020; Lewis et al., 2020) by contrasting suboptimal model generations obtained via diverse sampling (An et al., 2022) or adding perturbations on the embedding space (Lee et al., 2021), against the ground truth. On the other hand, it is not intuitive to develop an effective contrastive learning strategy for decoder-only models. A recent work (Su et al., 2022) proposes SimCTG, a token-level contrastive learning approach that aims to separate each token apart from others within the same sequence by a predefined distance. As shown in Section 4, our temperature-based token-level contrastive learning approach, CONTRACLM-TOK, consistently outperforms SimCTG across different tasks. We conjecture that the fixed margin-based objective allows less flexibility for the token-level representation separation, especially considering how the semantic relevance among tokens can vary across contexts (sequences).

Code Generation and Beyond Language modeling for source code is a fast growing area of research. Various model architectures have been explored recently, including encoder-only (Feng et al., 2020; Guo et al., 2021), encoder-decoder (Ahmad et al., 2021; Wang et al., 2021; Li et al., 2022), and decoder-only models (Chen et al., 2021; Nijkamp et al., 2022; Chowdhery et al., 2022). Among them, the decoder-only models have been found to be effective for code generation. However, as shown in Section 4.3.2 and Appendix D.1, they suffer from unsatisfactory performance on various discriminative tasks (Lu et al., 2021; Huang et al., 2021; Guo et al., 2022). This motivates us to improve the decoder-only models on the discriminative tasks to extend their main usage beyond language generation. Furthermore, code is fundamentally different from natural language in that it is more structured, which helps validate the generalization of our approach beyond plain text.

3 Model

3.1 Causal Language Modeling

Let $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{|\mathbf{x}|}]$ denote a sequence with variable length $|\mathbf{x}|$, *e.g.,* a piece of text or a code snippet. Causal Language Modeling (CLM) is usually formulated as sequence distribution estimation over a set of sequences, $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N$. For tractable estimation, common practice is to factorize the joint distribution of each sequence into the product of conditional token prediction probabilities. The model is then trained via maximum likelihood estimation as follows,

$$\mathcal{L}_{\text{CLM}} = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{|\mathbf{x}^j|} \log p(\mathbf{x}_i^j | \mathbf{x}_{<i}^j) \; .$$

Here $\mathbf{x}_{<i}^j = [\mathbf{x}_1^j, \ldots, \mathbf{x}_{i-1}^j]$ denotes the subsequence before $\mathbf{x}_i^j$ and $|\mathbf{x}^j|$ is the sequence length.

3.2 Contrastive Learning for CLM

Let $\mathbf{h}^{(i)}, \mathbf{h}^{(i^+)}$ denote two representation variations of the same instance that preserve semantics, or a positive pair for contrastive learning. Then denote $\mathcal{I} = \{1, 2, \ldots, N\} \cup \{1^+, 2^+, \ldots, N^+\}$ as the set of representation indices associated with N instances. Further, let τ denote the temperature hyper-parameter and $\diamond$ denote cosine similarity. We then minimize the following,

$$\mathcal{L} = \sum_{j=1}^{N} - \left(\log \frac{\exp(\mathbf{h}^{(j)} \diamond \mathbf{h}^{(j^+)}/\tau)}{\sum_{k \in \mathcal{I} \setminus j} \exp(\mathbf{h}^{(j)} \diamond \mathbf{h}^{(k)}/\tau)} \right.$$
$$\left. + \log \frac{\exp(\mathbf{h}^{(j^+)} \diamond \mathbf{h}^{(j)}/\tau)}{\sum_{k \in \mathcal{I} \setminus j^+} \exp(\mathbf{h}^{(j^+)} \diamond \mathbf{h}^{(k)}/\tau)} \right) .$$

Note that in our setting, an instance can refer to either a token or a sequence. When $\mathbf{h}^{(j)}, \mathbf{h}^{(j+)}$ denote a pair of representation variations of the j-th token within a sequence, N is the sequence length that can vary across sequences; in this case the objective is $\mathcal{L}_{\text{Tok}}$. For the sequence-level contrastive loss, $\mathbf{h}^{(j)}, \mathbf{h}^{(j+)}$ refer to the pair of representations of the j-th sequence within a batch, and N denotes the batch size; in this case the objective is $\mathcal{L}_{\text{Seq}}$.[2]

Therefore, when applied at both token-level and sequence-level, the contrastive learning objective defined above tries to separate tokens at each distinct location apart from every other token within the same sequence, and sequences within the same randomly sampled batch apart from each other. Intuitively, such separation can improve the uniformity (isotropy) of the representations. Further, better discriminative representations are achieved due to the implicit grouping effect of contrastive learning on semantically similar instances. Such grouping effect of contrastive learning has been studied in recent work (Wang and Liu, 2021; Zhang et al., 2021; Wang and Isola, 2020) as well.

[2]Please refer to Appendix A for the complete formulations.

3.3 ContraCLM

In addition to the causal language modeling loss, ContraCLM optimizes the contrastive learning objective defined in Equation (1) at both the token-level ($\mathcal{L}_{\text{Tok}}$) and sequence-level ($\mathcal{L}_{\text{Seq}}$) as follows

$$\mathcal{L}_{\text{ContraCLM}} = \mathcal{L}_{\text{CLM}} + \mathcal{L}_{\text{Tok}} + \mathcal{L}_{\text{Seq}} \ .$$

Furthermore, to understand how the token-level and sequence-level contrastive learning contribute to the overall performance, we assess the performance of $\mathcal{L}_{\text{ContraCLM-Tok}} = \mathcal{L}_{\text{CLM}} + \mathcal{L}_{\text{Tok}}$ and $\mathcal{L}_{\text{ContraCLM-Seq}} = \mathcal{L}_{\text{CLM}} + \mathcal{L}_{\text{Seq}}$ in Section 4. Unless otherwise specified, we weigh each loss equally and set the temperature $\tau = 0.05$. Although better performance can be achieved by hyperparameter optimization, we mainly investigate how ContraCLM improves the representation quality and the zero-shot transfer learning performance. We hence leave hyperparameter optimization in a supervised setting as future work.

Positive pair of representations For GPT-2 (Radford et al., 2019), we consider the simple yet effective dropout-based augmentation (Gao et al., 2021), where the positive pair of representations is obtained by performing a forward pass of the same sequence twice. On the other hand, for CodeGen (Nijkamp et al., 2022), we simply duplicate the representation of each instance as positive pair for an apples-to-apples comparison since dropout is disabled during its initial pretraining stage. Unlike the existing findings that the dropout-based augmentation can boost the contrastive learning performance when (continually) training a language model, we find that the trends can vary when evaluating on discrimination tasks and generation tasks. A detailed ablation study can be found in Section 4.4 and Appendix D.2.

4 Experiments

To demonstrate the effectiveness of our proposed framework in different application domains, we evaluate our models and baselines on natural language and programming language tasks. We design our experiments to address – (1) Does contrastive learning improve the *discrimination ability* of representations? (2) Do the representations learned by contrastive learning lead to better performance on *language generation* tasks? (3) Is the joint contrastive learning at both token- and sequence-level necessary, and how do they benefit from each other? (4) How does the impact of contrastive learning vary across language domains?

4.1 Data and Models

Data & Models For text, we continue training GPT-2 (124M) (Radford et al., 2019) on WikiText-103, a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia (Merity et al., 2017). For code, we continue training CodeGen 350M monolingual (Nijkamp et al., 2022) on collected permissively licensed Python code from GitHub. Please refer to Appendix B for the training details. We consider the following objectives for the continual training of both GPT-2 and CodeGen:

- **CLM**. The standard left-to-right autoregression objective for training causal language models, which is also the objective used for pretraining both GPT-2 and CodeGen.
- **SimCTG** (Su et al., 2022). A predefined margin[3] based token-level contrastive learning framework that aims to separate tokens at each distinct location within a sequence apart from each other.
- **ContraCLM-Tok & ContraCLM-Seq**. As defined in Section 3.3, these two are obtained by combining the CLM objective with our proposed token-level or sequence-level contrastive loss, respectively. This investigation allows us to better understand how our token-level and seqeunce-level contrastive losses contribute to the overall performance of ContraCLM.

4.2 Evaluation on Natural Language

We first evaluate our model on discrimination and generation tasks in natural language.

