Exploring Distributional Shifts in Large Language Models for Code Analysis

Anonymous ACL submission

Abstract

We systematically study the capacity of two large language models for code - CodeT5 and Codex - to generalize to out-of-domain data. In this study, we consider two fundamental applications - code summarization, and code generation. We split data into domains following its natural boundaries - by an organization, by a project, and by a module within the software project. This makes recognition of in-domain vs out-of-domain data at the time of deployment trivial. We establish that samples from each new domain present both models with a significant challenge of distribution shift. We study how well different established methods can adapt models to better generalize to new domains. Our experiments show that while multitask learning alone is a reasonable baseline, combining it with few-shot finetuning on examples retrieved from training data can achieve very strong performance. In fact, according to our experiments, this solution can outperform direct finetuning for very low-data scenarios. Finally, we consider variations of this approach to create a more broadly applicable method to adapt to multiple domains at once. We find that in the case of code generation, a model adapted to multiple domains simultaneously performs on par with those adapted to a single domain.

1 Introduction

Since the late 2000s, researchers have been reporting poor generalization of statistical learning models to new software systems (Turhan, 2012; Zimmermann et al., 2009), a phenomenon that has become increasingly important with the rise of large language models (LLMs) for code, such as GitHub Copilot, Amazon CodeWhisperer, Replit, etc. Thus, it is crucial to understand when pretrained large language model performance on a private software system will differ from the performance obtained on a benchmark. Prior work has studied some aspects of this problem, among others studying generalization from older to newer code, large software projects, and small competition problems, authors, and code representations (Nie et al., 2022; Li et al., 2021; Hu et al., 2022).

However, the challenges of distribution shifts stemming from the hierarchical nature of software data, as depicted in Figure 1, have not been systematically studied with regard to large language models for code. Motivated by that, in this work we probe the generalization capacity of large language models for code, specifically Codex (Chen et al., 2021) and CodeT5 (Wang et al., 2021), in code generation and summarization tasks, examining three scenarios: generalization across companies, projects, and project components. These scenarios are routinely considered for analyzing software systems (Ma et al., 2012; Li et al., 2009; Mair et al., 2000) due to the careful consideration that goes into combining or separating such entities.

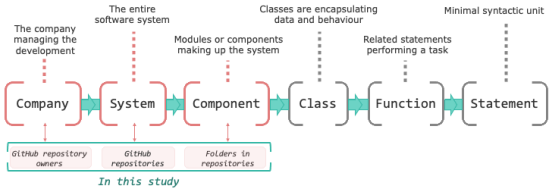


Figure 1: Organization of a software system by the granularity of its components

First, we want to understand *how models perform on new domains* - if models struggle with out-of-domain generalization, they should be used with caution. At the same time, we empirically establish the legitimacy of our definitions for out-of-domain scenarios by demonstrating that these examples present a distributional shift. To answer this question we compare the performance of the models without any additional adaptation, with that of the models that have been adapted on limited data from a random domain, or from the test domain. Adaptation with labeled examples from the test domain is the proxy for model performance if there were no distributional shift. We find that both models suffer from a drop in performance when

applied out-of-domain. In this experiment, the difference is more pronounced for code summarization, where adapting models with few in-domain examples, on average, leads to an improvement of over 10 BLEU (Papineni et al., 2002) score points.

Next, we explore ways to *improve the out-of-domain generalization of large language models for code*, recognizing that relying on labeled in-domain data for every new domain is impractical. Instead, we investigate the use of labeled out-of-domain data and small amounts of *unlabelled* in-domain data to enhance generalization. We test methods known to be successful in other transfer learning scenarios, such as meta-learning (Thrun and Pratt, 1998; Vilalta and Drissi, 2002) and multitask learning (Caruana, 1996; Silver, 1996). We also leverage unlabeled in-domain data to retrieve similar labeled examples from an out-of-domain corpus for adapting to the new domain. We find that while meta-learning and multitask learning do not solve the out-of-domain generalization problem, domain adaptation with retrieved examples is a good technique for low-data domains. Models using retrieved examples perform on par, or better, than models that have been adapted using a few samples (e.g., 8 or 16) of in-domain labeled data.

Lastly, *can we make the code models more broadly applicable and retain their generalization capacities*, rather than having to adapt them to every new domain? Depending on the approach to model adaptation (e.g. weight update vs in-context demonstrations) we varied the set of retrieved examples for each new domain, or for each test input individually. We compare performance obtained this way with that of the models that are adapted simultaneously to multiple domains (or instances, correspondingly). We find that Codex is very sensitive to these changes, so it is best to retrieve similar instances for each test data point. On the other hand, CodeT5 has a minor drop in code summarization and a negligible drop in code generation. This makes it feasible to adapt and apply CodeT5 to multiple domains simultaneously with minimal trade-off, eliminating the need to store separate copies of the model for each domain.

2 Background

Distribution shifts, the shifts in underlying semantics between the training and evaluation data, can be one of the most impacting factors for deteriorating performance at test time. Prior work in code analysis has mainly focused on *cross-project* distribution shifts, training the model on one set of code projects and evaluating them on unseen code projects. Additionally, the studies were mainly conducted in the context of *traditional machine learning methods*, such as linear classifiers, support vector machines, and later, LSTMs (Zimmermann et al., 2009; Turhan, 2012; Angioni et al., 2022).

Recently, there has been a resurgence of interest in studying distribution shifts in code analysis, with newer works considering shifts caused by different authors of the code, the timeline of the project, distributions of code tokens, etc (Li et al., 2021; Hu et al., 2022; Nie et al., 2022). Additionally, large language models trained on code have demonstrated remarkable capabilities in code analysis tasks, however, their abilities under domain shift are still under-explored. In this work, we conduct a comprehensive empirical analysis to probe the large language models' capabilities in handling three different granularity of distribution shifts (company, domain, module) when different training and adaptation methods are used. In addition to directly fine-tuning vanilla LLMs, we experiment with enhancing pretrained models using the methods described below.

