How Does Code Pretraining Affect Language Model Task Performance?

Anonymous authors
Paper under double-blind review

Abstract

Large language models are increasingly trained on corpora containing both natural language and non-linguistic data like source code. Aside from aiding programming-related tasks, anecdotal evidence suggests that including code in pretraining corpora may improve performance on other, unrelated tasks, yet to date no work has been able to establish a causal connection by controlling between language and code data. Here we do just this. We pretrain language models on datasets which interleave natural language and code in two different settings: *competitive*, in which the total volume of data seen during pretraining is held constant; and *additive*, in which the volume of language data is held constant. We study how the pretraining mixture affects performance on (a) compositionality, measured by generalization accuracy on semantic parsing and syntactic transformation tasks, and more broadly on (b) downstream non-code-related objectives, measured by performance on tasks from the BigBench benchmark. We find that pretraining on higher proportions of code improves performance on compositional tasks involving structured output (like semantic parsing), and mathematics. Conversely, increase code mixture can harm performance on other tasks, including on tasks that requires sensitivity to linguistic structure such as syntax or morphology, and tasks measuring real-world knowledge.

1 Introduction

Large language models (LLMs) are increasingly used not only as natural-language assistants, but also for programming. LLMs which are trained on corpora containing code in various programming languages are used as programming assistants capable of generating code from natural-language descriptions (Chen et al., 2021), translating code between programming languages (Lachaux et al., 2020), decompilation of machine code into back into human-readable source code (Hosseini & Dolan-Gavitt, 2022), repairing vulnerabilities in existing code (Pearce et al., 2022), and even acting as programming agents when paired with tools (Yang et al., 2024a). These use cases have motivated adding code to pretraining corpora (see, *inter alia*, Gemini Team et al. 2024; OpenAI et al. 2024; Anthropic AI 2024; Groeneveld et al. 2024).

Concomitant to the inclusion of code in pretraining corpora, the performance of LLMs on many tasks has improved. Relevant for our purposes, many of the best-performing models include code in their pretraining corpus (see, *inter alia*, Fu & Khot 2022; Ye & Durrett 2022; Ye et al. 2023; Zhang et al. 2023; Zhou et al. 2023; Kim et al. 2024; Ma et al. 2024; Yang et al. 2024b; Razeghi et al. 2024; Coda-Forno et al. 2024; Longpre et al. 2024). That models trained in part on code perform well on several non-programming benchmarks raises intriguing questions: Does pretraining on code confer an advantage on non-programming tasks? If so, given a fixed compute budget, how much data should be allocated to code instead of natural-language data?

Establishing a causal relationship between code pretraining and downstream performance is difficult. Earlier studies have tackled these questions by comparing off-the-shelf code and no-code models (see, *inter alia*, Kim et al. 2024; Coda-Forno et al. 2024). Such observational studies are limited by the design choices of model creators and the availability of information about hyperparameters and training data. Many of the models typically surveyed are proprietary, and don't disclose this information. While pairs of open-source models differing only in their pretraining corpora do exist, such as Llama 2 & Code Llama (Touvron et al., 2023;

Roziere et al., 2023) or Gemma & CodeGemma (Gemma Team et al., 2024; Google, 2024), they often come with two important caveats: first, the code-variants of the models are derived by taking the non-code variants and conducting additional pretraining on code data, meaning the comparisons cannot control for total data volume; second, each pair treats the inclusion of code data as a binary variable, either present or absent, frustrating attempts to explore how changes in the *amount* of code influence downstream behavior.

We address these issues directly. We construct datasets that mix natural-language and source-code data at varying ratios, treating code inclusion as a continuous variable. We then pretrain language models of equal size on these parameterized datasets in two different experimental setups: a *competitive* setting where we keep the total volume of training data constant and vary the percentage allocated between code and natural language; and an *additive* setting where we keep the volume of language data constant and add additional amounts of code on top.

Previous work has found that augmenting training data with synthetic formal languages instantiating compositional patterns can improve compositional generalization (Papadimitriou & Jurafsky, 2023; Yao & Koller, 2024; Lindemann et al., 2024). Like formal languages, source code has a number of qualities which may aid models on seemingly-unrelated tasks: it is highly structured, by virtue of its conformance to the syntax of the programming language its written in; it is generally high-quality, owing to the use of linting and bug-checking tools and programming methodologies employed by its authors; it has interpretable semantics which is grounded by the functionality it describes; and, notably for compositionality, it contains instances of identical arguments and functions (e.g., variable names and method signatures). Informed by these observations, we evaluate our trained models for compositional generalization by finetuning them on three compositional generalization benchmarks (COGS, COGS-vf, and English Passivization). We also measure their performance on a broad array of tasks from BigBench to see how well code helps or hurts performance on unrelated domains.

We find that including code in a model's pretraining corpus has noticeable impacts on its performance on downstream tasks, in varying directions. Higher code mixtures improve performance in arithmetic and compositionality in domains whose output has formal structure (like semantic parsing). Conversely, increased exposure to code can harm language model performance on purely-linguistic tasks and tasks involving factual knowledge. We conduct permutation tests to study the impact of pretraining on downstream tasks and show that code pretraining increases the variance on task performance while raising the performance on the upper-quartile of tasks.

2 Related Work

Earlier work has studied whether pretraining on code is beneficial for non-programming tasks. Observational studies have looked at the impact of code on downstream performance post-hoc. Fu & Khot (2022) speculated that code pretraining is at least partially responsible for the improvement in capabilities between the `-001` and `-002` series of GPT-3(.5) models, specifically highlighting chain-of-thought reasoning, long-term dependency sensitivity, and "complex reasoning" as likely resulting from code pretraining. Yang et al. (2024b) provides a broad study of how code impacts language model capabilities, arguing that code improves complex reasoning and structured data understanding. Mueller et al. (2024) shows that code pretraining improves generalization on syntax-sensitive in-context learning tasks. By contrast, Coda-Forno et al. (2024), in an observational study, conclude that code pretraining does *not* improve model performance on a benchmark of behavioral tasks motivated by cognitive psychology. Kim et al. (2024) show that code pretraining improves models' entity-tracking capabilities.

Several experimental studies on the impact of code pretraining have also been conducted. Ma et al. (2024) attempt to verify the impact of code experimentally, comparing the 2.6 B parameter CodePanGu2.6 model trained on a mixture of natural-language and code data to Zeng et al. (2021)'s 2.6 B and 13 B parameter PanGu models of the same architecture trained only on natural language data. They conclude that code exposure, both during pretraining and instruction finetuning, is beneficial for performance on logical, legal, analogical, and scientific reasoning, and for chain-of-thought capabilities, though their experimental design

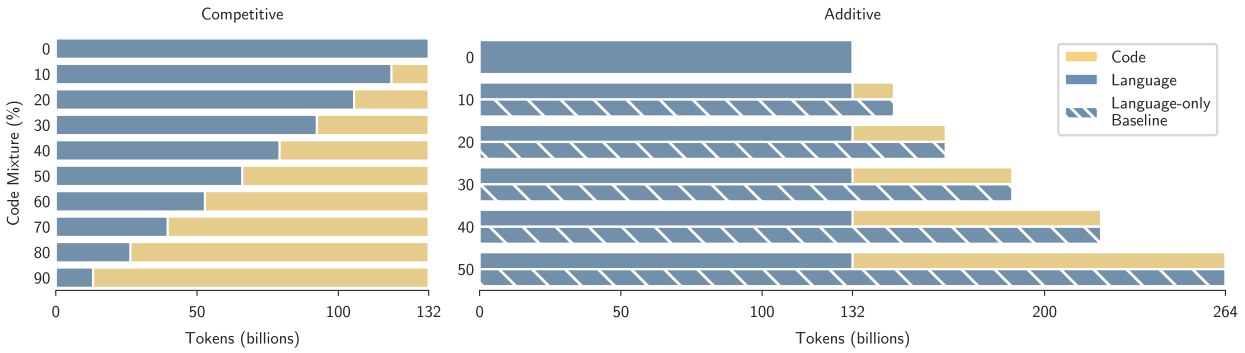

Figure 1: Code mixtures for the competitive and additive settings; for comparability between settings, we set the 0% mixture in both settings to have 132 B tokens of natural-language data (so $N_{\text{total}} = 132$ B tokens in the competitive setting, and $N_{\text{lang}} = 132$ B tokens in the additive setting). Note that these sequences are shuffled during training, so models see code and language data at the same time.

does not control for data volume (∼26.5 B tokens for PanGu2.6/13 versus[1] ∼42 B tokens for CodePanGu2.6) and does not quite control for model and training hyperparameters (models differ in the number of attention heads and use slightly different optimizer settings, which are magnified by the large difference in the number of training steps due to the difference in dataset size). Ma et al. (2024) also show exposing code to models early on during training can be helpful for some tasks. Longpre et al. (2024) show experimentally that removing code from a model's pretraining corpus harms performance on question answering in a number of different domains, though their experimental setup does not control for data volume and, consequently, other training hyperparameters sensitive to this.