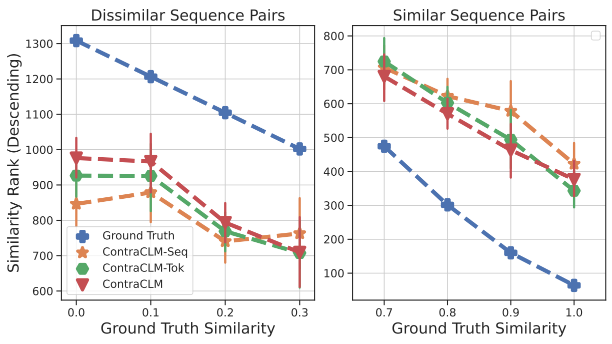
4.2.1 Semantic Textual Similarity

We assess ContraCLM on semantic textual similarity (STS), the most commonly used benchmark for evaluating the semantic discrimination capability of representations. STS consists of seven tasks, namely STS 2012-2016 (Agirre et al., 2012, 2013, 2014, 2015, 2016), the STS Benchmark (Cer et al., 2017), and the SICK-Relatedness (Marelli et al., 2014). In this benchmark, human annotators provide a fine-grained similarity score from 0 to 5 for each sequence pair. Following Reimers and Gurevych (2019), for the sequence pairs in each dataset, we report the overall Spearman's correlation between the cosine similarities of representa-

[3]For all experiments in this section, we set the margin $\rho = 0.5$ as recommended in Su et al. (2022).

Model	STS12	STS13	STS14	STS15	STS16	SICK-R	STS-B	Avg.
GPT2	25.84	28.90	26.20	34.74	35.70	42.72	26.27	31.48
CLM	27.14	20.34	18.73	37.56	27.40	35.70	27.97	27.83
SimCTG	30.32	37.10	31.99	39.68	42.73	46.26	25.27	36.19
ContraCLM-Tok	37.28	37.63	31.33	54.78	50.16	48.10	34.95	42.03
ContraCLM-Seq	29.66	39.89	34.50	43.20	41.99	44.52	25.51	37.04
ContraCLM	**37.54**	**45.23**	**36.41**	**56.74**	**50.30**	**51.52**	**39.49**	**45.32**

Table 1: Spearman rank correlation between the cosine similarity of sentence representation pairs and the ground truth similarity scores.



Similar Sequence Pair	Model	Rank↓
S1: a woman is stabbing a potato with a fork	Ground Truth	40
	ContraCLM-Seq	501
S2: a woman is punctu-ring a potato with a fork	ContraCLM-Tok	272
	ContraCLM	**251**
Dissimilar Sequence Pair	**Model**	**Rank↑**
S1: a man is opening a box and taking out paper	Ground Truth	1310
	ContraCLM-Seq	400
S2: a woman is peeling a potato	ContraCLM-Tok	1054
	ContraCLM	**1181**

Figure 2: ContraCLM-Tok is essential for making the sequence-level representations robust to spurious patterns of words or phrases (results reported on STS-B). We scale the ground truth similarity scores from [0, 5] to [0,1].

tions and the human-provided similarity scores in Table 1.

Effectively Enhancing Discrimination Table 1 shows that both GPT-2 and the one continually trained with CLM perform poorly on STS, which is a consequence of poor discrimination: the cosine similarities between semantically similar or dissimilar pairs are both almost at one (Figure 4 in Appendix C.1). Also note that continuing to train GPT-2 with CLM on WikiText-103 worsens performance, which can occur since the domains of WikiText-103 and the STS datasets are different.[4] In contrast, both ContraCLM and SimCTG largely outperform GPT-2, yet still, ContraCLM attains 25% relative improvement over SimCTG. Moreover, ContraCLM-Tok outperforms SimCTG on almost all STS benchmarks and the trend remains the same even without the dropout-based augmentation (Appendix D.3). Therefore, we posit that our temperature-based contrastive learning objective allows more flexibility towards separating representations based on token semantics, whereas requiring a predefined separation margin between tokens (as SimCTG does) is not ideal.

[4]STS datasets include text from image captions, news headlines and user forums. As a result, adapting GPT-2 to WikiText-103 reduces its transfer ability.

ContraCLM-Tok vs. ContraCLM-Seq Table 1 also indicates that ContraCLM-Tok and ContraCLM-Seq complement each other, as ContraCLM consistently performs better than both of them on STS. Note that ContraCLM-Seq performs worse than ContraCLM-Tok. It is surprising, especially since STS mainly assesses the sequence-level representation quality. We investigate this by dividing the sequence pairs into two groups – semantically similar pairs with human-annotated similarity scores no less than 0.7 and dissimilar pairs with human scores no larger than 0.3. We plot the rank of the model inferred similarity scores against the human similarity scores in Figure 2 (left). As we can see, ContraCLM-Seq struggles in ranking semantically dissimilar sequence pairs higher and similar pairs lower. This suggests that the token-level contrastive loss is essential for making the sequence-level representations robust to spurious patterns of tokens or phrases, *e.g.,* ranking semantically similar sequences with different synonyms low and dissimilar sequences high even in presence of the same phrase (Figure 2 (right)).

4.2.2 Text Generation

Next, we assess the open-ended language generation capability, where each model is required to generate text continuations given the prefixes from the

WikiText-103 test set. Following Su et al. (2022), we set the lengths of prefix and continuation to 32 and 128, respectively. We use nucleus sampling (Holtzman et al., 2020) with top-$p = 0.95$. In addition to Perplexity (PPL; evaluated on the ground truth only) and MAUVE, we also evaluate the discrimination of representations of generated text under different settings in Table 2.

ContraCLM Leads to More Semantically Coherent Generations It is desired that contextual token representations within the same or semantically similar sequences have relatively higher similarities among each other when compared to similarities between tokens sampled from random contexts. Therefore, given a prompt, lower discrimination scores are desired between the ground truth and generation, while higher discrimination values are desired between generations for randomly sampled prompts.

As reported in Table 2, compared to CLM, ContraCLM attains much better discrimination on the generations under dissimilar context (prompts) pairs, as indicated by the high value of Disc(D). Further, ContraCLM and ContraCLM-Tok achieve better or at least comparable semantic coherence between the generation and the ground truth, as indicated by the MAUVE scores. We argue that, the zero valued discrimination score between generation and ground truth, i.e., Disc(S), attained by GPT-2 and CLM does not imply better semantic coherence – this is a consequence of their inferior representations evidenced by the zero discrimination score between semantically irrelevant sequences.

Finally, a slight increase in PPL is probably expected, considering that PPL is better aligned with the standard CLM objective. Thereby, contrastive learning can be interpreted as a regularization that trades off between PPL and the desired representation properties.

4.3 Evaluation on Programming Language

In this section, we study the effectiveness of our proposed contrastive learning framework on programming language applications – code search, code completion, and code re-ranking. Since CodeGen models are pretrained without dropout activations, we follow the same for our models in this subsection helping us study the effectiveness of ContraCLM without dropout augmentations. We also investigate how dropout would affect the decoder-only models when evaluated on the downstream tasks in Section 4.4 and Appendix D.2.

Model	Generated Text			
	PPL↓	MAUVE↑	Disc(S)↓	Disc(D)↑
GPT-2	47.50	0.893	**0.00**	0.00
CLM	**22.48**	0.945	**0.00**	0.01
SimCTG	22.51	0.952	0.11	0.54
ContraCLM-Tok	22.99	**0.953**	0.12	0.49
ContraCLM-Seq	22.60	0.933	0.23	**0.83**
ContraCLM	23.01	0.947	0.18	0.62

Table 2: Evaluation on the Wikitext-103 test set. Disc(D) is the discrimination score computed between the generated continuations of two randomly sampled prompts. Disc(S) is computed between the ground truth text and the generated one associated with the same prompt. A lower Disc(S) indicates better coherence with the ground truth, while a higher Disc(D) indicates better representation discrimination among the generations under different contexts. All metrics are evaluated over the entire test set.