Meta-Learning and Multi-task Learning. In our work, we experimented with both Meta-Learning and Multi-task learning to get better initialization for few-shot performance on the downstream task. For meta-learning, we have chosen Model-agnostic Meta-Learning(MaML) (Finn et al., 2017) which is a gradient-based method. It is a conceptually simple and model-agnostic algorithm that has been shown to outperform existing approaches in several tasks. Multi-task Learning aims to learn a shared and generalized representation by jointly training on several tasks. We adopted the simplest approach to multi-task learning by jointly finetuning a shared language model on multiple tasks.

Parameter Efficient Methods. Parameter-efficient methods have been shown to obtain performance comparable to finetuning all model parameters with only a tiny fraction of model parameters. In our work, we have experimented with Low-Rank Adaptation (LoRA) (Hu et al., 2021), which is a low-rank update method.

In-context learning. GPT-3 (Brown et al., 2020) demonstrated the ability of large language models to perform few-shot predictions, where the model is given a description of the task in natural language

with few examples. In our work, we conducted experiments on in-context learning on Codex.

Retrieval Based Example Selection. It has been shown in Liu et al. (2021) that in-context examples selected following a strategy may serve as more informative input to unleash GPT3's extensive knowledge. Inspired by this, we leveraged a simple similarity-based retrieval module to augment Codex for in-context learning example selection. Also, for the few-shot training of CodeT5, we experimented with a retrieval-based stratified few-shot example selection approach.

3 Problem setting

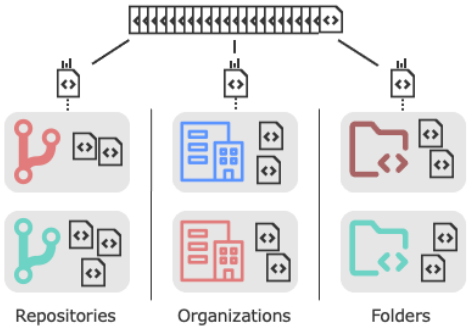


Figure 2: We divide and group the functions from CodeSearchNet by the repositories, organizations, and folders that they belong to.

We are considering the scenario where a user is looking to use a large language model, such as Codex or CodeT5, in their software project. We want to understand how these models will perform particularly considering that the code may be coming from an unseen organization, an unseen project, or a previously unseen part of the project.

Let us have two mutually exclusive sets of code data points: X_{train} and X_{test}. Assuming that the code in the data is extracted from some software projects, we can identify the *organization*, *project*, and the *module* within the project that the data point came from. Based on each of those characteristics we can group the data points into sets, and end up with three *sets of sets*, as illustrated in Figure 2. For example, the middle set in the figure contains multiple sets of data points. Each of those sets corresponds to a unique organization which all data points within it originated from. In other words, according to our prior definitions, all data points within a set belong to the same domain. For simplicity, we will refer to a set of examples from the same domain as τ_i. We also will refer to splits of such a set into train/development or test portions as τ_{train}, τ_{dev}, and τ_{test}.

	$\tau \subset X_{train}$ (total)	$\tau \subset X_{train}(\lvert\tau\rvert \geq 96)$	$\tau \subset X_{test}(\lvert\tau\rvert \geq 96)$
org.	9737	195	8
repos.	15858	147	15
fold.	25268	100	10

Table 1: CodeSearchNet dataset, split according to the domain definitions. The left column shows the set used for training. The middle column shows the number of domains of each kind from X_{train} that have at least 96 samples. The right column shows the number of domains in the X_{test} after filtering all domains with less than 96 samples.

3.1 Data

For our experimentation, we use CodeSearchNet (Husain et al., 2019) dataset[1], in particular, the partition containing JavaScript language. In our setup, the train section of the dataset corresponds to X_{train}, and development and test sections to X_{test}.

We wanted to keep all of the domains in X_{test} unseen, and for that reason, we removed any domain from X_{test} that has also appeared in X_{train}. This can happen because CodeSearchNet dataset was split into partitions by projects, so the same organizations can appear in differents splits. This way, any domain coming from X_{test} will be, by our definition, out-of-domain for the model trained on X_{train}. We further split each domain $\tau_i \subset X_{test}$ into τ_{train}, τ_{dev} and τ_{test}. The evaluation is performed on τ_{test}. τ_{train} and τ_{dev} are used to obtain a proxy for the upper-bound performance of the model if the domain τ_i was seen during training, i.e. if there was no distribution shift for τ_{test}.

Preprocessing We used the "path" field of the CodeSearchNet dataset to determine each code snippet's belonging to an organization, repository, and lowest-level folder. We use 5 different random seeds to divide a domain into τ_{train}, τ_{dev}, and τ_{test}. We aim to have at least 32 samples each in τ_{test} and τ_{dev}, and up to 32 samples for τ_{train}. Thus, from X_{test} we filtered any domain that had less than 96 samples in total. The final dataset statistics that we ended up with are presented in Table 1.

3.2 Applications and Metrics

We evaluated our method on two generation applications: code summarization and code generation. **Code summarization** aims to summarize a code snippet into a natural language description.

[1]Since the exact training data of Codex models is undisclosed, we cannot be sure that it did not include CodeSearchNet dataset. However, as seen later in the experiments, we see a performance difference for in-domain and out-of-domain experiments regardless of this.

The code snippet in CodeSearchNet dataset is a function, and the natural language description consists of the docstring of that function. The evaluation metric for this task is BLEU-4 (Papineni et al., 2002). **Code generation** performs the reverse operation - given a natural language description of code, the model is asked to generate the corresponding function. We follow prior work and use CodeBLEU (Ren et al., 2020) for evaluating generated code. We modified an existing CodeBLEU implementation by adding our own set of JavaScript keywords, the full list can be found in Appendix 7.1. However, recent research has established that CodeBLEU scores can disagree with human judgment scores (Evtikhiev et al., 2022), and motivated by these findings we additionally evaluate code generation models with chrF (Popovic, 2015) and RougeL (Lin, 2004) metrics. These metrics agree according to our results, so we report results for chrF and RougeL in Appendix 9.