3 Dataset Construction

To study how the amount of code in a language model's pretraining corpus impacts downstream performance, we construct datasets which interleave natural language and code sequences. The ingredients for our datasets are the English portion of the Colossal Cleaned Common Crawl (C4; Raffel et al. 2023) and cleaned code from GitHub.

Each dataset, which we refer to as a 'code mixture,' is parameterized by a single value $m \in [0, 1]$ representing the percentage of code in the training data, under the assumption that the C4 dataset has been fully cleaned of any code data. The mixture m relates the number of total tokens N_{total} in the dataset to the number of code N_{code} and language N_{lang} tokens via

$$N_{\text{code}} = m \cdot N_{\text{total}}, \qquad N_{\text{lang}} = (1 - m) \cdot N_{\text{total}}.$$

We construct families of training datasets in two different settings: *competitive*, in which the total amount of data is held constant while m varies, reducing the number of language tokens as the number of code tokens increases; and *additive*, in which the number of language tokens is held constant while the number of code tokens increases proportional to m (see fig. 1).

Competitive: Here, N_{total} is held constant while m varies between 0 and 0.9. This means that models trained on the 0% code mixture see $N_{\text{total}} = N_{\text{lang}}$ language tokens and 0 code tokens, while those trained on the 90% mixture see $0.1 \times N_{\text{total}}$ tokens of language data and $0.9 \times N_{\text{total}}$ tokens of code data.

[1]There is some ambiguity in the way Ma et al. (2024) describe their dataset: first, they cite that PanGu13 is trained on 1TB of data, but Zeng et al. (2021) report that it is trained on 100GB of data while their far larger 200 B parameter model is the one trained on 1TB of data; second, Ma et al. (2024) detail the individual data sources in GB but report the total dataset size in terms of tokens. It is unclear from phrasing whether their sampling strategy yields a dataset of 100 GB *in total*, or contains 100 GB of text data in addition to 50 GB of code data, but in either case the Table 4 in Zeng et al. (2021) shows that the 100 GB natural-language dataset used for the PanGu comparison models contains only ∼26.5 B tokens compared to CodePanGu's ∼42 B tokens.

This setting provides the clearest way to quantify the marginal utility of training on code instead of language, since we control for the total volume of data seen and consequently the total compute cost. However, the interpretability of results on mixtures with high values of m may be diminished since removing nearly all natural-language training data from a model's training corpus will lessen its ability to interpret and generate language; this in turn may greatly reduce its utility, even on code-related tasks, since the model will have far less ability to understand prompts or follow instructions. Additionally, the applicability of any results here to established pretraining setups may be limited by the fact that it will always be better in an absolute sense (and may be in a compute-optimal sense) to train a model on more data rather than less data (see, for instance, the conclusions of Hoffmann et al. 2022). Given this incentive, artificially limiting the amount of either code or language data provided to a model may not accurately reflect the considerations of model developers who, if they want to improve the code performance of a model, will simply add additional code data to the training corpus. To mitigate these issues, we also consider a second setting:

Additive: Here, N_{lang} is held constant while m varies between 0% and 50%. In order to keep N_{lang} fixed while m varies, we increase the number of total tokens proportionally:

$$N_{\text{total}} = N_{\text{lang}} \times \frac{1}{1-m}.$$

Since N_{total} increases unboundedly in m, we limit our study to consider additive mixtures of at most 50% code, which have twice as many tokens as the 0% mixture, which is identical to the 0% competitive mixture. This setting guarantees that all models have seen the same amount of natural language data, ameliorating the concern that any degradation in performance may result from insufficient exposure to natural language, but at the cost of failing to control for total data volume or compute. To further ensure that we can adequately compare code and non-code models across, we construct language-only baseline datasets for each code mixture. These datasets have the same number of total tokens, but with 100% of those tokens coming from natural language.

4 Experimental Setup

4.1 Model Construction & Training

We use the datasets constructed in section 3 as pretraining corpora for causally-masked decoder-only transformer language models (Vaswani et al., 2017; Radford et al., 2019). We construct 12-layer decoder-only models with roughly 374 M parameters. Model hyperparameters were chosen following the methdology of Wang et al. (2022) to approximate decoder-only versions of T5-large. We pretrain these models with a base natural language data volume of 132 B tokens. This means that all models in the competitive setting were trained with $N_{\text{total}} = 132$ B tokens, while the models in the additive setting were trained with $N_{\text{lang}} = 132$ B tokens, and hence N_{total} varying between 132 B tokens and 264 B tokens depending on the mixture; we use a batch size of 128, meaning that models were trained for between 1 M and 2 M steps, depending on the mixture and setting. For each combination of code mixture and setting, we pretrain models from five different random seeds.

4.2 Evaluation

We measure performance on three compositional generalization benchmarks and, more generally, on BigBench tasks. For each evaluation domain, we quantify the impact that code pretraining has on performance by calculating lines of best fit between performance (e.g., generalization accuracy for the compositional generalization benchmarks or multiple-choice grade for BigBench multiple choice tasks) and code mixture.

4.2.1 Compositional Generalization

Compositional generalization is a measure of how well a learner can generate and interpret novel, licit combinations of primitive pieces which have been previously learned. Originally motivated to describe human linguistic faculty—such as the ability of speakers to produce and understand an infinite number

COGS	x:	A hedgehog ate the cake .
	y:	$^*\text{cake}(x_4); \text{hedgehog}(x_1)$ AND $\text{eat.agent}(x_2, x_1)$ AND $\text{eat.theme}(x_2, x_4)$
COGS-vf	x:	A hedgehog ate the cake on the bed .
	y:	$\text{eat}(\text{agent} = \text{hedgehog}, \text{theme} = {}^*\text{cake}(\text{nmod.on} = {}^*\text{bed}))$
English Passivization	x:	our vultures admired her walrus above some zebra .
	y:	her walrus above some zebra was admired by our vultures .

Table 1: Examples of inputs (x) and targets (y) from each compositional generalization dataset.

of novel, grammatical sentences—compositionality is also a relevant property of many formal systems, like mathematics or programming languages. We hypothesize that the presence of source code in pretraining data may aid models in making this kind of generalization since source code often contains sequences in which a finite set of primitives (e.g., variable and method identifiers) are broadly combined.

To evaluate whether increased code mixture enables compositional generalization, we finetune our pretrained models on a suite of compositional generalization datasets: COGS (Kim & Linzen, 2020), a semantic parsing task in which natural-language sentences are transformed into a formal semantic representation; COGS-vf (Qiu et al., 2022), a variant of COGS which simplifies the output format; and English Passivization (Mueller et al., 2022), a natural-language transduction task in which synthetically generated active-voice sentences are transformed into passive variants. Each dataset contains training, validation, and generalization splits, where the generalization split is constructed to test licit-but-unattested combinations of familiar primitives. Table 1 shows examples of the input and output sequences for each of the datasets.

COGS and COGS-vf both divide their generalization split into two parts based on generalization type: either *lexical*, in which a known primitive is used in a grammatical position it has not been seen in before (e.g., *hedgehog* in subject position, when it had only been seen during training as an object); or *structural*, in which a known grammatical structure is used in a novel position (e.g., a prepositional phrase such as *on the mat* modifying the subject, when in training such phrases only modified objects). Previous studies involving COGS and COGS-vf have found the structural generalization examples in COGS to be much harder than the lexical generalization examples. Reducing the complexity of the output form, as is done in COGS-vf, makes the structural tasks somewhat *easier*, though not *easy*. Petty et al. (2024) found that models of a comparable size could attain accuracies near 90% on the lexical generalization examples from COGS but near 0% on the structural examples; on COGS-vf, models were able to attain accuracies greater than 95% on lexical cases and 10% on structural cases.

