PROGRAMMING BY BACKPROP: LEARNING BEHAVIOUR FROM SYMBOLIC DESCRIPTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are typically trained to acquire behaviours from demonstrations or experience, yet much of their training data consists of symbolic descriptions: instructions, rules, and strategies that specify procedures without examples. We investigate whether LLMs can learn to execute such behaviours directly from their abstract description, a process we term *Programming by Backprop* (PBB). We study this phenomenon in two domains: first, using source code as a canonical form of procedural description by comparing models finetuned on algorithms versus execution examples; and second, extending beyond code to abstract grammar rules, testing whether models learn to generate compliant text. Our findings show that PBB can be elicited through targeted finetuning, demonstrating that LLMs can acquire new behaviours from symbolic descriptions, albeit not yet with full reliability. Once elicited, PBB enables models to internalise reusable procedural abstractions — generalising across inputs, executing procedures implicitly in a forward pass, and benefiting further from chain-of-thought reasoning. These results position PBB as a distinct pathway through which LLMs acquire behavioural skills from symbolic descriptions, with implications for both more efficient capability acquisition and aligning models through formal specifications rather than demonstrations alone.

1 INTRODUCTION

When training a large language model (LLM) to learn a desired behaviour, the most common approaches involve imitation learning via sequence modelling on demonstrations (i.e., supervised finetuning, SFT), or repeated trial-and-error (i.e., reinforcement learning, RL). Yet much of what LLMs are exposed to during pretraining is not demonstrations but descriptions of procedures: instructions, rules, or strategies that specify how to act without showing concrete input-output examples. For humans, the ability to learn from such descriptions is a core component of skill acquisition, enabling us to augment practice with instruction manuals, strategy guides, or general discourse. Effectively leveraging training data at a level of abstraction above explicit demonstration is a key contributor to our remarkable sample efficiency (Dienes & Perner, 1999). For example, learning chess is accelerated not only by playing games, but also by studying written accounts of opening principles, tactics, and endgame strategies. If LLMs can similarly learn to execute behaviours from abstract descriptions, this would represent a crucial pathway for more efficient and generalisable learning.

Indeed, prior work has found that LLMs can emulate simple character descriptions (Berglund et al., 2023), extract general strategies from demonstrations (Betley et al., 2025), and that procedural descriptions in pretraining data influence related skills (Ruis et al., 2025). Building on these observations, we formalise the specific process of acquiring behavioural competence from symbolic descriptions as Programming by Backprop (PBB). PBB captures the concrete way in which learning from descriptions can arise through gradient-based training — specifically, autoregressive language modelling. By internalising abstract, input-general definitions of behaviour and later applying them during inference, PBB offers both a practical tool, teaching models new behaviours through symbolic descriptions alone, and a conceptual lens for understanding how LLMs generalise beyond surface-level pattern matching.

Two motivations drive the study of PBB. First, it offers a scalable approach to capability building: instead of requiring large datasets of demonstrations, models could be equipped with procedures