3.3 Models

We have experimented with two large language models for code: (1) CodeT5 (Wang et al., 2021), which is an encoder-decoder model based on T5 (Raffel et al., 2019) and (2) Codex (Chen et al., 2021), which is a decoder only model based on GPT-3 (Brown et al., 2020). Both T5 and GPT-3 have strong zero-shot learning (Wei et al., 2022; Sanh et al., 2022) and transfer learning capabilities, and their versions for programming languages exhibit strong performance across multiple benchmarks. The two models are of different sizes - the CodeT5 is using 700M parameters T5-large architecture, and the Codex model uses GPT-3 architecture with more than 100B parameters. We have provided a more detailed discussion of these models in the Appendix, Section 7.3.

4 Analysis

In this section, we formulate the research questions that we aim to answer, and give more detailed description of the setups that we have used for analyzing and answering each question.

RQ 1 *How do code models perform on new domains?*

We test models' capacity for generalization to new domains by comparing the performance of the models that have been adapted to the new domain using few-shot instances of in-domain data (ID) vs those that only encountered out-of-domain data. For CodeT5 few-shot domain adaptation data is used to update the model weights, whereas for Codex it is included as demonstrations in the prompt to the model.

CodeT5

Next, we discuss the adaptation techniques for the CodeT5 model. For these methods, we have experimented with using a different number of supervision examples - 8, 16, or 32.

The first adaptation method we used is full model **fine-tuning** (**FT**). Information on the hyperparameters for this and all other methods is available in Appendix 8. Besides FT, we also experiment with a parameter-efficient fine-tuning method - **Low-Rank Adaptation** (**LoRA**) (Hu et al., 2021). This method adds trainable pairs of rank decomposition matrices in parallel to existing weight matrices thus enabling parameter-efficient adaptation to new domains without forgetting. We used the implementation from T-Few (Liu et al., 2022) library.

Codex

For Codex, we do not perform weight updates. Instead, very large models, such as Codex, have been shown to be capable to generalize to unseen tasks using only the instruction for the task. In the simplest case, we evaluated Codex by directly presenting it with the instruction, for example "Summarize following JavaScript code", and input (i.e. **instruction only**). It has been established that Codex can be sensitive to the wording of the instructions, so we used a number of different instruction variations for each application and averaged the results.

Besides that, larger models have been shown to be able to "learn" from demonstration examples that are provided as part of their input, even though this process does not involve any weight updates. This phenomenon is known as in-context learning (**ICL**) technique, which is what we use for domain adaptation for Codex. Due to the limit on the size of the input to the Codex model (4096 tokens), we used as many demonstrations as would fit, including up to 8 demonstrations with each test example. And since the model can also be sensitive to the order of examples, we shuffled the order of the demonstrations 5 times and averaged the results.

Finding: Both models struggle on new domains

Tables 2 and 3 demonstrate the performance obtained by CodeT5 and Codex (additional results for code generation metrics chrF and rougeL are available in Appendix Section 9). We see that both models for code struggle with OOD generalization

Code summarization	folder			repo			org		
	8-shot	16-shot	32-shot	8-shot	16-shot	32-shot	8-shot	16-shot	32-shot
CodeT5 FT ID	14.39	16.06	18.31	12.68	14.73	16.82	13.14	16.35	17.65
CodeT5 LoRA ID	16.57	19.07	20.93	15.22	17.14	21.20	15.61	18.56	20.87
CodeT5 FT random	3.58	4.30	5.02	4.35	4.70	5.79	4.53	5.47	6.27
CodeT5 LoRA random	3.69	4.37	4.92	4.70	5.56	5.92	5.27	5.53	6.26
Codex ICL ID	20.72	-	-	20.34	-	-	19.00	-	-
Codex ICL random	6.73	-	-	7.17	-	-	6.84	-	-
Codex instr. only *(0-shot)*	(1.61)	-	-	(1.55)	-	-	(1.52)	-	-

Table 2: Comparison of model performance for code summarization on in-domain (**ID**) vs out-of-domain (**random**) test data. Reported metric is BLEU (higher is better).

Code generation	folder			repo			org		
	8-shot	16-shot	32-shot	8-shot	16-shot	32-shot	8-shot	16-shot	32-shot
CodeT5 FT ID	14.67	15.22	16.13	16.15	17.42	18.62	14.54	15.34	16.43
CodeT5 LoRA ID	14.14	15.06	16.36	16.23	17.45	18.96	14.17	15.30	16.62
CodeT5 FT random	15.23	14.94	15.15	14.19	14.14	14.67	13.39	13.43	14.44
CodeT5 LoRA random	14.45	14.29	15.37	14.29	13.74	15.04	13.76	13.85	14.81
Codex ICL ID	23.87	-	-	25.73	-	-	24.64	-	-
Codex ICL random	16.82	-	-	16.82	-	-	17.47	-	-
Codex instr. only *(0-shot)*	(5.77)	-	-	(5.49)	-	-	(5.72)	-	-

Table 3: Comparison of model performance for code generation on in-domain (**ID**) vs out-of-domain (**random**) test data. Reported metric is CodeBLEU (higher is better).

as demonstrated by the performance difference for models that have encountered in-domain examples vs those that have not. For example, CodeT5 model on code summarization in most scenarios gains about 200% relative improvement after updating the model with few-shot in-domain data.

It is worth noting that while there is still difference in performance for CodeT5 model on code generation ID and OOD, the performance difference is next to negligible. We hypothesize that this can be due to the fact that code generation is a more challenging task for a large language model, and so the effect of distribution shift is less noticeable. This way, for CodeT5, which is a smaller model and is evaluated lower on OOD code generation, the gain is smaller. On the other side, Codex is evaluated higher on OOD code generation, and for it the addition of the in-domain data results in up to 50% of relative improvement.