For all compositional generalization datasets, we finetune models for 10 K steps and report the mean full-sequence accuracy (i.e., 1 if every autoregressively-generated token is correct, 0 otherwise) over all examples in the generalization split for each random pretraining seed.

4.2.2 BigBench

We also evaluate models on BigBench (Srivastava et al., 2023), a benchmark of 204 diverse and challenging tasks presented in a common format. We evaluate models in a zero-shot setting, where a question is given in context (e.g., `What is 697 times 205?` from the *3-digit multiplication* task) and the model must either generate the correct label (e.g, `(a).`) from a provided list of responses (for multiple-choice tasks) or generate the correct answer (for generative tasks). Since our focus is on the effect of code in pretraining on non-code tasks, we exclude from consideration tasks which are explicitly designed to test the capabilities of models at understanding or generating source code. Table 2 shows examples of the input and output sequences for the BigBench tasks we discuss in detail.

5 Results

Code improves compositional generalization for structured outputs. When we finetune on COGS and COGS-vf, where the output domain has a formal structure, we find that performance improves as the

`bb-arithmetic`	x: What is 68824 times 42716? y: 9033448237, 3839424324, 18962582, 564059290599, banana, house, **2939885984**
`bb-common-morpheme`	x: What is the common morpheme among these words: pyre, empyrean, antipyretic, pyrotechnics? y: **fire**, hot, oxygen, medicine
`bb-fantasy-reasoning`	x: Long ago you had sold your soul to the devil, but the postal service was so utterly bad that they had lost the package where your soul was. Since the transaction was completed before it, you have the benefits of the deal while the devil still has no control over you. Does the devil have any control over your soul now? y: Yes, **No**
`bb-general-knowledge`	x: How many legs do horses have? y: two, **four**, six, three, one, none
`bb-implicatures`	x: Does Speaker 2's answer mean yes or no? Speaker 1: 'But aren't you afraid?' Speaker 2: 'Ma'am, sharks never attack anybody.' y: yes, **no**

Table 2: Examples of inputs (x) and answers (y) from selected multiple-choice BigBench tasks. Correct answers are bolded.

proportion of code increases in both the competitive and additive settings (see fig. 2 and table 3). The effect is most pronounced for the structural generalization examples from COGS-vf in the competitive and additive settings (regression coefficients $\hat{\beta} = 0.147$ and $\hat{\beta} = 0.165$, respectively; this indicates that the best-fit line predicts an accuracy increase of 14.7% as the proportion of code increases from 0% to 100%), though all code-mixture models show a non-negative relationship between code mixture and generalization accuracy. Code helped the least on the structural generalization examples from COGS, where absolute performance remained near-zero. In the additive setting, we find that code-mixture models perform as well (on lexical generalization examples) or better (on structural generalization examples) than the equivalent language-only baseline models.

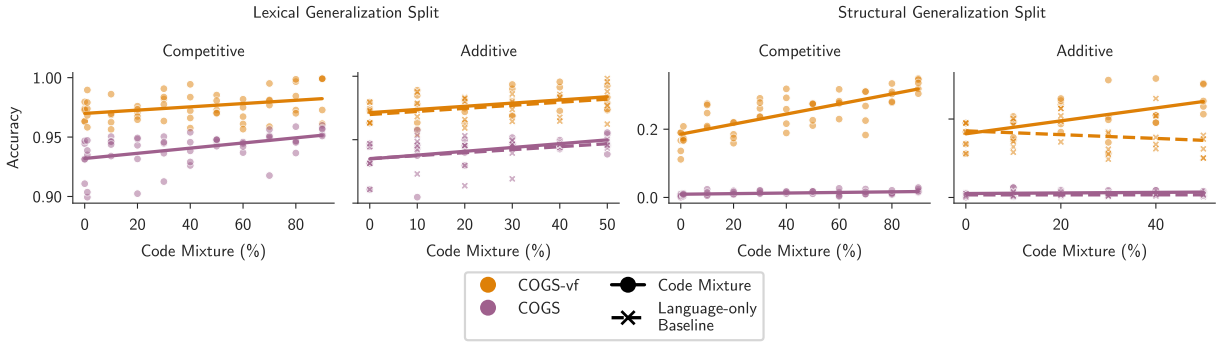

Figure 2: Full-sequence accuracy on the generalization set increases with code mixture on COGS and COGS-vf in both the competitive and additive settings. In the additive setting, code-mixture models outperform language-only baselines on the harder structural generalization cases. In all cases, validation accuracy is 100%.

In order for models to generalize compositionally, two things must happen: first, models must correctly generalize the distribution of arguments and predicates to match the true-but-unseen patterns of composition (e.g., they must learn that syntactic objects become arguments to 'theme' for all primitives, even those only previously seen as subjects); and they must produce well-formed outputs. Kim & Linzen (2020, §G.2) note that Transformer models in particular often failed at producing syntactically well-formed logical expressions for the generalization examples in COGS. Since code has similar syntactic requirements to those of COGS

logical expression (e.g., well-balanced parentheses), the improvement we observe in generalization accuracy may be due to improvements in the well-formedness of outputs, rather than due to better compositional generalization. To test this hypothesis, we compute a very high-level measure of syntactic well-formedness for model outputs—namely, whether or not the decoded logical forms have well-balanced parentheses—and examine how well-formedness varies by code mixture.

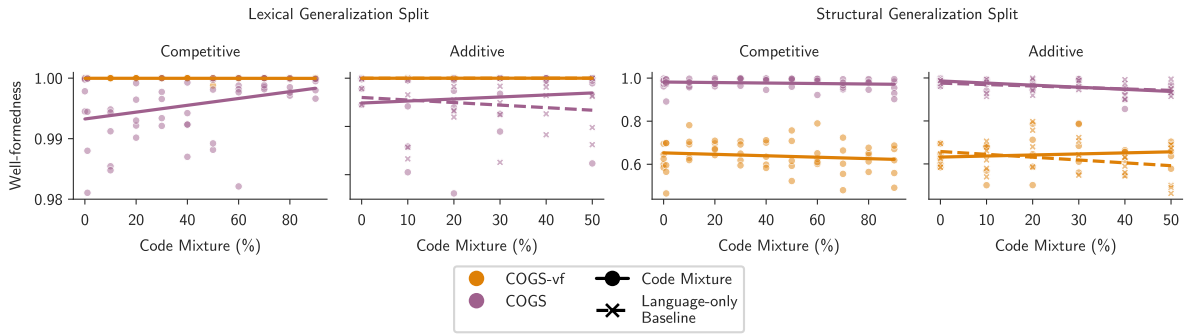

Figure 3: Pretraining code mixture has little impact on the well-formedness of generalization outputs in any setting.

Figure 3 shows that exposure to code does not, in general, improve the well-formedness of generalization outputs. Only on structural generalization examples from COGS-vf in the additive setting does the regression coefficient $\hat{\beta} = 0.049$ exceed 0.01; for all other code-mixture models, increased code mixture has a near-zero or negative impact on syntactic well-formedness (table 4). This means that the observed relationship between higher code mixture and generalization accuracy is attributable to models learning better generalizations for argument distribution rather than merely producing more well-formed outputs.

Code improves performance on arithmetic, up to a point. On multiple-choice multi-digit arithmetic tasks from BigBench, increased code mixture has a generally positive impact on performance. In both competitive and additive settings, higher code mixture results in greater multiple-choice accuracy, with the impact growing more pronounced as the number of digits increases (see fig. 4 and table 6). In the competitive setting, performance peaks at a code mixture between 40% and 50% and thereafter tends to decrease, though the overall trend remains positive; this inverted-U shaped performance curve also grows more pronounced as the number of digits increases.