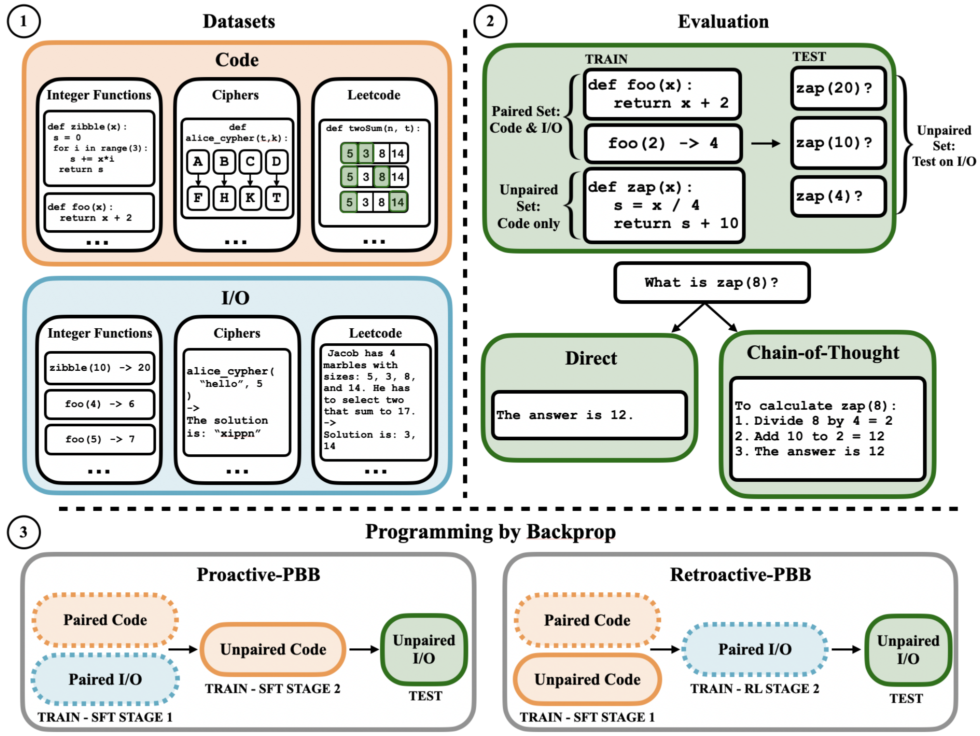

Figure 1: Illustration of *Programming by Backprop* (PBB) — the learning of behaviours from symbolic descriptions — for the code execution tasks used in our experiments. PBB emerges in two regimes: **Proactive PBB**, where models learn to interpret paired procedures before being exposed to unpaired ones, and **Retroactive PBB**, where initial exposure to all procedures is followed by activation through paired examples. Models trained under PBB internalise executable abstractions, allowing them to execute novel programs directly in a forward pass, and to improve further when using chain-of-thought reasoning.

expressed succinctly in symbolic form. Prior work has shown that LLMs benefit from having symbolically described procedures like code snippets in-context when reasoning (Chen et al., 2023), naturally motivating a study of whether the same is true for procedures described in training data. This opens the door to training methods that are more data-efficient, interpretable, and compositional. Second, it carries significant safety implications. If LLMs can acquire behavioural competence from symbolic specifications, then carefully chosen formal descriptions, such as constitutional principles or alignment objectives, may serve as a foundation for shaping model behaviour more robustly than demonstrations or experience alone.

However, this potential raises fundamental questions. Can LLMs already learn to execute procedures they have only “read about” during downstream finetuning? If not, can we elicit this ability with specific approaches to finetuning? Furthermore, what are the properties of such a learning mechanism? Does the language of the description matter (e.g., code versus natural language)? Can models combine independently learned procedures to solve new, composite problems, and does explicit reasoning, like chain-of-thought, help them apply this learned knowledge? Finally, does learning from an abstract rule yield more robust and generalisable behaviour compared to learning from a narrow or imbalanced set of examples?

In this paper, we formalise PBB as a meta-learning problem and provide a holistic, empirical investigation into these questions. We first study code as a canonical form of symbolic description, comparing models finetuned on algorithm source code alone to those trained with input-output (I/O) examples and those trained on a combination of the two. We then generalise to a non-coding domain, testing whether models trained on abstract grammar rules learn to generate compliant text. *Across these domains, we evaluate whether models can generalise from symbolic descriptions to correct execution on inputs.*

In our experiments, we find that PBB does not emerge reliably out-of-the-box: pretrained models show little evidence of learning to execute behaviours after finetuning on descriptions alone. However, PBB is possible following targeted finetuning designed to elicit it. Training on a mixture of descriptions paired with demonstrations enables models to generalise to unpaired descriptions, acquiring the ability to execute procedures they have never seen demonstrated. This reveals two key insights. First, LLMs are in principle capable of learning behaviours from symbolic descriptions, a capacity that, once elicited, generalises across inputs and domains. Second, the fact that standard post-training pipelines fail to reliably produce PBB highlights an important limitation in current approaches to training LLMs. This is in keeping with prior work demonstrating key limitations in the ability to generalise from knowledge in finetuning data (Berglund et al., 2024; Allen-Zhu & Li, 2025). Addressing this gap may be critical both for improving efficiency in capability acquisition and for aligning models through formal specification in addition to imitation or experience.

Our main contributions are as follows:

- We introduce and formalise Programming by Backprop (PBB), the process by which large language models can acquire new behavioural skills by training on abstract, symbolic descriptions of procedures rather than on input-output demonstrations.
- We conduct an investigation of PBB across two distinct domains: algorithmic execution from source code and text generation from abstract grammar rules.
- We demonstrate that PBB does not emerge reliably from pretraining alone but can be elicited through targeted finetuning strategies.
- We discover several key properties of PBB that inform future work, showing that: (1) its efficacy is highly dependent on the description language, with formal symbolic systems yielding substantially better results than natural language; (2) while learned procedures can be executed implicitly in a forward pass, chain-of-thought reasoning significantly enhances performance on algorithmic tasks; (3) models exhibit a nascent capacity for compositionality, retrieving and combining independently learned subroutines; and (4) PBB imparts greater robustness to input variations compared to skills learned from narrow or imbalanced demonstrations.