RQ 2 *How to get better out-of-domain generalization?*

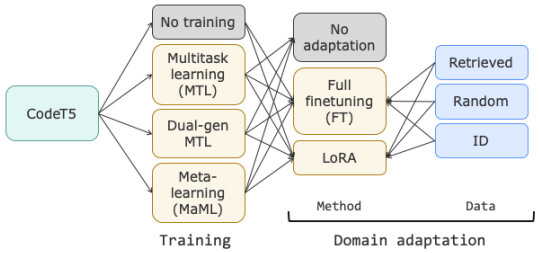
We saw that models for code performed significantly better after being adapted for new domains using in-domain data. However, there are many reasons why adapting to every new domain with the help of labeled examples might be impractical. Thus, we consider some alternative approaches, that would not require labeled data but can hopefully close the performance gap partially or fully. A high-level overview is illustrated in Figure 3.

CodeT5

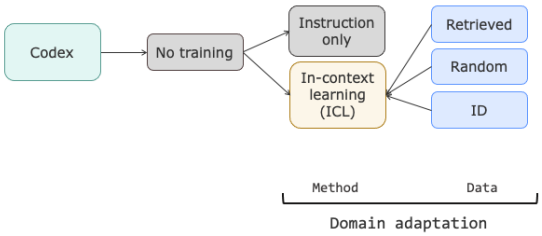
In the previous setup, we started from a pre-trained checkpoint of the model and experimented with different approaches for domain adaptation. To answer the current question, we additionally consider different methods to use before the domain adaptation stage, particularly, multi-task learning and meta-learning. The resulting experimental setups are illustrated in Figure 3a.

Multitask learning (MTL) MTL is the simple method of combining data from different domains and training the model on all the domains simultaneously. For code summarization, we used the model checkpoint provided by the authors of CodeT5, which was fine-tuned on the training portion of CodeSearchNet. For code generation, we performed our own training since the original model was not trained to generate JavaScript code.

Dual-gen MTL In the setup described above, the model is trained to perform either code generation or summarization. In addition to that, we experiment with a multitask model that has been trained



(a) CodeT5



(b) Codex

Figure 3: For the CodeT5 model we perform evaluation of different methods for training and domain adaptation techniques, as well as using different data sources during the domain adaptation stage. For Codex we perform evaluation of scenarios with different data sources during the domain adaptation stage.

on both code generation and code summarization simultaneously. We refer to this model as "dual-gen" MTL, following the authors of CodeT5. We prepended the inputs to the model with a generation or summarization instruction for each instance.

Model-Agnostic Meta Learning For model-agnostic meta-learning or MaML (Finn et al., 2017), we filtered the domains in X_{train} set, only leaving those that have at least 96 samples (see the middle column of Table 1). This was to ensure that each domain contains disjoint sets of adequate size for both training and meta-training. We used the library Higher (Grefenstette et al., 2019) for our implementation.

Stratified example retrieval for supervision In addition to the strategies above, we experiment with a domain adaptation method that does not require in-domain labeled data for supervision. We used cosine similarity on embeddings obtained from the pre-trained CodeT5 model checkpoint to **retrieve** k most similar examples for every example in τ_{test} from X_{train}. We set k to 4, 8, or 32, and since $|\tau_{test}| = 32$ the combined size of the set would be 128, 256, or 1024. Finally, we removed any duplicates. We will refer to this set as τ_{ret}.

Codex

Stratified example retrieval for demonstrations Similarly to the strategy for CodeT5, for Codex we employed in-context learning with retrieved demonstration examples. For each test query, instead of using random sets of in-domain or out-of-domain demonstrations, we used 4 or 8 of the query's most similar samples from X_{train} as demonstrations. This case will be referred to as **ICL ret**.

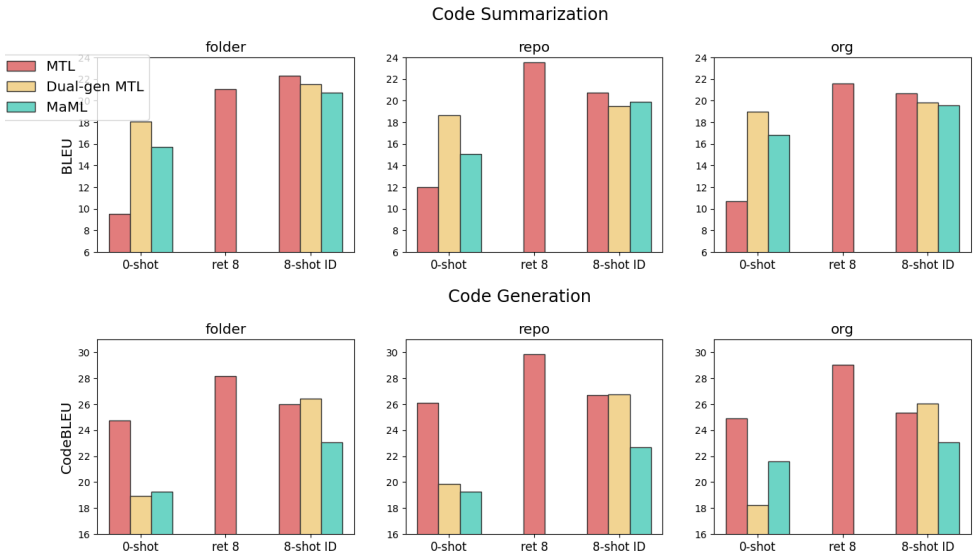
Finding: Strategic adaptation has best out-of-domain performance in low data scenarios

Figure 4a and 4b demonstrate the performance of the CodeT5 and Codex models. For CodeT5, it contains the performance obtained without adaptation (0-shot), as well as after in-domain few-shot fine-tuning (additional results for LoRA are presented in Appendix 9). None of the evaluated methods perform comparably in zero-shot setting to those with few-shot domain adaptation - whether on examples retrieved from training data or obtained from test domains. So these training methods do not result in a general-purpose model that handles out-of-domain generalization well.