Code distracts from linguistic- and world-knowledge. We also identify cases where increased exposure to code *harms* performance by looking for tasks whose performance is negatively correlated with code mixture. These tasks include ones which involve purely linguistic knowledge (such as the English Passivization compositional generalization task as well as the Implicatures and Common Morpheme BigBench tasks) as well as those which involve reasoning or world-knowledge (such as the General Knowledge and Fantasy Reasoning BigBench tasks).

Figure 5 shows this negative trend on the English Passivization compositional generalization benchmark, where performance (as measured by mean full-sequence accuracy on the generalization split) decreases as code mixture increases in both the competitive and additive settings. Furthermore, in the additive setting the language-only baseline models outperform the code-mixture models. See table 5 for exact regression coefficients.

These negative trends show that increased exposure to code during pretraining does not uniformly improve the ability of language models to generalize compositionally independent of the output domain; whereas COGS and COGS-vf, whose output domain is formal logic expressions, benefit from increased code exposure, generalization tasks which involve natural-language output domains appear to obviate any compositionality benefit conferred to models through code exposure. This may make intuitive sense, as decreased exposure to natural language data (in either an absolute or relative sense) may reduce any linguistically-relevant

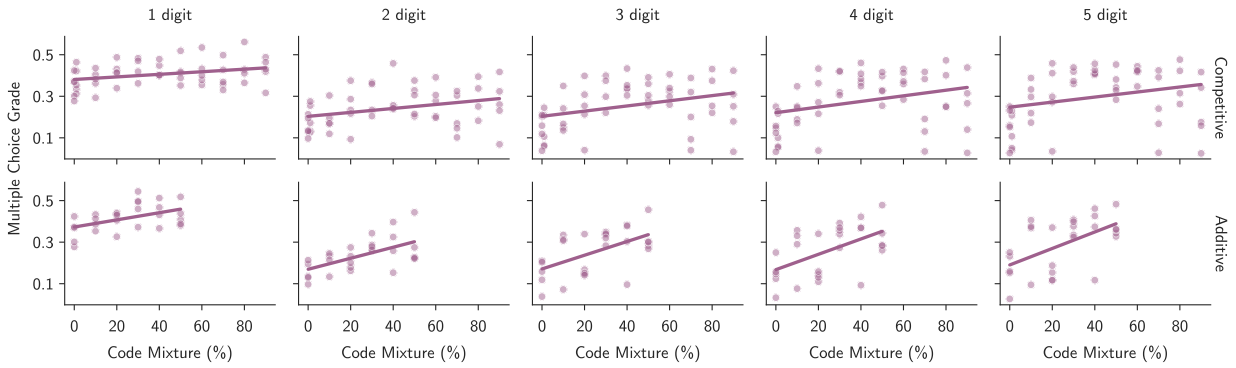

Figure 4: On multi-digit multiple choice arithmetic tasks, performance modestly increases with code mixture in the additive setting, while it increases then decreases in competitive. In both settings, the effect is more pronounced as the number of digits (rows) increases.

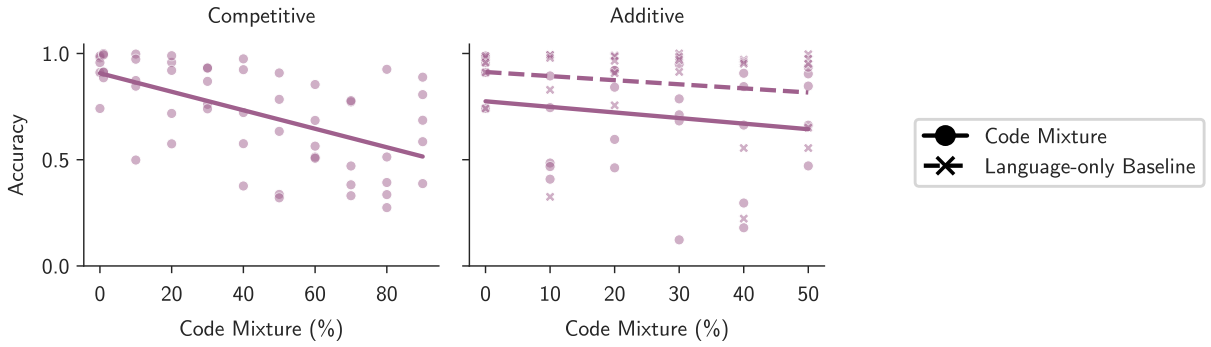

Figure 5: On English Passivization, a compositional generalization benchmark where (unlike COGS) both the inputs and outputs are in natural language, increased code mixture results in lower full-sequence generalization accuracy in both settings. In the additive setting, code-mixture models underperform language-only baselines on the harder structural generalization cases.

inductive biases models need, in partial conflict with Mueller et al. (2024)'s finding that code pretraining aids syntax-sensitive generalization for *in-context learning* tasks.

We also find instances of BigBench tasks where code mixture is negatively correlated with performance; Figure 6 highlights four such tasks where increased exposure to code during pretraining harms performance in both competitive and additive settings. See table 7 for exact regression coefficients.

5.1 The impact of code in aggregate

The results presented above highlight particular cases where code mixture has a noticeable impact on performance, but how does code pretraining affect the remaining BigBench tasks? We want to know how code pretraining impacts performance in aggregate for two reasons. First, we want to know if adding code helps *in general*: is adding code helpful or harmful for most tasks? Second, since it's likely that following any type of intervention models will be better at some tasks and worse at others than before the intervention, we want to confirm if the effects of code we observe are statistically significant or could have arisen due to chance.

To answer this, we perform a permutation test on the slopes derived above from best-linear-fits of task performance versus code mixture. We start by taking the underlying performance-by-mixture data and

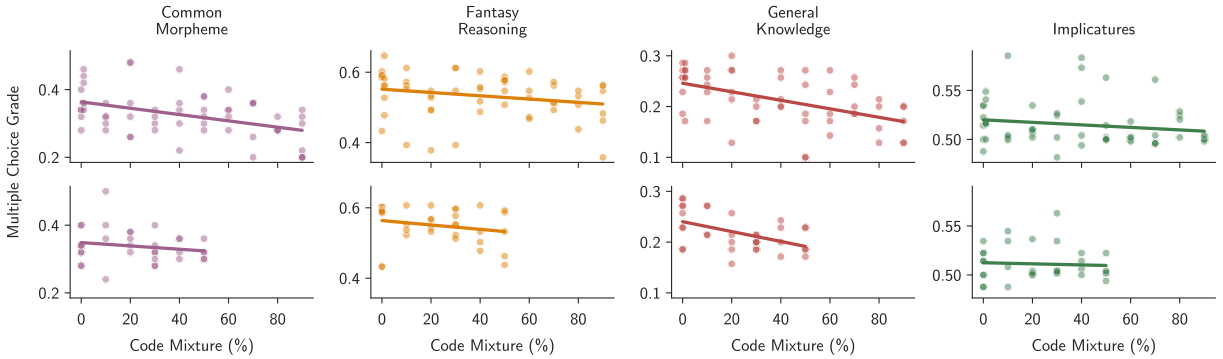

Figure 6: On a variety of BigBench tasks involving linguistic or factual knowledge, increased code mixture reduces accuracy.

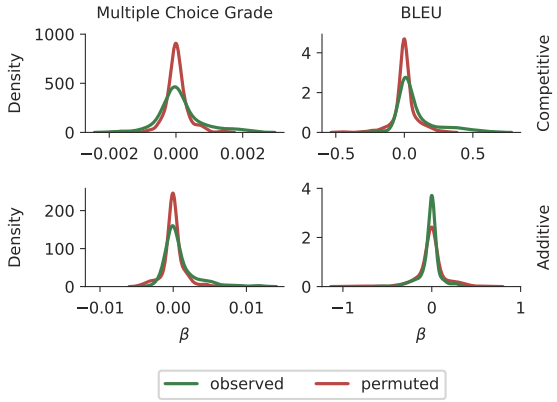

Figure 7: Kernel Density Estimates for the slopes β of linear regressions between task performance and code mixture on BigBench tasks.

shuffling the independent variable (code mixture) within each task and recompute slopes for the lines-of-best-fit. Figure 7 shows the distribution of slopes for the observed (treatment) and counterfactual, permuted (control) data for both settings and metrics. For multiple choice tasks in both settings and for generative tasks in the competitive setting, the distribution of treatment slopes (i.e., those observed) is less concentrated around 0 than the control distribution.