4.3.1 Code Search

Code search is the task of retrieving relevant code fragments given a code fragment as a *query*. We perform in-language (query and relevant code are in the same language) and cross-language (query and relevant code are in different languages) code searches. We provide an example in Figure 5 (Appendix C.2.2). In this study, we experiment in the *zero-shot* setting - we use the models described in Section 4.1 to generate dense representations of code and perform a nearest neighbor search to retrieve relevant code fragments. We use publicly available implementations of Guo et al. (2022).[5]

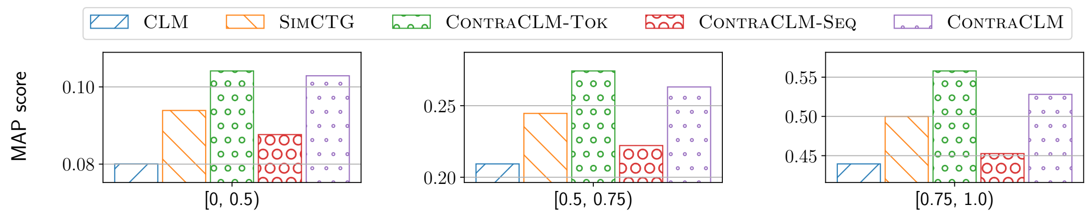
Contrastive Learning Yields More Discriminative Code Representations For the code-to-code search task, Guo et al. (2022) used problem solutions in Ruby, Python, and Java languages from CodeNet (Puri et al., 2021). They propose to use each program as a query and retrieve all programs that solve the same problem. We present detailed statistics of the dataset in Table 6 (Appendix C.2.2). We set the maximum sequence length as 512[6] and use cosine similarity between two mean vectors of the last hidden states as relevance scores. We then sort the candidates by their scores to calculate the Mean Average Precision (MAP) score. We present

[5]https://github.com/microsoft/CodeBERT/tree/master/UniXcoder/downstream-tasks

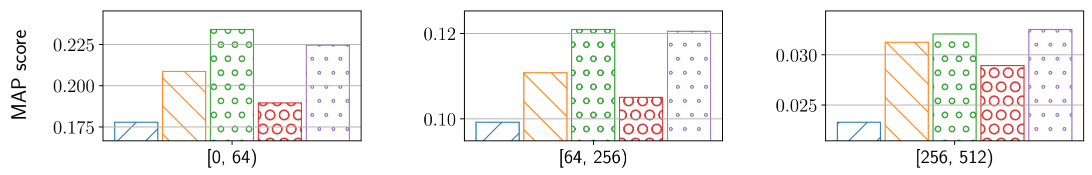
[6]We also performed experiments with maximum length 1024 but didn't observe any significant difference.

Model	Ruby			Python			Java			Avg.
	Ruby	Python	Java	Ruby	Python	Java	Ruby	Python	Java	
CodeGen	16.18	5.90	0.52	2.66	18.11	0.36	1.61	1.65	10.16	6.35
CLM	16.36	6.67	0.80	3.07	15.72	0.46	1.41	2.11	10.25	6.32
SimCTG	17.66	7.19	1.94	7.63	18.31	1.78	1.63	2.32	10.83	7.70
ContraCLM-Tok	**18.02**	**7.84**	2.51	8.76	**20.46**	2.48	**1.91**	**2.58**	**11.43**	**8.44**
ContraCLM-Seq	16.76	5.45	1.06	7.40	16.74	1.41	1.55	2.25	10.23	6.98
ContraCLM	17.90	7.78	**2.56**	**9.05**	19.74	**2.64**	1.90	2.50	11.32	8.38

Table 3: MAP score (%) of the zero-shot code search task. The language names mentioned in the top two rows indicate the languages queries and candidates are written in.



(a) Performance breakdown based on edit similarities (x-axis).



(b) Performance breakdown based on length differences (x-axis).

Figure 3: Code search performances based on (a) and (b) between the query code fragments (in Python) and their relevant code fragments (in Python). We observe that in both cases, ContraCLM-Tok outperforms CLM, SimCTG, and ContraCLM-Seq.

the results for the code search tasks in Table 3.[7]

We observe ContraCLM-Tok and ContraCLM frameworks improve upon CodeGen trained with CLM by 33.5% (absolute 2.12) and 32.6% (absolute 2.06) on average, respectively. We also point out that the performance gap between ContraCLM-Tok and SimCTG are apples-to-apples comparisons since the dropout-based augmentation is not used in either models. As aforementioned, the consistently better performance of ContraCLM-Tok suggests the superiority of our temperature-based contrastive learning objective. On the other hand, ContraCLM-Seq improves over the CLM baseline by 10.4% only. Code search results indicate that ContraCLM-Seq performs poorly compared to ContraCLM-Tok. This performance gap is larger than what we observed in the natural language evaluation. We conjecture that ContraCLM-Tok generates better discriminative representations for code sequences since the finer-grained understanding of the code tokens is crucial to understanding the code sequences' functionality (semantics). To verify this, we check if non-semantic factors impact model performances in the following section.

Token-level Contrastive Learning is Effective for Code Understanding We break down the code search performance based on edit similarities and length differences between query code and their relevant code fragments. While edit similarity indicates how much queries and their relevant code overlap, the length difference indicates whether models effectively capture relevance between two code fragments if they are similar in length or differ significantly. We present the results for Python language in Figure 3 (for all the languages, see Figures 7 & 8 in Appendix C.2.3). The results show that ContraCLM-Tok outperforms CLM, SimCTG, and ContraCLM-Seq irrespective of edit similarities and length differences. Therefore, we can

[7]We present a comparison with encoder-only and encoder-decoder models in Table 7b in the Appendix.

Model	Pass@k		Ranked Pass@k	
	k=1	k=5	k=1	k=5
CodeGen	12.65	16.89	13.42 $_{(+0.77)}$	17.07 $_{(+0.18)}$
CLM	13.42	18.08	15.38 $_{(+1.96)}$	18.29 $_{(+0.21)}$
SimCTG	13.26	17.29	15.24 $_{(+1.98)}$	18.29 $_{(\mathbf{+1.00})}$
CONTRACLM-TOK	12.96	17.01	15.24 $_{(+2.96)}$	17.68 $_{(+0.67)}$
CONTRACLM-SEQ	13.64	15.85	16.99 $_{(\mathbf{+3.35})}$	16.46 $_{(+0.61)}$
CONTRACLM	**14.63**	**18.83**	**17.07** $_{(+2.44)}$	**18.90** $_{(+0.07)}$

Table 4: Evaluation results on the HumanEval benchmark. The numbers in the subscript indicate the difference between ranked pass@k and pass@k. While CONTRACLM-TOK and CONTRACLM-SEQ perform competitively to the baselines, CONTRACLM significantly outperforms them.

conclude that sequence overlap or length are not the reasons for improvements in CONTRACLM-TOK. Presumably, a finer-grained understanding of code tokens makes CONTRACLM-TOK more effective for code representations.

4.3.2 Code Completion and Re-Ranking

Given a sequence of tokens composed of natural language, function signature, and input-output examples (as a whole, we call them prompt), the goal of the code completion task is to complete the function. To evaluate the functional correctness of a complete code, we use existing benchmarks that include unit tests. If the generated code successfully passes the unit tests, we refer to this as successful execution. We compute pass@k for $k \leq n$ following (Chen et al., 2021). In addition, we compare the models on the code re-ranking task – given n sampled code using a code completion model, the goal is to order the generated samples, for which we use the mean log probability of each sampled code (Chen et al., 2021). For code re-ranking evaluation, we report ranked pass@k (Inala et al., 2022). Figure 6 (Appendix C.2.2) illustrates both the code completion and re-ranking tasks. We detail the evaluation metrics in Appendix C.2.1.

Contrastive Learning Improves Source Code Generation Chen et al. (2021) introduced HumanEval, a collection of 164 handwritten programming problems and their respective unit tests. Each problem in this dataset is presented using a prompt for a function, and the task is to complete the function, such that it can pass all unit tests. In all our experiments, we use nucleus sampling (Holtzman et al., 2020) with top $p = 0.95$. We sample $n = 10$ completions per problem with sampling temperature 0.2. Table 4 presents the evaluation results on the HumanEval benchmark.

While CONTRACLM-TOK and CONTRACLM-SEQ perform comparably to CLM and SimCTG, CONTRACLM outperforms them significantly, *i.e.,* by 9% and 10.3% in terms of pass@1 accuracy respectively, and by 11% and 12% in terms of ranked pass@1 accuracy, respectively. While CONTRACLM-SEQ underperforms in code completion, it boosts code re-ranking significantly. We hypothesize the improvement is due to the contrastive learning's alignment with the mean log probability-based re-ranking choice.

4.4 Discussion

Impact of Dropout Dropout-based augmentation (Gao et al., 2021) for contrastive learning on language models has shown to have a significant improvement on discriminative tasks. We observe the same trend on both GPT-2 and CodeGen (see Table 8 in Appendix D.2). However, we observed the opposite for language generation, no matter when training with CLM only or with contrastive learning (see Table 9 in Appendix D.2). Dropout has been one of the key ingredients for training large models. Further investigation on proper ways to use and evaluate it are indeed required. Nevertheless, even without dropout, Section 4.3 shows CONTRACLM still yields significant improvement.