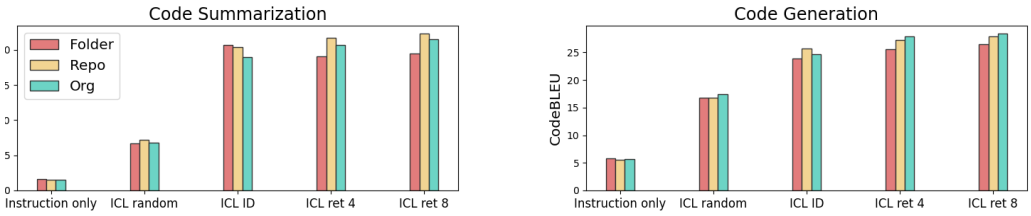
Adapting the MTL model to test domains with the help of stratified supervision provides a considerable boost to the performance of CodeT5 and Codex. Results for CodeT5 are shown in Figure 5 with bars marked "ret k", where k refers to the number of examples included in τ_{ret} per test example. For Codex, Figure 4b reports the performance using 4 or 8 retrieved demonstrations, signified as "ICL ret 4" and "ICL ret 8" respectively.

First of all, we notice that there is a saturation in terms of gained performance vs the number of stratified supervision or demonstration examples used. For CodeT5 using 32 examples per test instance is almost always worse than using 4 or 8 examples. For Codex, using 4 or 8 examples results in approximately the same performance.

Next, for code summarization, retrieving 4 or 8 examples from out-of-domain train data leads to performance comparable, or even better, than that of the model adapted using 8 examples from the test domain. This trend is observed for both Codex and CodeT5, particularly strongly when generalizing to new repositories and new organizations. A similar trend can be observed for code generation, and to a much stronger degree for CodeT5 - stratified supervision models can even outperform models trained with 32 examples from the test domain. However, while the performance of the stratified supervision models plateau after a certain number of examples, supervision on in-domain samples does not demonstrate such a trend.



(a) CodeT5



(b) Codex

Figure 4: Performance for models with downstream adaptations on ID and retrieved data.

RQ 3 *Can we have more generic solutions for out-of-domain generalization?*

In the previous experiment, we saw that models can generalize better to new domains without relying on labeled data from that domain. Unfortunately, this still requires adapting to every test domain individually for CodeT5, and even more strictly - to every test sample individually - for Codex. For example, for CodeT5, this means maintaining multiple copies of the model, performing the training for the adaptation stage multiple times, and storing a large amount of out-of-domain data to retrieve examples from.

In this experiment, we try to create more general models. Our interpretation of such a model for CodeT5 is generalizing to multiple domains without needing to train on them separately. For Codex, since previously we were obtaining demonstrations for each individual example, we consider sampling from demonstrations collected for the entire domain - in other words, sampling demonstrations from τ_{ret}. For CodeT5, we finetune it on the combined set of τ_{ret} for all domains. For Codex, for a query from τ_{test}, we consider sampling 4 or 8 demonstration examples from τ_{ret}.

Finding: We can build more generic models for code generation with finetuning without sacrificing the performance

The results for both models are presented in Table 4. Results for CodeT5 for this experiment are referred to as "FT: combined k", where k is the number of retrieved examples per test example. For each cell in the table, the first number is the raw score obtained by the "combined FT" model. It is followed by the difference between the score of the combined model and the score that we had previously obtained with domain-specific models. As can be seen, training a single model on combined retrieved samples results in a moderate drop in performance for code summarization, and a negligible drop for code generation. In other words, a model finetuned on stratified supervision data for new domains can be a viable solution for the out-of-domain generalization problem for code generation. Interestingly, this also indicates that for code generation, good performance on one domain does not hinder the performance on another domain, i.e. there is little to no negative transfer between different domains.

For Codex, the results of the experiment are referred to as "ICL: k from τ_{ret}" in Table 4, where k

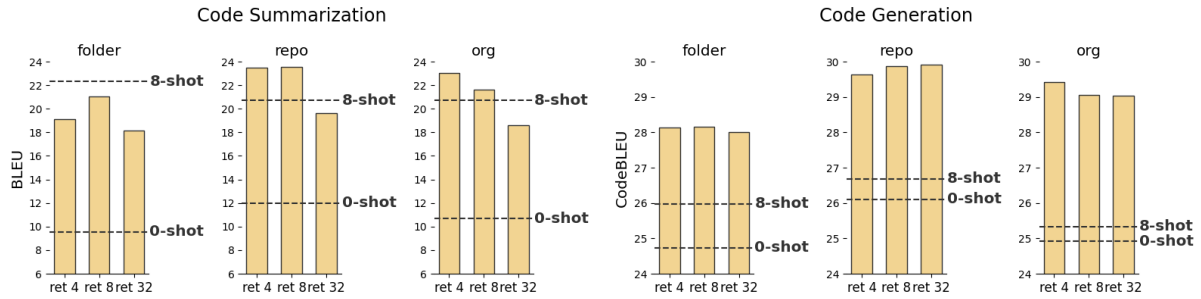


Figure 5: Performance for CodeT5 model finetuned with retrieved supervision, with different number of retrieved examples per test sample. Scores reported are BLEU for code summarization (left-most three plots), and CodeBLEU for code generation (right-most three plots). The performances of the CodeT5 MTL model evaluated in zero-shot, and 8-shot (ID) scenarios are illustrated with dotted lines for reference.