To quantify the difference between these distributions, we compute several different test statistics: the difference of means ($\Delta\mu$) as a measure of whether training on code improves task performance on average; the difference of variance (ΔVar) as a measure of whether training on code increases the variance of task performance; the difference of skew (ΔSkew) as a measure of whether training on code moves the distribution of task performance asymmetrically; and the differences in upper and lower quartiles (ΔUpper/LowerQuartile) as a measure of whether training on code increases the model's performance on its best and worst-performing tasks.

We then perform two-sided permutation tests against the null hypothesis that the treatment and control distributions are drawn from the same underlying distribution by combining and randomly-repartitioning the samples 10 K times and recomputing each test statistic. We do this test independently for each setting (competitive and additive) and each BigBench question type: multiple choice (MCG) and generative (where performance is measured by BLEU).

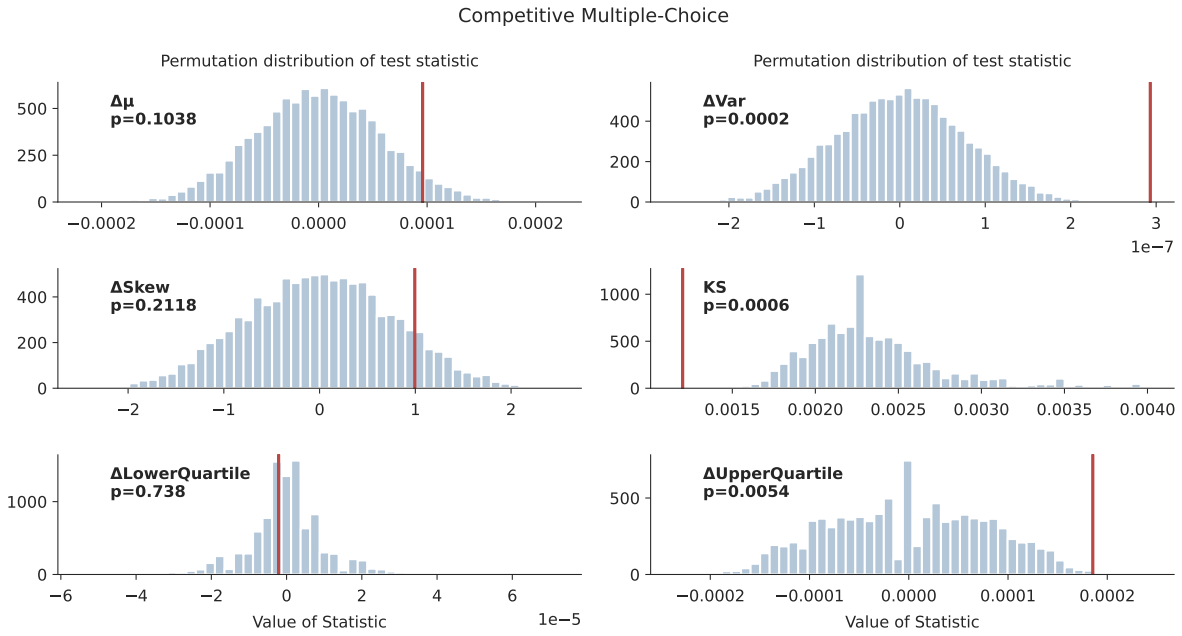

Figure 8: Null distributions (blue histogram) and observed values (red vertical rules) for various test statistics under a permutation test for slopes of performance by code mixture on Big Bench tasks with multiple choice grades in the competitive setting.

Figure 8 shows the null distributions for each of the test statistics and the observed values for the multiple-choice questions in the competitive setting, along with the significance scores (p-values) for each statistic. We find a statistically significant difference of variance ($p = 0.0002$) and upper-quartiles ($p = 0.006$) at a significance level of $\alpha = 0.05$, indicating that increased code exposure in pretraining does have strong benefits for some tasks, while it increases the variance in downstream task performance in general. Other statistics measured were not significant at this significance level. Results are similar, in general, for other conditions.

6 Discussion

We find that including code in a model's pretraining corpus influences its performance on downstream, non-code tasks. Adding code improves performance on compositional generalization tasks whose output domain is highly structured, akin to the syntactic constraints of source code. Exposure to code during pretraining also improves performance on arithmetic tasks, an trend which grows more pronounced as the number of digits of the numbers included in those arithmetic tasks increases. Conversely, we also find tasks where increased exposure to code harms model performance, such as compositional generalization tasks involving natural-language output or tasks involving linguistic or real-world knowledge. These trends appear in both a competitive setting, where increases in code data result in reduction of language data, and in a additive setting, where all models see a fixed amount of language data.

Despite the fact that code improves compositional generalization only in cases where the output domain is 'code-like,' we find that increased code exposure does not meaningfully improve the syntactic well-formedness of outputs in these cases; rather, the benefit conferred by code is to allow models to better learn the correct generalization for the distribution of arguments. We hypothesize that the deleterious impact of code on tasks involving linguistic or real-world knowledge comes from a reduction in linguistically-relevant inductive biases as models see less natural language data (either in an absolute sense in the competitive setting or a relative sense in the additive setting).

We conduct permutation tests on the distributions of per-task trend lines of performance-by-code-mixture to quantify the impact that code has on performance. We find that, in aggregate, training on code tends to improve performance on BigBench tasks at a statistically-significant level.

6.1 Limitations and Future Work

Scale We survey relatively small models (374 M parameters), which limits our ability to establish how code pretraining affects capabilities which require models at the multi-billion parameter scale, like instruction following and advanced in-context learning. We also only consider pretraining corpora of between 132 B and 264 B tokens.

Data Sources We treat 'code' and 'language' as a monolithic and disjoint data sources, but in reality source code contains linguistic data in the form of comments while natural language datasets may contain code-like structures even after cleaning and curation. It is possible that effect sizes would be increased with a more thorough separation of code and language data.

Task Limitations We study a small set of tasks and evaluation modalities (fine-tuning on compositional generalization benchmarks and zero-shot performance on assorted BigBench tasks). Code pretraining may have impacts on other tasks, and those impacts may differ between fine-tuning, zero-shot, and multi-shot in-context learning.

References

Anthropic AI. The Claude 3 Model Family: Opus, Sonnet, Haiku, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri 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, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Julian Coda-Forno, Marcel Binz, Jane X. Wang, and Eric Schulz. Cogbench: a large language model walks into a psychology lab, 2024. URL `https://arxiv.org/abs/2402.18225`.

Hao Fu, Yao; Peng and Tushar Khot. How does gpt obtain its ability? tracing emergent abilities of language models to their sources. *Yao Fu's Notion*, Dec 2022. URL `https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57`

Gemini Team, Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry, Lepikhin, Timothy Lillicrap, Jean baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, James Molloy, Jilin Chen, Michael Isard, Paul Barham, Tom Hennigan, Ross McIlroy, Melvin Johnson, Johan Schalkwyk, Eli Collins, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Clemens Meyer, Gregory Thornton, Zhen Yang, Henryk Michalewski, Zaheer Abbas, Nathan Schucher, Ankesh Anand, Richard Ives, James Keeling, Karel Lenc, Salem Haykal, Siamak Shakeri, Pranav Shyam, Aakanksha Chowdhery, Roman Ring, Stephen Spencer, Eren Sezener, Luke Vilnis, Oscar Chang, Nobuyuki Morioka, George Tucker, Ce Zheng, Oliver Woodman, Nithya Attaluri, Tomas Kocisky, Evgenii Eltyshev, Xi Chen, Timothy Chung, Vittorio Selo, Siddhartha Brahma, Petko Georgiev, Ambrose Slone, Zhenkai Zhu, James Lottes, Siyuan Qiao, Ben Caine, Sebastian Riedel, Alex Tomala, Martin Chadwick, Juliette Love, Peter Choy, Sid Mittal, Neil Houlsby, Yunhao Tang, Matthew Lamm, Libin Bai, Qiao Zhang, Luheng He, Yong Cheng, Peter