Bridge the Gap In comparison with the causal (left-to-right) attention mechanism of the decoder-only models, the bidirectional attention mechanism better leverages the context of sequences, yielding better representations for discriminative tasks. Take the encoder-only models as an example: as Table 7a in Appendix shows, both BERT-Base (Devlin et al., 2019) and RoBERTa-Base (Liu et al., 2019) outperform GPT-2 by at least 60% relative performance on STS. Although the performance gap between CodeGen and the encoder-only or encoder-decoder models decreases in Table 7b, it is still significant considering that both the model and pretraining data sizes used by CodeGen are much larger. Such a large performance gap severely limits the usage of decoder-only models in many discriminative tasks. On the bright side, contrastive learning shows the promise to bridge the gap, *e.g.,* reducing the relative performance gap between GPT-2 and the encoder-only models by at least 50% when evaluating on STS (see Table 7a). Please refer to Appendix D.1 for more detailed discussions.

5 Conclusion

In this paper, we present CONTRACLM, an effective contrastive learning framework to resolve the representation degeneration issue of CLMs trained with the autoregression objective. We assess the effectiveness of CONTRACLM on various downstream tasks in both the natural language and code domains, where we attain significant improvements on both discrimination and generation tasks. While we explored only the decoder-only CLMs, our proposed contrastive learning framework can serve as a drop-in term for encoder-decoder, encoder-only, or prefixLM models also. We leave these explorations as future work.

Limitations

While our work displays many strengths, we highlight some limitations. First, we focus on Python for programming language evaluation, which is one of the most widely used programming languages. However, we believe that our proposed approach, CONTRACLM, would benefit Code LMs trained on any programming language. Second, the empirical findings presented in this work are mainly based on the smaller versions of GPT-2 and CodeGen with 124M and 350M parameters, respectively. However, as shown in Figure 1b, by continuing to train the pretrained models with our proposed objective, CONTRACLM is able to address not only the isotropy and poor discrimination issue that both GPT2-small and CodeGen suffer from, but also improve the representation quality of GPT2-large which has a good starting point for both isotropy and discrimination. Therefore, we believe the effectiveness of CONTRACLM should be applicable to larger versions of these LMs, regardless of whether they suffer from the anisotropy issue (*e.g.,* large CodeGen models) or not (large scale GPT-2 models). We leave the explorations of larger models as future work.

Ethics Statement

Training data We use WikiText-103 and source code in Python from permissively licensed GitHub repositories to train GPT2 and CodeGen, respectively. We do not perform any preprocessing that would get rid of any personally identifiable information or offensive content. However, the use of code LMs comes with certain risks, e.g., generating biased, toxic, and insecure code. We refer readers to Chen et al. (2021) (Section 7) for a detailed discussion on the broader impact of code LMs.

Compute We use an in-house cluster of 128 A100s for all jobs in this paper. Each run takes a couple of hours to one day to finish, depending on the configuration and the model size. We performed one round of training for each setting as it is very expensive to repeat them multiple times. However, we perform the code completion and reranking evaluation with three seeds. STS and code search evaluation do not need multiple runs of inference (as the predictions are deterministic).

Author Contributions

Dejiao and Wasi proposed the initial framework for CONTRACLM and completed the paper writing. Nihal and Dejiao setup the pretraining code. Nihal processed the pretraining data for the programming language experiments. Dejiao designed and completed all natural language related training and evaluations. Nihal and Wasi completed the associated counterparts for programming language data. Zijian was in-charge of the pretraining data collection and multinode distributed training of CONTRACLM models on the programming language data. Feng and Xiaopeng helped with our preliminary explorations on natural language data evaluation. All the other co-authors provided thought-provoking discussions and suggestions for this project, and helped shape and proofread the paper draft.

Acknowledgments

We thank all the helpful discussions and comments from colleagues at AWS AI Labs.

References

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Iñigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, and Janyce Wiebe. 2015. SemEval-2015 task 2: Semantic textual similarity, English, Spanish and pilot on interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 252–263.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. SemEval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 81–91.

Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2016. SemEval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 497–511.

Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. 2012. SemEval-2012 task 6: A pilot on semantic textual similarity. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 385–393, Montréal, Canada. Association for Computational Linguistics.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43.

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668, Online. Association for Computational Linguistics.

Chenxin An, Jiangtao Feng, Kai Lv, Lingpeng Kong, Xipeng Qiu, and Xuanjing Huang. 2022. CoNT: Contrastive neural text generation. In *Advances in Neural Information Processing Systems*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14.

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*, abs/2107.03374.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljacic, Shang-Wen Li, Scott Yih, Yoon Kim, and James Glass. 2022. DiffCSE: Difference-based contrastive learning for sentence embeddings. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4207–4218, Seattle, United States. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13042–13054.

Kawin Ethayarajh. 2019. How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65.

Hongchao Fang and Pengtao Xie. 2020. Cert: Contrastive self-supervised learning for language understanding. *arXiv preprint arXiv:2005.12766*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547.

Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Representation degeneration problem in training natural language generation models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

John Giorgi, Osvald Nitski, Bo Wang, and Gary Bader. 2021. DeCLUTR: Deep contrastive learning for unsupervised textual representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 879–895.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225, Dublin, Ireland. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcode{bert}: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 9726–9735.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. CoSQA: 20,000+ web queries for code search and question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5690–5700, Online. Association for Computational Linguistics.

Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu K Lahiri, Madanlal Musuvathi, and Jianfeng Gao. 2022. Fault-aware neural code rankers. In *Advances in Neural Information Processing Systems*.

Taeuk Kim, Kang Min Yoo, and Sang-goo Lee. 2021. Self-guided contrastive learning for BERT sentence representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2528–2540.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *ACL*.

Seanie Lee, Dong Bok Lee, and Sung Ju Hwang. 2021. Contrastive learning with adversarial perturbations for conditional text generation. In *International Conference on Learning Representations*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9119–9130.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223.

Yu Meng, Chenyan Xiong, Payal Bajaj, Saurabh Tiwary, Paul Bennett, Jiawei Han, and Xia Song. 2021. Coco-lm: Correcting and contrasting text sequences for language model pretraining. *arXiv preprint arXiv:2102.08473*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *arXiv preprint arXiv:2203.13474*.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. MAUVE: Measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems*.

Ruchir Puri, David Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian T Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3505–3506.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.

Krishna Srinivasan, Karthik Raman, Jiecao Chen, Michael Bendersky, and Marc Najork. 2021. Wit: Wikipedia-based image text dataset for multimodal multilingual machine learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2443–2449.

Jianlin Su, Jiarun Cao, Weijie Liu, and Yangyiwen Ou. 2021. Whitening sentence representations for better semantics and faster retrieval. *arXiv preprint arXiv:2103.15316*.

Yixuan Su and Nigel Collier. 2022. Contrastive search is what you need for neural text generation. *arXiv preprint arXiv:2210.14140*.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022. A contrastive framework for neural text generation. In *Advances in Neural Information Processing Systems*.

Feng Wang and Huaping Liu. 2021. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2495–2504.

Lingxiao Wang, Jing Huang, Kevin Huang, Ziniu Hu, Guangtao Wang, and Quanquan Gu. 2020. Improving neural language generation with spectrum control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9929–9939.

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. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708.

Xing Wu, Chaochen Gao, Liangjun Zang, Jizhong Han, Zhongyuan Wang, and Songlin Hu. 2022. ESimCSE: Enhanced sample building method for contrastive learning of unsupervised sentence embedding. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3898–3907, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. Clear: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466*.

Yuanmeng Yan, Rumei Li, Sirui Wang, Fuzheng Zhang, Wei Wu, and Weiran Xu. 2021. ConSERT: A contrastive framework for self-supervised sentence representation transfer. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5065–5075.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Dejiao Zhang, Feng Nan, Xiaokai Wei, Shang-Wen Li, Henghui Zhu, Kathleen McKeown, Ramesh Nallapati, Andrew O. Arnold, and Bing Xiang. 2021. Supporting clustering with contrastive learning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5419–5430.

Dejiao Zhang, Wei Xiao, Henghui Zhu, Xiaofei Ma, and Andrew Arnold. 2022. Virtual augmentation supported contrastive learning of sentence representations. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 864–876, Dublin, Ireland. Association for Computational Linguistics.