	Code Summarization BLEU / Δ BLEU			Code Generation CodeBLEU / Δ CodeBLEU		
	org	repo	folder	org	repo	folder
FT: combined 4	18.74 / -4.74	18.59 / -4.47	18.06 / -1.06	29.46 / -0.19	29.41 / -0.01	26.60 / -1.53
FT: combined 8	18.46 / -5.07	18.58 / -3.03	17.57 / -3.48	29.13 / -0.73	28.83 / -0.22	27.23 / -0.92
FT: combined 32	17.35 / -2.31	17.63 / -0.94	15.57 / -2.56	26.28 / -3.63	25.01 / -4.02	25.14 / -2.88
ICL: 4 from τ_{ret}	14.66 / -7.04	12.68 / -7.95	12.10 / -6.96	20.52 / -6.73	20.06 / -7.78	19.39 / -6.21
ICL: 8 from τ_{ret}	13.77 / -8.53	12.96 / -8.52	12.26 / -7.17	20.81 / -7.05	20.23 / -8.16	19.48 / -7.00

Table 4: Results for models using retrieved supervision examples in modified scenarios.

is the number of sampled demonstrations. The first number in each cell is the raw score obtained for Codex with sampling from similar examples for the domain, and the second number is the difference between that score, and the score obtained with similar demonstrations for each individual test example. It appears that for Codex replacing demonstrations selected for individual examples with those selected for a domain introduce too much noise, and degrade the performance a lot.

5 Limitations and Threats to Validity

As can be seen from Table 1, as a result of the process of filtering, we skew the data towards larger projects and eliminate from the dataset many samples that could potentially come from smaller projects. We believe that this step is necessary to make the results more reliable, due to the high variance that can be observed in datasets with very small test sets. However, we wanted to draw attention to this circumstance once more, to make sure that our findings are interpreted correctly.

Additionally, while we do evaluate distribution shift and out-of-domain generalization, we believe it is important to highlight again that the out-of-domain data in our analysis still originated from the same dataset. Thus its distribution is likely closer to the original training set distribution than it will be the case in the wild.

6 Conclusion

In this work, we systematically evaluated two large language models for code - CodeT5 and Codex (code-cushman-001) - on two fundamental code applications - code generation and code summarization. We studied how the models perform under distribution shifts that can commonly occur due to the nature of the software. We experimented with three granularities for defining domains in applications for code - organization, project, and module, or folder, level. Our experiments showed that both CodeT5 and Codex are susceptible to reduced performance due to domain shifts. We experimented with a number of training and domain adaptation techniques for achieving better out-of-domain generalization. We discovered that retrieving similar out-of-domain examples from training data is the most effective approach for adapting to a new, low-resource domain. In addition, we experimented with adapting models to multiple new domains simultaneously and found that such models can perform very well for code generation. However, we found the generality of the model to be a tradeoff for its performance for code summarization.

References

Daniele Angioni, Luca Demetrio, Maura Pintor, and Battista Biggio. 2022. Robust machine learning for malware detection over time. In *Proceedings of the Italian Conference on Cybersecurity (ITASEC 2022), Rome, Italy, June 20-23, 2022*, volume 3260 of *CEUR Workshop Proceedings*, pages 169–180. CEUR-WS.org.

Antreas Antoniou, Harrison Edwards, and Amos J. Storkey. 2018. How to train your maml. *ArXiv*, abs/1810.09502.

Ankur Bapna, N. Arivazhagan, and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

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, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff 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. *ArXiv*, abs/2005.14165.

Rich Caruana. 1996. Algorithms and applications for multitask learning. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 87–95. Morgan Kaufmann.

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28:41–75.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

Rajarshi Das, Manzil Zaheer, Dung Ngoc Thai, Ameya Godbole, Ethan Perez, Jay Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. 2021. Case-based reasoning for natural language queries over knowledge bases. In *Conference on Empirical Methods in Natural Language Processing*.

Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. 2022. Out of the BLEU: how should we assess quality of the code generation models? *CoRR*, abs/2208.03133.

Chelsea Finn, P. Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*.

Chelsea Finn, Kelvin Xu, and Sergey Levine. 2018. Probabilistic model-agnostic meta-learning. In *Neural Information Processing Systems*.

Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. 2019. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685.

Qiang Hu, Yuejun Guo, Xiaofei Xie, Maxime Cordy, Lei Ma, Mike Papadakis, and Yves Le Traon. 2022. Codes: A distribution shift benchmark dataset for source code learning. *arXiv preprint arXiv:2206.05480*.

Hamel Husain, Hongqi Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *ArXiv*, abs/1909.09436.

Gregory R. Koch. 2015. Siamese neural networks for one-shot image recognition.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *CoRR*, abs/2207.01780.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *ArXiv*, abs/2104.08691.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *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)*, abs/2101.00190.

Yan-Fu Li, Min Xie, and T. N. Goh. 2009. A study of mutual information based feature selection for case based reasoning in software cost estimation. *Expert Syst. Appl.*, 36(3):5921–5931.

Yufei Li, Simin Chen, and Wei Yang. 2021. Estimating predictive uncertainty under program data distribution shift. *CoRR*, abs/2107.10989.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *arXiv preprint arXiv:2205.05638*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? In *Workshop on Knowledge Extraction and Integration for Deep Learning Architectures; Deep Learning Inside Out*.

Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. 2012. Transfer learning for cross-company software defect prediction. *Inf. Softw. Technol.*, 54(3):248–256.

Carolyn Mair, Gada F. Kadoda, Martin Lefley, Keith Phalp, Chris Schofield, Martin J. Shepperd, and Steve Webster. 2000. An investigation of machine learning based prediction systems. *J. Syst. Softw.*, 53(1):23–29.

Elliot Meyerson and Risto Miikkulainen. 2019. Modular universal reparameterization: Deep multi-task learning across diverse domains. *ArXiv*, abs/1906.00097.

Pengyu Nie, Jiyang Zhang, Junyi Jessy Li, Raymond J. Mooney, and Milos Gligoric. 2022. Impact of evaluation methodologies on code summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 4936–4960. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Maja Popovic. 2015. chrf: character n-gram f-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation, WMT@EMNLP 2015, 17-18 September 2015, Lisbon, Portugal*, pages 392–395. The Association for Computer Linguistics.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, M. Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *ArXiv*, abs/2009.10297.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Adam Santoro, Sergey Bartunov, Matthew M. Botvinick, Daan Wierstra, and Timothy P. Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*.