Humphreys, Yujia Li, Sergey Brin, Albin Cassirer, Yingjie Miao, Lukas Zilka, Taylor Tobin, Kelvin Xu, Lev Proleev, Daniel Sohn, Alberto Magni, Lisa Anne Hendricks, Isabel Gao, Santiago Ontanon, Oskar Bunyan, Nathan Byrd, Abhanshu Sharma, Biao Zhang, Mario Pinto, Rishika Sinha, Harsh Mehta, Dawei Jia, Sergi Caelles, Albert Webson, Alex Morris, Becca Roelofs, Yifan Ding, Robin Strudel, Xuehan Xiong, Marvin Ritter, Mostafa Dehghani, Rahma Chaabouni, Abhijit Karmarkar, Guangda Lai, Fabian Mentzer, Bibo Xu, YaGuang Li, Yujing Zhang, Tom Le Paine, Alex Goldin, Behnam Neyshabur, Kate Baumli, Anselm Levskaya, Michael Laskin, Wenhao Jia, Jack W. Rae, Kefan Xiao, Antoine He, Skye Giordano, Lakshman Yagati, Jean-Baptiste Lespiau, Paul Natsev, Sanjay Ganapathy, Fangyu Liu, Danilo Martins, Nanxin Chen, Yunhan Xu, Megan Barnes, Rhys May, Arpi Vezer, Junhyuk Oh, Ken Franko, Sophie Bridgers, Ruizhe Zhao, Boxi Wu, Basil Mustafa, Sean Sechrist, Emilio Parisotto, Thanumalayan Sankaranarayana Pillai, Chris Larkin, Chenjie Gu, Christina Sorokin, Maxim Krikun, Alexey Guseynov, Jessica Landon, Romina Datta, Alexander Pritzel, Phoebe Thacker, Fan Yang, Kevin Hui, Anja Hauth, Chih-Kuan Yeh, David Barker, Justin Mao-Jones, Sophia Austin, Hannah Sheahan, Parker Schuh, James Svensson, Rohan Jain, Vinay Ramasesh, Anton Briukhov, Da-Woon Chung, Tamara von Glehn, Christina Butterfield, Priya Jhakra, Matthew Wiethoff, Justin Frye, Jordan Grimstad, Beer Changpinyo, Charline Le Lan, Anna Bortsova, Yonghui Wu, Paul Voigtlaender, Tara Sainath, Shane Gu, Charlotte Smith, Will Hawkins, Kris Cao, James Besley, Srivatsan Srinivasan, Mark Omernick, Colin Gaffney, Gabriela Surita, Ryan Burnell, Bogdan Damoc, Junwhan Ahn, Andrew Brock, Mantas Pajarskas, Anastasia Petrushkina, Seb Noury, Lorenzo Blanco, Kevin Swersky, Arun Ahuja, Thi Avrahami, Vedant Misra, Raoul de Liedekerke, Mariko Iinuma, Alex Polozov, Sarah York, George van den Driessche, Paul Michel, Justin Chiu, Rory Blevins, Zach Gleicher, Adrià Recasens, Alban Rrustemi, Elena Gribovskaya, Aurko Roy, Wiktor Gworek, Sébastien M. R. Arnold, Lisa Lee, James Lee-Thorp, Marcello Maggioni, Enrique Piqueras, Kartikeya Badola, Sharad Vikram, Lucas Gonzalez, Anirudh Baddepudi, Evan Senter, Jacob Devlin, James Qin, Michael Azzam, Maja Trebacz, Martin Polacek, Kashyap Krishnakumar, Shuo yiin Chang, Matthew Tung, Ivo Penchev, Rishabh Joshi, Kate Olszewska, Carrie Muir, Mateo Wirth, Ale Jakse Hartman, Josh Newlan, Sheleem Kashem, Vijay Bolina, Elahe Dabir, Joost van Amersfoort, Zafarali Ahmed, James Cobon-Kerr, Aishwarya Kamath, Arnar Mar Hrafnkelsson, Le Hou, Ian Mackinnon, Alexandre Frechette, Eric Noland, Xiance Si, Emanuel Taropa, Dong Li, Phil Crone, Anmol Gulati, Sébastien Cevey, Jonas Adler, Ada Ma, David Silver, Simon Tokumine, Richard Powell, Stephan Lee, Kiran Vodrahalli, Samer Hassan, Diana Mincu, Antoine Yang, Nir Levine, Jenny Brennan, Mingqiu Wang, Sarah Hodkinson, Jeffrey Zhao, Josh Lipschultz, Aedan Pope, Michael B. Chang, Cheng Li, Laurent El Shafey, Michela Paganini, Sholto Douglas, Bernd Bohnet, Fabio Pardo, Seth Odoom, Mihaela Rosca, Cicero Nogueira dos Santos, Kedar Soparkar, Arthur Guez, Tom Hudson, Steven Hansen, Chulayuth Asawaroengchai, Ravi Addanki, Tianhe Yu, Wojciech Stokowiec, Mina Khan, Justin Gilmer, Jaehoon Lee, Carrie Grimes Bostock, Keran Rong, Jonathan Caton, Pedram Pejman, Filip Pavetic, Geoff Brown, Vivek Sharma, Mario Lučić, Rajkumar Samuel, Josip Djolonga, Amol Mandhane, Lars Lowe Sjösund, Elena Buchatskaya, Elspeth White, Natalie Clay, Jiepu Jiang, Hyeontaek Lim, Ross Hemsley, Zeyncep Cankara, Jane Labanowski, Nicola De Cao, David Steiner, Sayed Hadi Hashemi, Jacob Austin, Anita Gergely, Tim Blyth, Joe Stanton, Kaushik Shivakumar, Aditya Siddhant, Anders Andreassen, Carlos Araya, Nikhil Sethi, Rakesh Shivanna, Steven Hand, Ankur Bapna, Ali Khodaei, Antoine Miech, Garrett Tanzer, Andy Swing, Shantanu Thakoor, Lora Aroyo, Zhufeng Pan, Zachary Nado, Jakub Sygnowski, Stephanie Winkler, Dian Yu, Mohammad Saleh, Loren Maggiore, Yamini Bansal, Xavier Garcia, Mehran Kazemi, Piyush Patil, Ishita Dasgupta, Iain Barr, Minh Giang, Thais Kagohara, Ivo Danihelka, Amit Marathe, Vladimir Feinberg, Mohamed Elhawaty, Nimesh Ghelani, Dan Horgan, Helen Miller, Lexi Walker, Richard Tanburn, Mukarram Tariq, Disha Shrivastava, Fei Xia, Qingze Wang, Chung-Cheng Chiu, Zoe Ashwood, Khuslen Baatarsukh, Sina Samangooei, Raphaël Lopez Kaufman, Fred Alcober, Axel Stjerngren, Paul Komarek, Katerina Tsihlas, Anudhyan Boral, Ramona Comanescu, Jeremy Chen, Ruibo Liu, Chris Welty, Dawn Bloxwich, Charlie Chen, Yanhua Sun, Fangxiaoyu Feng, Matthew Mauger, Xerxes Dotiwalla, Vincent Hellendoorn, Michael Sharman, Ivy Zheng, Krishna Haridasan, Gabe Barth-Maron, Craig Swanson, Dominika Rogozińska, Alek Andreev, Paul Kishan Rubenstein, Ruoxin Sang, Dan Hurt, Gamaleldin Elsayed, Renshen Wang, Dave Lacey, Anastasija Ilić, Yao Zhao, Adam Iwanicki, Alejandro Lince, Alexander Chen, Christina Lyu, Carl Lebsack, Jordan Griffith, Meenu Gaba, Paramjit Sandhu, Phil Chen, Anna Koop, Ravi Rajwar, Soheil Hassas Yeganeh, Solomon Chang, Rui Zhu, Soroush Radpour, Elnaz Davoodi, Ving Ian Lei, Yang Xu, Daniel Toyama, Constant Segal, Martin Wicke, Hanzhao Lin, Anna Bulanova, Adrià Puigdomènech