Supplementary Material: Appendices

A Contrastive Learning for CLM

We detail our proposed token-level and sequence-level contrastive losses. Before that, we first call out the following notations that will be used throughout this section. Let $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{|\mathbf{x}|}]$ denote a sequence with variable length $|\mathbf{x}|$, *e.g.,* a text document or a code snippet, and $\mathbf{h} = [\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_{|\mathbf{x}|}]$ be its representation output by the last hidden layer of the decoder. For a randomly sampled batch $\mathcal{B} = \{\mathbf{x}^j\}_{j=1}^N$ with N sequences, we use $\mathbf{x}_i^j$ and $\mathbf{h}_i^j$ to denote the i^{th} token and its representations in the j^{th} sequence, respectively. Let $\mathbf{h}^j, \mathbf{h}^{j+}$ denote the representation pair of sequence $\mathbf{x}^j$ and $\mathbf{h}_i^j, \mathbf{h}_i^{j+}$ correspond to the representations of the i-th token. Such representation pairs are referred to as positive pairs in contrastive learning, which are often obtained via data augmentation.

A.1 Token-Level Contrastive Learning

As aforementioned, $\mathbf{h}_i^j, \mathbf{h}_i^{j+}$ are a pair of representations for $\mathbf{x}_i^j$, the i-th token in the j-th sequence. Let $\mathcal{I}_j = \{1, 2, \ldots, |\mathbf{x}_j|\}$ denote the indices of tokens in $\mathbf{x}_j$. Further let τ denote the temperature hyper-parameter and $\diamond$ denotes the cosine similarity, *i.e.,* $\mathbf{a} \diamond \mathbf{b} = \mathbf{a}^T \mathbf{b} / \|\mathbf{a}\|_2 \|\mathbf{b}\|_2$. Then we minimize $\mathcal{L}_{\text{Tok}}$ defined in Table 5.

A.2 Sequence-Level Contrastive Learning

Let $\mathcal{I}_B = \{1, 2, \ldots, N\} \cup \{1^+, 2^+, \ldots, N^+\}$ denote indices of all $2N$ sequence-level representations for batch $\mathcal{B}$. The sequence-level contrastive loss is defined as $\mathcal{L}_{\text{Seq}}$ in Table 5.

B Training Details

Training Data For text, we use WikiText-103, a collection of over 100 million tokens extracted from the set of verified and featured articles on Wikipedia (Merity et al., 2017). For code, we collect permissively licensed Python code from GitHub. Following (Chen et al., 2021; Nijkamp et al., 2022), we perform filtering and deduplication and further remove data that contains a significant use of non-English languages or is not parsable, resulting in a dataset of 101GB code.

Model We use GPT-2 (Radford et al., 2019) and CodeGen 350M monolingual (Nijkamp et al., 2022) for all experiments on natural language (text) and programming language (code), respectively. We set the batch size to 512 and continue to train GPT-2 on WikiText-103 and CodeGen on the GitHub data for 12 and 2 epochs, respectively. We train both models using a max sequence length of 512 tokens and 1024 for WikiText-103 and Code data, respectively. We set the learning rate to 2e-5, warm-up steps as 500 with linear annealing after peak learning rate, weight decay of 0.1, temperature of 0.05 (when using contrastive losses), and gradient clipping of 1.0. We use AdamW optimizer (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ following (Nijkamp et al., 2022). Our training pipeline is based on PyTorch Lightning[8], and we use DeepSpeed (Rasley et al., 2020) for training optimization.

Processing Code Training Data Our pre-processing strategy for code datasets used for training is designed to ensure that we optimize for data utilization while retaining the syntactic structure of programming language sequences. We also eliminate duplicate sequences since this benefits training large language models (Lee et al., 2022). Specifically, we break long sequences into chunked sequences of smaller lengths to retain most parts of the original program. Further, we maintain syntactic structure in the chunks by ensuring that each chunk ends with a '\n' character. Each chunk obtained this way contains at most `max_chars_per_seq` characters where `max_chars_per_seq` = `max_tokens_per_seq * chars_per_tok`. In our experiments, we fix `chars_per_tok` $=$ 3.2 and `max_tokens_per_seq` $= 1024$. We also perform deduplication using character-based exact matches between chunked sequences over the entire dataset. This step helps eliminate exact duplicates that might be present after the chunking stage.

C More on Evaluation

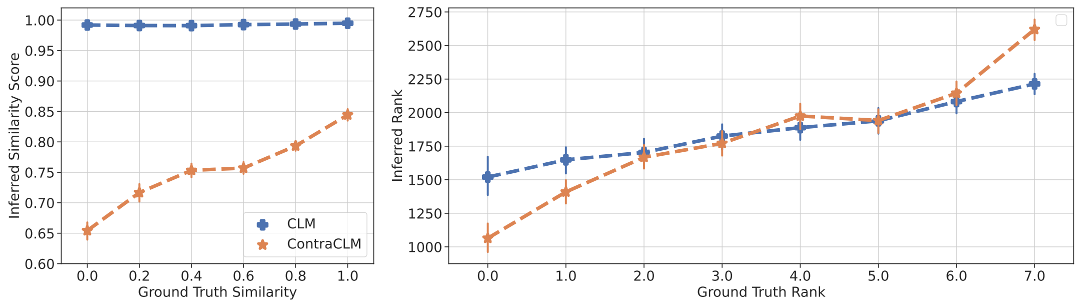
C.1 Representation Quality Evaluated on STS

For each sequence pair in STS, a fine-grained similarity score ranging from 0 to 5 is provided, with a high similarity score indicating semantically similar pairs and low similarity scores suggesting semantically dissimilar or irrelevant pairs. For better illustration, we scale the human-annotated similar-

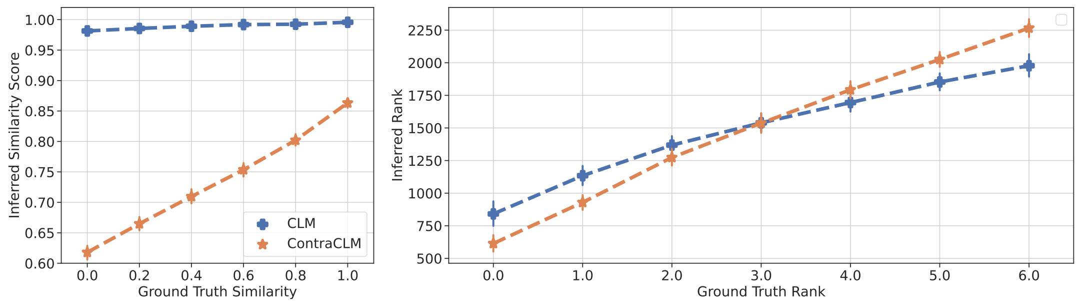
[8]https://www.pytorchlightning.ai/

Contrastive Loss	Expression
$\mathcal{L}_{\text{Tok}}$	$\sum_{j=1}^{N}\sum_{i=1}^{\lvert\mathbf{x}^j\rvert} -\left(\log \frac{\exp(\mathbf{h}_i^j \diamond \mathbf{h}_i^{j^+}/\tau)}{\exp(\mathbf{h}_i^j \diamond \mathbf{h}_i^{j^+}/\tau) + \sum_{t\in\mathcal{I}_j\setminus i}\left[\exp(\mathbf{h}_i^j \diamond \mathbf{h}_t^j/\tau) + \exp(\mathbf{h}_i^j \diamond \mathbf{h}_t^{j^+}/\tau)\right]} + \log \frac{\exp(\mathbf{h}_i^{j^+} \diamond \mathbf{h}_i^j/\tau)}{\exp(\mathbf{h}_i^{j^+} \diamond \mathbf{h}_i^j/\tau) + \sum_{t\in\mathcal{I}_j\setminus i}\left[\exp(\mathbf{h}_i^{j^+} \diamond \mathbf{h}_t^j/\tau) + \exp(\mathbf{h}_i^{j^+} \diamond \mathbf{h}_t^{j^+}/\tau)\right]}\right)$
$\mathcal{L}_{\text{Seq}}$	$\sum_{j=1}^{N} -\left(\log \frac{\exp(\mathbf{h}^j \diamond \mathbf{h}^{j^+}/\tau)}{\exp(\mathbf{h}^j \diamond \mathbf{h}^{j+}/\tau) + \sum_{k\in\mathcal{I}_\mathcal{B}\setminus j,j^+}\exp(\mathbf{h}^j \diamond \mathbf{h}^k/\tau)} + \log \frac{\exp(\mathbf{h}^{j^+} \diamond \mathbf{h}^j/\tau)}{\exp(\mathbf{h}^{j+} \diamond \mathbf{h}^j/\tau) + \sum_{k\in\mathcal{I}_\mathcal{B}\setminus j,j^+}\exp(\mathbf{h}^{j+} \diamond \mathbf{h}^k/\tau)}\right)$

Table 5: Formulation of our token-level and sequence-level contrastive losses denoted as $\mathcal{L}_{\text{Tok}}$ and $\mathcal{L}_{\text{Seq}}$ respectively.