Richard Shin, C. H. Lin, Sam Thomson, Charles C. Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jas' Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. *ArXiv*, abs/2104.08768.

Daniel L. Silver. 1996. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connect. Sci.*, 8(2):277–294.

Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical networks for few-shot learning. *ArXiv*, abs/1703.05175.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. 2017. Learning to compare: Relation network for few-shot learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

Sebastian Thrun and Lorien Y. Pratt. 1998. Learning to learn: Introduction and overview. In Sebastian Thrun and Lorien Y. Pratt, editors, *Learning to Learn*, pages 3–17. Springer.

Burak Turhan. 2012. On the dataset shift problem in software engineering prediction models. *Empir. Softw. Eng.*, 17(1-2):62–74.

Ricardo Vilalta and Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. 2017. Matching networks for one shot learning.

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

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Yongxin Yang and Timothy M. Hospedales. 2016. Deep multi-task representation learning: A tensor factorisation approach. *ArXiv*, abs/1605.06391.

Thomas Zimmermann, Nachiappan Nagappan, Harald C. Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009*, pages 91–100. ACM.

7 Appendix

7.1 Javascript Keywords

The Javascript keywords that we included in the CodeBleu implementation for evaluation is listed in table 7.1.

7.2 Extended Background

7.2.1 Meta-learning and Multi-task-learning

Meta-learning focuses on adapting knowledge gained from previous tasks to be applied to new tasks with limited training examples. Most meta-learning algorithms can be categorized into three groups: 1) Black-box meta-learning approaches (Santoro et al., 2016) train a black-box model to take in training data of a target task to output parameters for the neural network used for making prediction for that task; 2) Optimization-based methods (Finn et al., 2017, 2018; Antoniou et al., 2018) uses gradient descent to learn model parameters which can be adapted to a future target task with few gradient steps on a few-shot training dataset; 3) Non-parametric methods (Vinyals et al., 2017; Snell et al., 2017; Sung et al., 2017; Koch, 2015) learns a metric space in which predictions can be performed by computing some similarity metric, like distance and cosine similarity, to representations of each class. In our work, we are using the MAML (Finn et al., 2017) approach, which is a gradient-based method and learns model initialization (i.e., initial parameters) that is amenable to fast fine-tuning with few instances. This method is a conceptually simple and model-agnostic algorithm that has been shown to outperform existing approaches in several tasks.

Multi-task Learning aims to jointly learn several related tasks providing a generalized representation with the added benefit of compute and memory in terms of shared model parameters (Yang and Hospedales, 2016; Caruana, 1997; Meyerson and Miikkulainen, 2019). MTL also has a regularization effect on the model parameters. By definition, MTL aims to solve a fixed number of known tasks, whereas the point of meta-learning is often to solve unseen future tasks. But both methods capture a good prior from the training tasks, which can be used for getting model parameters for future target tasks.

In our work, we have experimented with both MAML and multi-task learning to check which of the method gives us a better prior for few-shot performance in our setting.

7.2.2 Few-shot Methods

Parameter-efficient finetuning: Conventional fine-tuning methods retrains all the model parameters for every new task, which becomes infeasible as the model size increases to the level of GPT-3. In recent times, parameter-efficient methods have been studied and it has been demonstrated that state-of-the-art PEFT methods can match the performance of finetuning all the model's parameters while updating only a tiny fraction of the model parameters. Initially adapters (Raffel et al., 2019; Houlsby et al., 2019; Bapna et al., 2019) were introduced, which are new feed-forward modules added between the layers of the fixed pre-trained model. Since then, various sophisticated PEFT methods have been proposed, including methods like LoRA that produce low-rank updates (Hu et al., 2021) and prompt tuning (Lester et al., 2021) and prefix-tuning (Li and Liang, 2021) concatenate learned continuous embeddings to the model's input or activations to induce it to perform a task.

Retrieval-based Example selection: In a study conducted by Liu et al. (2021) , they explored how different prompts can impact the performance of GPT-3 and found that the use of in-context examples has a significant influence on the downstream results. To achieve this, they utilized an unsupervised sentence encoder to encode training examples and then retrieved the nearest neighbors for each test instance. On a similar note, Das et al. (2021) developed a supervised prompt retriever for answering knowledge-based questions. Their method used tailored supervision specifically designed for knowledge-based queries and relied on surface similarity between formal queries. Furthermore, Shin et al. (2021) employed GPT-3 to select examples for the prompt in few-shot semantic parsing. They demonstrated the effectiveness of this approach by using GPT-3 to identify relevant examples for the prompt, which in turn improved the overall performance of the system.

7.3 Models

CodeT5: CodeT5 (Wang et al., 2021) is a pre-trained encoder-decoder transformer model based on T5 (Raffel et al., 2019) for programming languages. It uses a unified framework to support code understanding and generation tasks seam-

Languages	*Keywords*
javascript	await, break, case, catch, class, const, continue, debugger, default, delete, do, else, enum, export, extends, false, finally, for, function, if, implements, import, in, instanceof, interface, let, new, null, package, private, protected, public, return, super, switch, static, this, throw, try, true, typeof, var, void, while, with, yield

Table 5: Keywords used for CodeBLEU evaluation

lessly. To improve the model's ability to handle the unique characteristics of programming languages, CodeT5 is trained on an identifier-aware pretraining task. Additionally, the model is trained to exploit user-written code comments with a bimodal dual-generation task for better alignment between natural language and programming languages. This makes this model suitable for the applications that we consider. For both of our applications, we used the CodeT5-large model (Le et al., 2022) without making any changes to the model architecture.

Codex Codex (Chen et al., 2021) is the language model for code released by OpenAI. It is a GPT language model finetuned on 54 million public software repositories hosted on GitHub, containing 179 GB of unique Python files under 1 MB. VLLMs are capable of zero-shot generalization to unseen tasks, which is achieved by providing them with an *instruction* of what the model is expected to do. This allowed us to successfully evaluate Codex for both code generation and code summarization without any need for training.