Badia, Nemanja Rakićević, Pablo Sprechmann, Angelos Filos, Shaobo Hou, Víctor Campos, Nora Kassner, Devendra Sachan, Meire Fortunato, Chimezie Iwuanyanwu, Vitaly Nikolaev, Balaji Lakshminarayanan, Sadegh Jazayeri, Mani Varadarajan, Chetan Tekur, Doug Fritz, Misha Khalman, David Reitter, Kingshuk Dasgupta, Shourya Sarcar, Tina Ornduff, Javier Snaider, Fantine Huot, Johnson Jia, Rupert Kemp, Nejc Trdin, Anitha Vijayakumar, Lucy Kim, Christof Angermueller, Li Lao, Tianqi Liu, Haibin Zhang, David Engel, Somer Greene, Anaïs White, Jessica Austin, Lilly Taylor, Shereen Ashraf, Dangyi Liu, Maria Georgaki, Irene Cai, Yana Kulizhskaya, Sonam Goenka, Brennan Saeta, Ying Xu, Christian Frank, Dario de Cesare, Brona Robenek, Harry Richardson, Mahmoud Alnahlawi, Christopher Yew, Priya Ponnapalli, Marco Tagliasacchi, Alex Korchemniy, Yelin Kim, Dinghua Li, Bill Rosgen, Kyle Levin, Jeremy Wiesner, Praseem Banzal, Praveen Srinivasan, Hongkun Yu, Çağlar Ünlü, David Reid, Zora Tung, Daniel Finchelstein, Ravin Kumar, Andre Elisseeff, Jin Huang, Ming Zhang, Ricardo Aguilar, Mai Giménez, Jiawei Xia, Olivier Dousse, Willi Gierke, Damion Yates, Komal Jalan, Lu Li, Eri Latorre-Chimoto, Duc Dung Nguyen, Ken Durden, Praveen Kallakuri, Yaxin Liu, Matthew Johnson, Tomy Tsai, Alice Talbert, Jasmine Liu, Alexander Neitz, Chen Elkind, Marco Selvi, Mimi Jasarevic, Livio Baldini Soares, Albert Cui, Pidong Wang, Alek Wenjiao Wang, Xinyu Ye, Krystal Kallarackal, Lucia Loher, Hoi Lam, Josef Broder, Dan Holtmann-Rice, Nina Martin, Bramandia Ramadhana, Mrinal Shukla, Sujoy Basu, Abhi Mohan, Nick Fernando, Noah Fiedel, Kim Paterson, Hui Li, Ankush Garg, Jane Park, DongHyun Choi, Diane Wu, Sankalp Singh, Zhishuai Zhang, Amir Globerson, Lily Yu, John Carpenter, Félix de Chaumont Quitry, Carey Radebaugh, Chu-Cheng Lin, Alex Tudor, Prakash Shroff, Drew Garmon, Dayou Du, Neera Vats, Han Lu, Shariq Iqbal, Alex Yakubovich, Nilesh Tripuraneni, James Manyika, Haroon Qureshi, Nan Hua, Christel Ngani, Maria Abi Raad, Hannah Forbes, Jeff Stanway, Mukund Sundararajan, Victor Ungureanu, Colton Bishop, Yunjie Li, Balaji Venkatraman, Bo Li, Chloe Thornton, Salvatore Scellato, Nishesh Gupta, Yicheng Wang, Ian Tenney, Xihui Wu, Ashish Shenoy, Gabriel Carvajal, Diana Gage Wright, Ben Bariach, Zhuyun Xiao, Peter Hawkins, Sid Dalmia, Clement Farabet, Pedro Valenzuela, Quan Yuan, Ananth Agarwal, Mia Chen, Wooyeol Kim, Brice Hulse, Nandita Dukkipati, Adam Paszke, Andrew Bolt, Kiam Choo, Jennifer Beattie, Jennifer Prendki, Harsha Vashisht, Rebeca Santamaria-Fernandez, Luis C. Cobo, Jarek Wilkiewicz, David Madras, Ali Elqursh, Grant Uy, Kevin Ramirez, Matt Harvey, Tyler Liechty, Heiga Zen, Jeff Seibert, Clara Huiyi Hu, Andrey Khorlin, Maigo Le, Asaf Aharoni, Megan Li, Lily Wang, Sandeep Kumar, Norman Casagrande, Jay Hoover, Dalia El Badawy, David Soergel, Denis Vnukov, Matt Miecnikowski, Jiri Simsa, Praveen Kumar, Thibault Sellam, Daniel Vlasic, Samira Daruki, Nir Shabat, John Zhang, Guolong Su, Jiageng Zhang, Jeremiah Liu, Yi Sun, Evan Palmer, Alireza Ghaffarkhah, Xi Xiong, Victor Cotruta, Michael Fink, Lucas Dixon, Ashwin Sreevatsa, Adrian Goedeckemeyer, Alek Dimitriev, Mohsen Jafari, Remi Crocker, Nicholas FitzGerald, Aviral Kumar, Sanjay Ghemawat, Ivan Philips, Frederick Liu, Yannie Liang, Rachel Sterneck, Alena Repina, Marcus Wu, Laura Knight, Marin Georgiev, Hyo Lee, Harry Askham, Abhishek Chakladar, Annie Louis, Carl Crous, Hardie Cate, Dessie Petrova, Michael Quinn, Denese Owusu-Afriyie, Achintya Singhal, Nan Wei, Solomon Kim, Damien Vincent, Milad Nasr, Christopher A. Choquette-Choo, Reiko Tojo, Shawn Lu, Diego de Las Casas, Yuchung Cheng, Tolga Bolukbasi, Katherine Lee, Saaber Fatehi, Rajagopal Ananthanarayanan, Miteyan Patel, Charbel Kaed, Jing Li, Shreyas Rammohan Belle, Zhe Chen, Jaclyn Konzelmann, Siim Põder, Roopal Garg, Vinod Koverkathu, Adam Brown, Chris Dyer, Rosanne Liu, Azade Nova, Jun Xu, Alanna Walton, Alicia Parrish, Mark Epstein, Sara McCarthy, Slav Petrov, Demis Hassabis, Koray Kavukcuoglu, Jeffrey Dean, and Oriol Vinyals. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev,

Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024.

Google. Codegemma: Open code models based on gemma. `https://storage.googleapis.com/deepmind-media/gemma/codegemma_report.pdf`, 2024.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. Olmo: Accelerating the science of language models, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL `https://arxiv.org/abs/2203.15556`.

Iman Hosseini and Brendan Dolan-Gavitt. Beyond the c: Retargetable decompilation using neural machine translation. In *Proceedings 2022 Workshop on Binary Analysis Research*, BAR 2022. Internet Society, 2022. doi: 10.14722/bar.2022.23009. URL `http://dx.doi.org/10.14722/bar.2022.23009`.

Najoung Kim and Tal Linzen. COGS: A compositional generalization challenge based on semantic interpretation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9087–9105, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.731. URL `https://aclanthology.org/2020.emnlp-main.731`.

Najoung Kim, Sebastian Schuster, and Shubham Toshniwal. Code pretraining improves entity tracking abilities of language models, 2024.

Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages, 2020.

Matthias Lindemann, Alexander Koller, and Ivan Titov. Strengthening structural inductive biases by pretraining to perform syntactic transformations, 2024. URL `https://arxiv.org/abs/2407.04543`.

Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, and Daphne Ippolito. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3245–3276, Mexico City, Mexico, June 2024. Association for Computational Linguistics. URL `https://aclanthology.org/2024.naacl-long.179`.

Yingwei Ma, Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, and Shanshan Li. At which training stage does code data help LLMs reasoning? In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=KIPJKST4gw`.

Aaron Mueller, Robert Frank, Tal Linzen, Luheng Wang, and Sebastian Schuster. Coloring the blank slate: Pre-training imparts a hierarchical inductive bias to sequence-to-sequence models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 1352–1368, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.106. URL https://aclanthology.org/2022.findings-acl.106.

Aaron Mueller, Albert Webson, Jackson Petty, and Tal Linzen. In-context learning generalizes, but not always robustly: The case of syntax. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4761–4779, Mexico City, Mexico, June 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.naacl-long.267.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

Isabel Papadimitriou and Dan Jurafsky. Injecting structural hints: Using language models to study inductive biases in language learning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 8402–8413, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.563. URL https://aclanthology.org/2023.findings-emnlp.563.

Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. Examining zero-shot vulnerability repair with large language models, 2022.