(a) STS14: **(Left)** Predicted cosine similarity vs. human annotated ground truth. **Right** Similarity ranking according to the model predicted similarity scores vs Human similarity based ranking.



(b) STS15: **(Left)** Predicted cosine similarity vs. human annotated ground truth. **Right** Similarity ranking according to the model predicted similarity scores vs Human similarity-based ranking.

Figure 4: CLM versus Contrastive Learning in Similarity Prediction and Ranking. We report the results on two STS benchmarks: (1) *STS14* where CLM performs the worst when compared to its own performance on the other STS tasks; and *STS15* where CLM attains the best performance when compared with its own performance on other STS tasks. For the purposes of illustration, we scale the human-annotated similarity scores from $[0, 5]$ to $[0, 1]$. A good CLM is expected to predict discriminative similarity scores such that the resulting ranking results are as close to the ranks provided by humans as possible.

ity scores to $[0, 1]$ to align with the model-predicted cosine similarity scores. This does not affect the evaluation as the spearman correlation reported in Section 4.2 is a rank-based correlation metric.

CLM yields poorly discriminative representations We report the model predicted similarity scores of sequence pairs in the left column in Figure 4. A good model is expected to yield representations that attain higher similarity scores between similar sequence pairs and lower similarity values for dissimilar sequences. Thereby, a large gap between the predicted similarity scores of similar and dissimilar pairs is desired. However, as seen in Figure 4 (left), the similarity scores attained by the model trained with the standard CLM only objective are almost at one for both similar and dissimilar sequence pairs. This suggests that the representations yielded by CLM can[9] be squeezed into a tiny cone in the representation space rather than being scattered apart to leverage the vector space's capacity better. Despite the resulting similarity ranks not being entirely flattened, as shown in the right column in Figure 4b, CLM struggles in ranking similar sequences lower and dissimilar sequences higher as a consequence of the poor discriminative representations.

In contrast, Figure 4 (left) further validates that contrastive learning effectively yields more discriminative representations with a comparatively larger similarity gap between similar pairs and dissimilar pairs. Thereby, the similarity ranking results of the sequence pairs are more aligned with those obtained according to similarity scores provided by humans, as shown in Figure 4 (right).

C.2 Programming Language Evaluation

C.2.1 Evaluation Metrics

Mean Average Precision (MAP) For a set of queries, it indicates the mean of the average precision scores for each query.

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

where Q is the number of queries.

Pass@k Given a problem (code prompt as shown in Figure 6), pass@k indicates the functional correctness of model-generated code samples. A problem is considered solved if any sample passes the unit tests. Following (Chen et al., 2021), we generate $n \geq k$ samples per problem (in this paper, we use $n = 10$ and $k \in \{1, 5\}$), count the number of correct samples $c \leq n$ that pass unit tests, and calculate the unbiased estimator of pass@k as:

$$\text{pass@}k := \mathop{\mathbb{E}}_{Problems}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right].$$

Ranked Pass@k Unlike Pass@k, where we randomly chose k out of n samples, in ranked pass@k, we chose the top-k samples based on model-provided scores and then computed pass@k.

C.2.2 Examples and Statistics

In Figure 5, we present an example of a query code fragment in Python and relevant code fragments in Python and Java, respectively. While in-language code-to-code search refers to retrieving relevant code fragments in the same language, cross-language code-to-code search refers to retrieving code fragments in a different language. We present the statistics of the code search dataset in Table 6. To demonstrate the code completion task, we illustrate an example in Figure 6.

C.2.3 Detailed Code Search Results

We provide a comparison between encoder-only (Feng et al., 2020; Guo et al., 2021), encoder-decoder (Ahmad et al., 2021; Wang et al., 2021), and decoder-only models (main focus of this work) on the zero-shot code-to-code search task in Table 7b. We see that CONTRACLM-TOK and CONTRACLM outperform the encoder-only model CodeBERT and both the encoder-decoder models. It is important to note that the comparison across these models is not apple-to-apple as these models differ in size, the scale of pretraining, and language settings. This comparison's purpose is to show the promise of decoder-only models being used in discriminative tasks like code search.

We further break down the code search performances based on edit similarities and length differences between query code and their relevant code fragments. We present the results in Figure 7 and 8. We observe a similar performance trend in all three languages, although cross-lingual search performance still needs to improve. Nonetheless, the objective of this performance analysis is to show that sequence overlap or length are not the reasons for improvements in CONTRACLM-TOK. Instead, a finer-grained understanding of code tokens due to the token-level contrastive learning makes CONTRACLM-TOK more effective.

D More Analysis and Discussions

D.1 Bridge the Gap on Discriminative Tasks

Compared to the causal (left-to-right) attention mechanism of the decoder-only models, the bidirectional attention mechanism in both encoder-only and encoder-decoder models allows for better leverage of the context of the sequence and hence leads to better representations.

Taking the encoder-only models in Table 7a for illustration, on average, BERT-Base (Devlin et al., 2019) and Roberta-Base (Liu et al., 2019) outperform GPT-2 with 67.25% (absolute 21.17%) and 84.62% (absolute 26.64%) relative improvement on STS, respectively. Although the performance gap

[9]As investigated in Figure 1, the decoder-only models pretrained with the CLM-only can suffer from the anisotropy issue, which depends on the model size and domain.

	Ruby	Python	Java
Total problems	1,708	2072	3142
Total #solution	11,744	15,594	23,530
Avg. #solution / problem	6.9	7.5	7.5
Avg. length / solution	160.4	214.4	894.9
Stdev. of length / solution (problem-wise)	112.80	113.79	813.0
Solutions with length > 512	409	1,200	10,023
Solutions with length > 1024	78	278	4,766
Avg. edit similarity	$0.48_{(+0.13)}$	$0.52_{(+0.13)}$	$0.49_{(+0.13)}$

Table 6: Statistics of code-to-code search task dataset created from CodeNet (Puri et al., 2021). We truncate the code if its length exceeds the maximum sequence length, which is set to 512.

between CodeGen and the BERT models trained on programming languages, *i.e.,* CodeBERT (Feng et al., 2020) and GraphCodeBERT (Guo et al., 2021), decreases or even diminishes when evaluated on the code search tasks, the performance gap is still significant as both the model size and pretraining data in CodeGen are much larger than those used by the encoder-only models in Table 7b. Similar trends were observed in the performance gap between the decoder-only and encoder-decoder models on both natural language (Lewis et al., 2020; Raffel et al., 2020) and programming language (Ahmad et al., 2021; Wang et al., 2021).

The large performance gap severely limits the decoder-only models used in many discriminative tasks. To this end, contrastive learning shows the promise to largely bridge the gap. As seen in Table 7a, on STS, CONTRACLM reduces the relative performance gap from 67.24% (absolute 21.12%) to 16.17% (absolute 7.33%) regarding BERT-Base, and from 84.62% (absolute 26.64%) to 28.24% (absolute 12.8%). Similarly, Table 7b shows that CONTRACLM outperforms encoder-decoder models and performs comparably to the encoder-only model, GraphCodeBERT.

D.2 Dropout for Contrastive Learning

Gao et al. (2021) showed that the dropout-based augmentation is an effective strategy for unsupervised contrastive learning, and the follow-up works (Chuang et al., 2022; Wu et al., 2022) endorse the effectiveness. This motivates us to study dropout-based augmentation in our proposed contrastive learning framework. We present the results on discriminative and generation tasks in Tables 8 and 9, respectively. From the results, it is evident that the adoption of dropout-based augmentation improves the discrimination task performances, which corroborates the findings of (Gao et al., 2021). In contrast, dropout-based augmentation hurts the generation task performances. On the other hand, for code completion, we had anticipated that dropout-based augmentation would hurt performance since we used the CodeGen model (Nijkamp et al., 2022) which does not use dropout activations during its initial pretraining stage. However, we observe a drop in perplexity due to disabling dropout for both CLM and CONTRACLM in Table 9, which does not go with our anticipation, especially considering that, unlike CodeGen, GPT-2 is pretrained with dropout enabled. We leave diving deeper into the reasoning behind this finding as future work.