8 Hyperparameters and training details

For full finetuning of CodeT5, we updated the model for 500 steps using batch size of 8, the best model was identified by the performance on the τ_{dev} portion. For LoRA, we use a rank of 4 with an initialization scale of 0.01 and update all the attention and feedforward layers. We train for 1000 steps with a batch size of 8.

For multitask learning (MTL) of CodeT5, we update the model for 150K steps on 80% of the X_{train} data, using a batch size of 4. The best checkpoint is selected by evaluating the model on the remaining 20% of X_{train} which was held-out from training. For dual-gen MTL, we followed the same train/dev division strategy as for MTL for code generation, and updated the model for 150K steps with batch size of 4. The best checkpoints were again decided by evaluating the model on the created development set. In particular, we selected two checkpoints - one according to CodeBLEU metric, and another according to BLEU metric for code generation and code summarization respectively. For Model-agnostic meta-learning, we updated the model from the pretrained CodeT5 checkpoint for 10K steps and used the last checkpoint in our experiments.

9 Additional experimental results

Besides the experiments presented in the main paper, in this section, we report some additional experiments, such as the results for code generation as measured using chrF and rougeL metrics, or comparison of LoRA parameter efficient finetuning method with the full model finetuning for CodeT5.

Code generation	folder			repo			org		
	8-shot	**16-shot**	**32-shot**	**8-shot**	**16-shot**	**32-shot**	**8-shot**	**16-shot**	**32-shot**
CodeT5 FT ID	19.36	20.92	21.95	20.42	22.44	24.47	19.29	20.73	22.6
CodeT5 LoRA ID	20.05	21.66	22.56	20.81	23.12	24.52	20.08	21.28	22.99
CodeT5 FT random	17.61	18.03	17.94	16.92	17.50	17.59	16.47	17.46	17.85
CodeT5 LoRA random	17.87	18.02	17.81	17.45	17.15	17.63	17.24	17.13	17.29
Codex ICL ID	28.78	-	-	31.05	-	-	29.19	-	-
Codex ICL random	20.62	-	-	20.87	-	-	21.10	-	-
Codex instr. only *(0-shot)*	(10.24)	-	-	(10.6)	-	-	(10.25)	-	-

Table 6: Comparison of model performance for code generation on in-domain (**ID**) vs out-of-domain (**random**) test data. Reported metric is ChrF (higher is better).

Code generation	folder			repo			org		
	8-shot	**16-shot**	**32-shot**	**8-shot**	**16-shot**	**32-shot**	**8-shot**	**16-shot**	**32-shot**
CodeT5 FT ID	14.15	15.84	16.73	14.93	16.98	19.19	13.75	14.93	16.94
CodeT5 LoRA ID	14.49	16.58	17.87	15.47	17.69	19.60	14.10	15.48	17.61
CodeT5 FT random	11.34	11.62	11.73	9.91	10.10	10.32	9.49	10.20	10.68
CodeT5 LoRA random	11.45	12.05	12.58	10.09	10.04	11.08	10.15	10.3	11.15
Codex ICL ID	23.70	-	-	24.62	-	-	22.58	-	-
Codex ICL random	15.76	-	-	15.67	-	-	15.81	-	-
Codex instr. only *(0-shot)*	(6.44)	-	-	(6.5)	-	-	(6.18)	-	-

Table 7: Comparison of model performance for code generation on in-domain (**ID**) vs out-of-domain (**random**) test data. Reported metric is RougeL (higher is better).

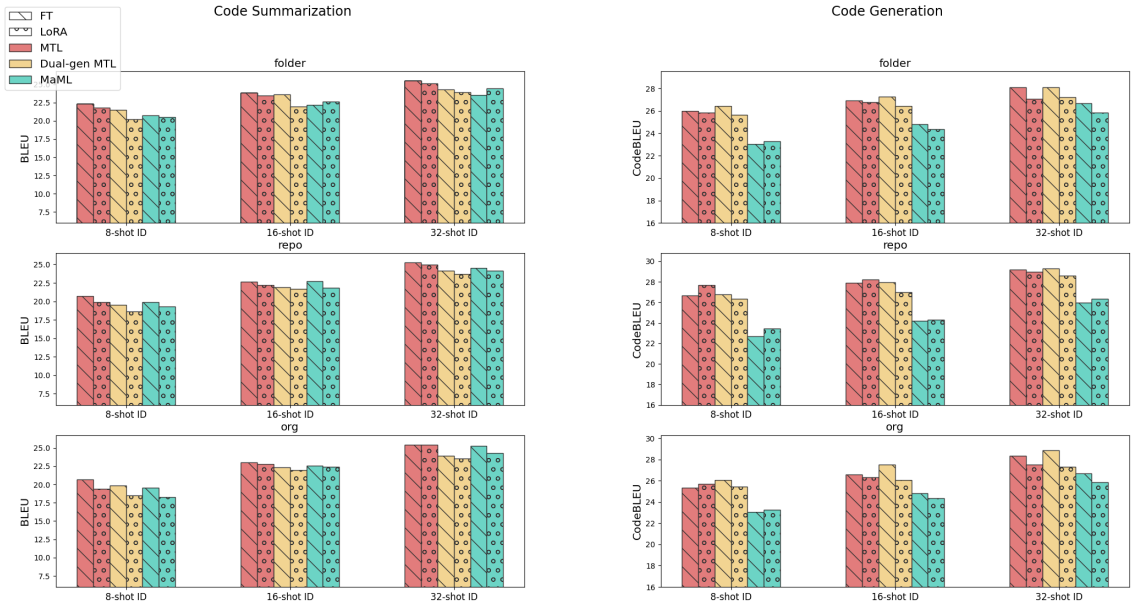


Figure 6: Performance for CodeT5 model finetuned with LoRA compared to regular finetuning.