Jackson Petty, Sjoerd van Steenkiste, Ishita Dasgupta, Fei Sha, Dan Garrette, and Tal Linzen. The impact of depth on compositional generalization in transformer language models, 2024.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. Improving compositional generalization with latent structure and data augmentation. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4341–4362, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.323. URL https://aclanthology.org/2022.naacl-main.323.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

Yasaman Razeghi, Hamish Ivison, Sameer Singh, and Yanai Elazar. BACKTRACKING MATHEMATICAL REASONING OF LANGUAGE MODELS TO THE PRETRAINING DATA. In *The Second Tiny Papers Track at ICLR 2024*, 2024. URL https://openreview.net/forum?id=otHhLO7GZj.

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

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew Dai, Andrew La, Andrew Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakaş, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartłomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, César Ferri Ramírez, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Chris Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodola, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, Francois Chollet, Frieda Rong, Gaurav

Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Wang, Gonzalo Jaimovitch-López, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Shevlin, Hinrich Schütze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Kocoń, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, Jose Hernandez-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh D. Dhole, Kevin Gimpel, Kevin Omondi, Kory Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros Colón, Luke Metz, Lütfi Kerem Şenel, Maarten Bosma, Maarten Sap, Maartje ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ramírez Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, Mátyás Schubert, Medina Orduna Baitemirova, Melody Arnaud, Melvin McElrath, Michael A. Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michał Swędrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, Mukund Varma T, Nanyun Peng, Nathan A. Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha S. Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter Chang, Peter Eckersley, Phu Mon Htut, Pinyu Hwang, Piotr Miłkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan LeBras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Ruslan Salakhutdinov, Ryan Chi, Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel S. Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima, Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven T. Piantadosi, Stuart M. Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsu Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Ramasesh, Vinay Uday Prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller,

Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. What language model architecture and pretraining objective work best for zero-shot generalization?, 2022.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024a.

Ke Yang, Jiateng Liu, John Wu, Chaoqi Yang, Yi R. Fung, Sha Li, Zixuan Huang, Xu Cao, Xingyao Wang, Yiquan Wang, Heng Ji, and Chengxiang Zhai. If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents, 2024b.

Yuekun Yao and Alexander Koller. Simple and effective data augmentation for compositional generalization, 2024. URL https://arxiv.org/abs/2401.09815.

Xi Ye and Greg Durrett. The unreliability of explanations in few-shot prompting for textual reasoning, 2022.

Xi Ye, Srinivasan Iyer, Asli Celikyilmaz, Ves Stoyanov, Greg Durrett, and Ramakanth Pasunuru. Complementary explanations for effective in-context learning, 2023.

Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, Chen Li, Ziyan Gong, Yifan Yao, Xinjing Huang, Jun Wang, Jianfeng Yu, Qi Guo, Yue Yu, Yan Zhang, Jin Wang, Hengtao Tao, Dasen Yan, Zexuan Yi, Fang Peng, Fangqing Jiang, Han Zhang, Lingfeng Deng, Yehong Zhang, Zhe Lin, Chao Zhang, Shaojie Zhang, Mingyue Guo, Shanzhi Gu, Gaojun Fan, Yaowei Wang, Xuefeng Jin, Qun Liu, and Yonghong Tian. Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation, 2021. URL https://arxiv.org/abs/2104.12369.

Li Zhang, Liam Dugan, Hainiu Xu, and Chris Callison-Burch. Exploring the curious case of code prompts, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023.

A Regression Coefficients

Dataset	**Gen. type**	**Setting**	**Baseline**	$\hat{\beta}$	$\hat{\alpha}$	R^2
COGS	Lexical	Competitive	—	0.022	0.932	0.776
COGS	Lexical	Additive	False	0.030	0.935	0.792
COGS	Lexical	Additive	True	0.024	0.935	0.869
COGS	Structural	Competitive	—	0.009	0.009	0.816
COGS	Structural	Additive	False	0.007	0.010	0.960
COGS	Structural	Additive	True	0.000	0.007	1.000
COGS-vf	Lexical	Competitive	—	0.014	0.970	0.877
COGS-vf	Lexical	Additive	False	0.025	0.971	0.816
COGS-vf	Lexical	Additive	True	0.024	0.970	0.851
COGS-vf	Structural	Competitive	—	0.147	0.186	0.413
COGS-vf	Structural	Additive	False	0.165	0.162	0.692
COGS-vf	Structural	Additive	True	−0.048	0.170	0.961

Table 3: Coefficients of linear regressions $\hat{y} = \hat{\beta}x + \hat{\alpha}$ predicting generalization accuracy by code mixture on COGS and COGS-vf.

Dataset	**Gen. type**	**Setting**	**Baseline**	$\hat{\beta}$	$\hat{\alpha}$	R^2
COGS	Lexical	Competitive	—	0.006	0.993	0.883
COGS	Lexical	Additive	False	0.004	0.995	0.988
COGS	Lexical	Additive	True	−0.005	0.996	0.969
COGS	Structural	Competitive	—	−0.012	0.982	0.980
COGS	Structural	Additive	False	−0.098	0.986	0.718
COGS	Structural	Additive	True	−0.067	0.976	0.860
COGS-vf	Lexical	Competitive	—	0.000	1.000	0.999
COGS-vf	Lexical	Additive	False	0.000	1.000	0.948
COGS-vf	Lexical	Additive	True	0.024	1.000	0.847
COGS-vf	Structural	Competitive	—	−0.033	0.653	0.978
COGS-vf	Structural	Additive	False	0.049	0.632	0.987
COGS-vf	Structural	Additive	True	−0.132	0.658	0.914

Table 4: Coefficients of linear regressions $\hat{y} = \hat{\beta}x + \hat{\alpha}$ predicting generalization well-formedness by code mixture on COGS and COGS-vf.

Dataset	**Setting**	**Baseline**	$\hat{\beta}$	$\hat{\alpha}$	R^2
English Passivization	Competitive	—	−0.416	0.894	0.718
English Passivization	Additive	False	−0.263	0.775	0.966
English Passivization	Additive	True	−0.193	0.913	0.973

Table 5: Coefficients of linear regressions $\hat{y} = \hat{\beta}x + \hat{\alpha}$ predicting generalization accuracy by code mixture on English Passivization.

Dataset	**# of Digits**	**Setting**	$\hat{\beta}$	$\hat{\alpha}$	R^2
BB Arithmetic JSON	1	Competitive	0.062	0.381	0.906
BB Arithmetic JSON	1	Additive	0.172	0.373	0.780
BB Arithmetic JSON	2	Competitive	0.095	0.203	0.901
BB Arithmetic JSON	2	Additive	0.265	0.169	0.654
BB Arithmetic JSON	3	Competitive	0.124	0.204	0.891
BB Arithmetic JSON	3	Additive	0.330	0.171	0.706
BB Arithmetic JSON	4	Competitive	0.135	0.221	0.898
BB Arithmetic JSON	4	Additive	0.369	0.168	0.710
BB Arithmetic JSON	5	Competitive	0.121	0.248	0.925
BB Arithmetic JSON	5	Additive	0.397	0.190	0.706

Table 6: Coefficients of linear regressions $\hat{y} = \hat{\beta}x + \hat{\alpha}$ predicting generalization accuracy by code mixture on BB Arithmetic JSON.

Dataset	**Setting**	$\hat{\beta}$	$\hat{\alpha}$	R^2
BB Common Morpheme JSON	Competitive	−0.093	0.364	0.804
BB Common Morpheme JSON	Additive	−0.049	0.349	0.968
BB Fantasy Reasoning JSON	Competitive	−0.047	0.552	0.946
BB Fantasy Reasoning JSON	Additive	−0.062	0.564	0.955
BB General Knowledge JSON	Competitive	−0.084	0.246	0.749
BB General Knowledge JSON	Additive	−0.097	0.240	0.759
BB Implicatures JSON	Competitive	−0.013	0.520	0.971
BB Implicatures JSON	Additive	0.006	0.512	0.997

Table 7: Coefficients of linear regressions $\hat{y} = \hat{\beta}x + \hat{\alpha}$ predicting generalization accuracy by code mixture on BB Common Morpheme, Fantasy Reasoning, General Knowledge, and Implicatures JSON.