D.3 CONTRACLM outperforms SimCTG

To better understand the performance gap between CONTRACLM and SimCTG (Su et al., 2022), we run the following ablations on GPT-2 and report the evaluations on STS. In Table 10, we report the results of (1) running CONTRACLM w/o dropout-based data augmentation and compare it with the original SimCTG model and (2) augmenting SimCTG with both the sequence-level contrastive loss and dropout-based augmentation and compare it with our proposed CONTRACLM model. As we can see, CONTRACLM consistently outperforms SimCTG in both settings. Figure 10 together with our results reported in Section 4.3, where we disabled the dropout-based augmentation for CONTRACLM and its variations but still observed consistently better performance than SimCTG on both discrimination and generation tasks, conclude that CONTRACLM is better than SimCTG across domains and settings.

Model	STS12	STS13	STS14	STS15	STS16	SICK-R	STS-B	Avg.
Encoder-only Models								
BERT-Base	30.92	59.96	47.72	60.35	63.72	58.25	47.36	52.65
RoBERTa-Base	53.95	47.42	55.87	64.73	63.55	62.94	58.40	**58.12**
Encoder-Decoder Models								
BART-Base	34.46	52.49	44.50	62.51	61.99	57.72	52.30	52.28
T5-Base	37.78	56.81	49.37	65.50	64.65	60.11	57.52	**55.96**
Decoder-only Models								
GPT2	25.84	28.90	26.20	34.74	35.70	42.72	26.27	31.48
CLM	27.14	20.34	18.73	37.56	27.40	35.70	27.97	27.83
SimCTG	30.32	37.10	31.99	39.68	42.73	46.26	25.27	36.19
CONTRACLM-TOK	37.28	37.63	31.33	54.78	50.16	48.10	34.95	42.03
CONTRACLM-SEQ	29.66	39.89	34.50	43.20	41.99	44.52	25.51	37.04
CONTRACLM	37.54	45.23	36.41	56.74	50.30	51.52	39.49	**45.32**

(a) Spearman rank correlation between the cosine similarity of sentence pairs and the human-annotated similarity scores.

Model	Ruby			Python			Java			Avg.
	Ruby	Python	Java	Ruby	Python	Java	Ruby	Python	Java	
Encoder-only Models										
CodeBERT	13.55	3.18	0.71	3.12	14.39	0.96	0.55	0.42	7.62	4.94
GraphCodeBERT	17.01	9.29	6.38	5.01	19.34	6.92	1.77	3.50	13.31	**9.17**
Encoder-Decoder Models										
PLBART	18.60	10.76	1.90	8.27	19.55	1.98	1.47	1.27	10.41	**8.25**
CodeT5-base	18.22	10.02	1.81	8.74	17.83	1.58	1.13	0.81	10.18	7.81
Decoder-only Models										
CodeGen	16.18	5.90	0.52	2.66	18.11	0.36	1.61	1.65	10.16	6.35
CLM	16.36	6.67	0.80	3.07	15.72	0.46	1.41	2.11	10.25	6.32
SimCTG	17.66	7.19	1.94	7.63	18.31	1.78	1.63	2.32	10.83	7.70
CONTRACLM-TOK	18.02	7.84	2.51	8.76	20.46	2.48	1.91	2.58	11.43	**8.44**
CONTRACLM-SEQ	16.76	5.45	1.06	7.40	16.74	1.41	1.55	2.25	10.23	6.98
CONTRACLM	17.90	7.78	2.56	9.05	19.74	2.64	1.90	2.50	11.32	8.38

(b) MAP score (%) of the zero-shot code-to-code search task. The language names mentioned in the top two rows indicate the languages queries and candidates are written in.

Table 7: CONTRACLM bridges the gap between CLM and Encoder-Only / Encoder-Decoder models.

Model	Ruby			Python			Java			Avg.
	Ruby	Python	Java	Ruby	Python	Java	Ruby	Python	Java	
CLM $_{+\text{Dropout}}$	18.04	6.47	1.21	5.52	18.70	1.18	1.62	2.35	11.26	7.37
CLM $_{-\text{Dropout}}$	16.36	6.67	0.8	3.07	15.72	0.46	1.41	2.11	10.25	6.32
ContraCLM $_{+\text{Dropout}}$	**20.09**	**8.84**	**3.66**	**9.25**	**22.39**	**3.13**	**1.93**	**3.06**	**12.02**	**9.37**
	17.90	7.78	2.56	9.05	19.74	2.64	1.90	2.50	11.32	8.38

(a) MAP score (%) of zero-shot code-to-code search.

Model	STS12	STS13	STS14	STS15	STS16	SICK-R	STS-B	Avg.
CLM $_{+\text{Dropout}}$	27.14	20.34	18.73	37.56	27.40	35.70	27.97	27.83
CLM $_{-\text{Dropout}}$	25.60	15.23	13.95	31.64	28.13	34.96	26.15	25.09
ContraCLM $_{+\text{Dropout}}$	37.54	**45.23**	**36.41**	**56.74**	**50.30**	**51.52**	**39.49**	**45.32**
ContraCLM $_{-\text{Dropout}}$	**38.22**	40.15	33.57	53.16	45.35	47.47	36.10	42.00

(b) Spearman rank correlations between the cosine similarity of sentence representation pairs and the ground truth similarity scores for STS benchmarks.

Table 8: Discriminative task performances with ($_{+\text{Dropout}}$) and without ($_{-\text{Dropout}}$) Dropout augmentation applied to CLM and ContraCLM. We apply Dropout (0.1) to all the layers of the models.

Model	Pass@k		Ranked Pass@k	
	k=1	k=5	k=1	k=5
CLM $_{+\text{Dropout}}$	12.65	15.54	13.42 $_{(+0.77)}$	16.46 $_{(+0.92)}$
CLM $_{-\text{Dropout}}$	13.42	18.08	15.38 $_{(+1.96)}$	18.29 $_{(+0.21)}$
ContraCLM $_{+\text{Dropout}}$	13.19	15.92	13.41 $_{(+0.22)}$	16.46 $_{(+3.05)}$
ContraCLM $_{-\text{Dropout}}$	**14.63**	**18.83**	**17.07** $_{(+2.44)}$	**18.90** $_{(+0.07)}$

(a) Evaluation results on the HumanEval benchmark. The numbers in the subscript indicate the difference between ranked pass@k and pass@k accuracy.

	CLM $_{-\text{Dropout}}$	CLM $_{+\text{Dropout}}$	ContraCLM $_{-\text{Dropout}}$	ContraCLM $_{+\text{Dropout}}$
Perplexity	**21.86**	22.48	22.07	23.01

(b) Perplexity of continually trained GPT-2 on the test set of WikiText-103.

Table 9: Generation task performances with ($_{+\text{Dropout}}$) and without ($_{-\text{Dropout}}$) Dropout augmentation applied to CLM and ContraCLM. We apply Dropout (0.1) to all the layers of the models.

Model	STS12	STS13	STS14	STS15	STS16	SICK-R	STS-B	Avg.
SimCTG	30.32	37.10	31.99	39.68	42.73	46.26	25.27	36.19
ContraCLM $_{-\text{Dropout}}$	38.22	40.15	33.57	53.16	45.35	47.47	36.10	42.00
SimCTG$_{+\mathcal{L}_{\text{Seq}}+\text{Dropout}}$	**38.70**	43.60	36.29	50.01	45.19	48.25	33.36	42.20
ContraCLM $_{+\text{Dropout}}$	37.54	**45.23**	**36.41**	**56.74**	**50.30**	**51.52**	**39.49**	**45.32**

Table 10: ContraCLM outperform SimCTG (Su et al., 2022) even without dropout-based data augmentation (first two rows); or augmenting SimCTG with dropout and sequence-level contrastive loss defined in Table 5.

Query Program in Python

```python
import math
h,w=map(int, input().split())
if h%3==0 or w%3==0:
    print(0)
else:
    x,y=max(h,w),min(h,w)
    ans=y
    for hi in range(1,h):
        M=max(hi*w,(h-hi)*((w+1)//2),(h-hi)*(w//2))
        m=min(hi*w,(h-hi)*((w+1)//2),(h-hi)*(w//2))
        ans=min(ans,M-m)
    for wi in range(1,w):
        M=max(wi*h,(w-wi)*((h+1)//2),(w-wi)*(h//2))
        m=min(wi*h,(w-wi)*((h+1)//2),(w-wi)*(h//2))
        ans=min(ans,M-m)
    print(ans)
```

Relevant Program in Python

```python
def solve(H,W):
    p1 = [H//3*W, (H-H//3)*(W//2), (H-H//3)*(W-W//2)]
    p2 = [ceil(H/3)*W, (H-ceil(H/3))*(W//2), (H-ceil(H/3))*(W-W//2)]
    S1 = max(p1)-min(p1)
    S2 = max(p2)-min(p2)
    S3 = 0 if H%3==0 else W
    return min(S1,S2,S3)

from math import ceil
H, W = map(int, input().split())
print(min(solve(H,W), solve(W,H)))
```

Relevant Program in Java

```java
import java.util.*;
public class Main {
  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    long w = sc.nextInt();
    long h = sc.nextInt();
    if(w%3==0 || h%3==0)
      System.out.println(0);
    else
      System.out.println(Math.min(solve(w, h), solve(h, w)));
  }
  static long solve(long w, long h){
    long min = Long.MAX_VALUE;
    for(int i=1;i<h;i++){
      long a = w*i;
      long b, c = 0, 0;
      if(w%2==0){
        b = w/2*(h-i);
        c = b;
        min = Math.min(min, Math.max(a, Math.max(b, c))-Math.min(a, Math.min(b, c)));
      }
      else if((h-i)%2==0){
        b = w*((h-i)/2);
        c = b;
        min = Math.min(min, Math.max(a, Math.max(b, c))-Math.min(a, Math.min(b, c)));
      }
      else{
        b = w*((h-i)/2);
        c = w*((h-i)/2+1);
        min = Math.min(min, Math.max(a, Math.max(b, c))-Math.min(a, Math.min(b, c)));
        b = w/2*(h-i);
        c = (w/2+1)*(h-i);
        min = Math.min(min, Math.max(a, Math.max(b, c))-Math.min(a, Math.min(b, c)));
      }
    }
    return min;
  }
}
```

Figure 5: An example of a query and relevant documents in code-to-code search task, where both query and candidates are complete programs that solve a programming problem. In this search task, each program is used as a query and retrieve all programs (from a collection of 11,744/15,594/23,530 programs in Ruby, Python, and Java, respectively) that solve the same problem.

Prompt (function signature and docstring)

```
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
    """
```

Unit tests

```
def check(candidate):
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
    assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
    assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
    assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
    assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True
    assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
    assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False

check(has_close_elements)
```

Completion 1 (passed; mean_logp: -0.1146)

```
    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
```

Completion 2 (failed; mean_logp: -0.1211)

```
    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
```

Completion 3 (failed; mean_logp: -0.1598)

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

Completion 4 (failed; mean_logp: -0.1607)

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

Completion 5 (failed; mean_logp: -0.3904)

```
    return abs(sum(numbers) - sum(numbers) / len(numbers)) < threshold
```

Figure 6: An example of prompt from HumanEval and five unique code completions generated by the CONTRACLM model. We rank them based on the mean_logp scores. Considering these 5 completions and one of the passes unit tests, pass@1 is 0.2 while ranked pass@1 is 1.0.

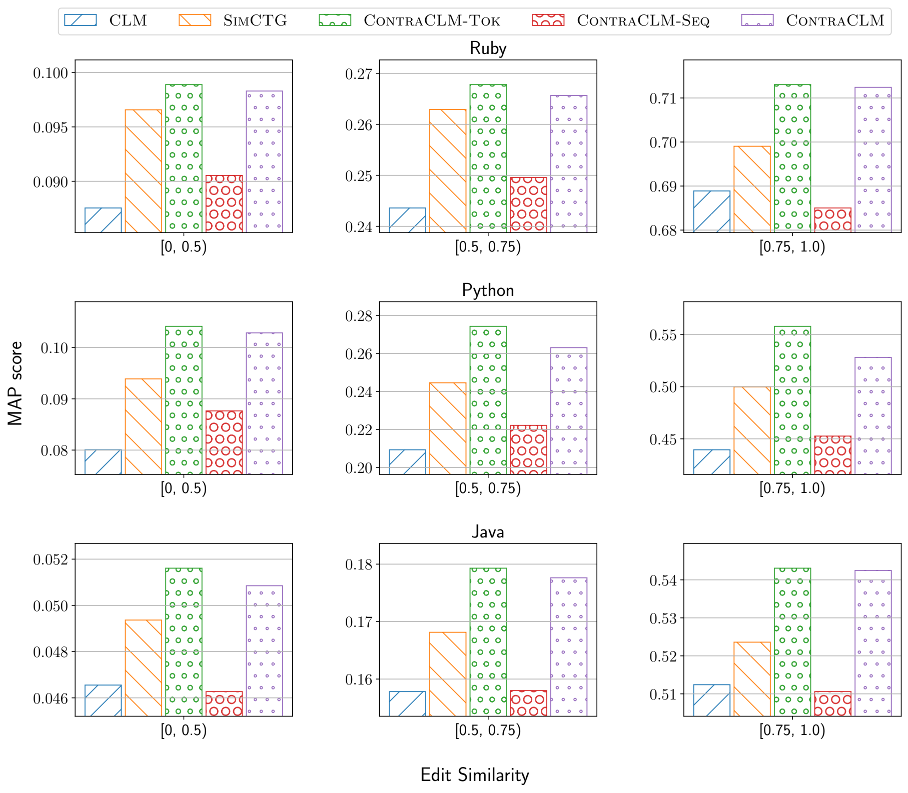


Figure 7: Code-to-code search performance (MAP score) breakdown for different models based on the edit similarities ([0, 1]) between the query code fragments and the relevant code fragments. Higher and lower edit similarity indicates the search task is trivial or difficult, respectively.

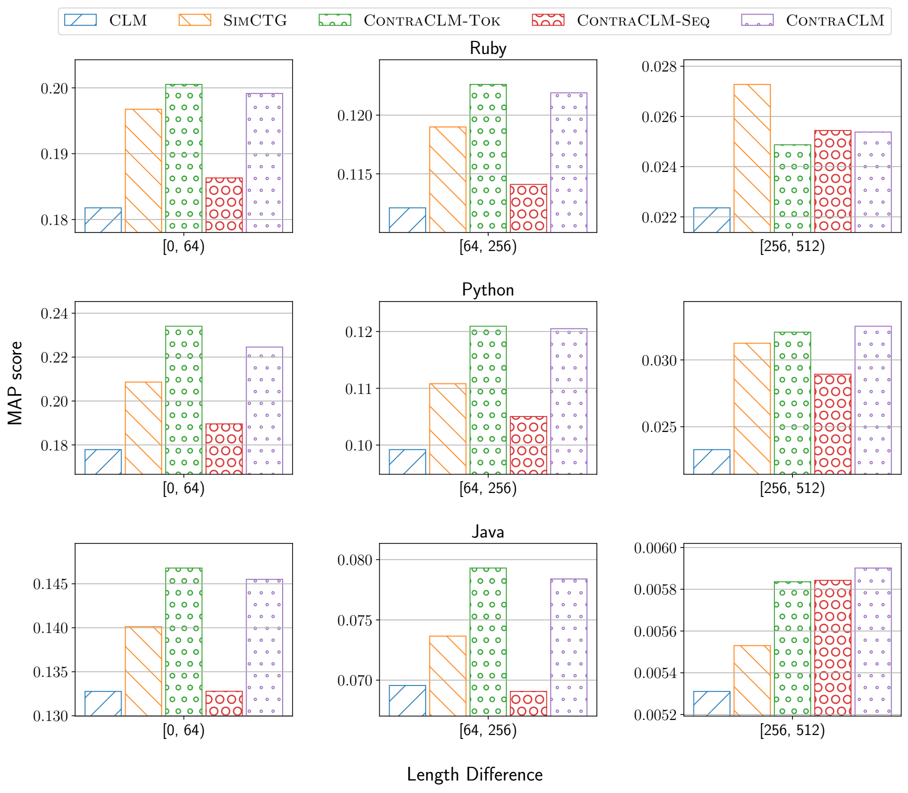


Figure 8: Code-to-code search performance (MAP score) breakdown based on (absolute) length differences between the query code fragments and their relevant code fragments.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations section after the conclusion section.

☑ A2. Did you discuss any potential risks of your work?
Ethics statement after the limitation section.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and introduction

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 4: Experiments

☑ B1. Did you cite the creators of artifacts you used?
Section 4: Experiments

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Ethics statement

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Ethics statement

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Justification provided in the ethics statement

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Section 4: Experiments

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Appendix

C ☑ Did you run computational experiments?

Section 4: Experiments

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 4: Experiments, Ethics statement

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Ethics statement

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 4: Experiments

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?
Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.