2 Related Work

Our work is related to the literature on learning to execute programs from demonstrations or symbolic inputs. Zaremba & Sutskever (2015) show that recurrent networks can learn to map program text to outputs given execution supervision. Tian et al. (2019) extend this direction by inferring symbolic “shape programs” form perceptual data and executing them for 3D reconstruction — conceptually related to PBB in that both investigate how symbolic structure can drive executable behaviour, though their goal is program *inference* from raw data, whereas ours is execution from symbolic descriptions *within training data*. Yan et al. (2020) study transformer-based execution engines trained on explicit I/O demonstrations of algorithmic subroutines, and Rahaman et al. (2021) introduce Neural Interpreters that achieve modular execution through architectural design. In contrast, *PBB explores whether standard LLMs can learn to execute procedures where only their symbolic description is trained on*, after learning a general mapping between symbolic descriptions and execution from other, possibly OOD, procedures.

A second line of work studies how code pretraining changes model behaviour. Exposure to source code has been shown to improve reasoning and problem-solving in natural language (Aryabumi et al., 2024; Petty et al., 2024), suggesting that the regularity and explicit structure of code (e.g., variables, control flow, compositional syntax) supports reasoning skills that transfer to natural language. Beyond raw source code, training on synthetic procedural traces, such as edit sequences, improves code synthesis by forcing models to represent intermediate steps in a transformation (Piterbarg et al., 2025). Other approaches explicitly couple language models with an external interpreter, showing that natural language reasoning can be grounded in code execution to improve accuracy (Li et al., 2024). Together, this work demonstrates that symbolic supervision can scaffold logical reasoning in ways natural language alone does not.

Recent research has investigated the role of procedural text in pretraining. Ruis et al. (2025) show that exposure to input-general procedures in pretraining data, such as code functions, strongly influences models' ability to solve related input-specific reasoning problems, for example when a math question reduces to executing the function. This motivates our in-depth study of this phenomenon through controlled finetuning experiments. Similarly, work on in-context compositionality finds that models can synthesise new behaviours from symbolic instructions given in-context (Chen et al., 2024). *Our work investigates a related capability, but for procedures that are internalised in the model's weights, rather than provided in-context.*

Internalising behaviour that generalises is not guaranteed. Parallel work on arithmetic reasoning finds that LLMs often rely on surface heuristics rather than executing explicit algorithms (Nikankin et al., 2025), though structured prompting can elicit more systematic behaviour (Chen et al., 2023). Mechanistic analyses further suggest that the autoregressive training objective shapes which kinds of procedures can be internalised, constraining both successes and failure modes (McCoy et al., 2024; Wang et al., 2024). Allen-Zhu & Li (2025) demonstrate that while LLMs excel in knowledge retrieval, they struggle in classification and comparison tasks when chain-of-thought (CoT) is not used during training and inference, offering a potential explanation for LLMs' "knowing-doing gap" (Paglieri et al., 2024). Lampinen et al. (2025) investigate how LLMs generalise differently from knowledge in-context versus in their training data. *Our results contribute to this discussion by showing that models can learn to execute procedures that are described in their training data.*

Finally, our work connects to studies on how LLMs generalise from their training data in sophisticated ways. This includes findings that models can perform out-of-context reasoning (OOCR) (Berglund et al., 2023; Betley et al., 2025), exhibit forms of implicit meta-learning (Krasheninnikov et al., 2024), acquire latent multi-hop reasoning abilities (Yang et al., 2024), and infer latent structure from training data (Treutlein et al., 2024). *We propose PBB as an under-investigated form of OOCR, where the model learns to treat symbolic descriptions seen during training as executable procedures.* This process provides a concrete pathway for models to acquire generalisable behavioural skills directly from abstract descriptions, rather than from demonstrations alone.

3 Programming by Backprop

We define Programming by Backprop (PBB) as the process by which a sequence model $\mathcal{M}_\theta$, with parameters θ, learns to execute a procedure $f : \mathcal{X} \to \mathcal{Y}$ by training on its symbolic description s_f rather than on a dataset input-output execution examples $(x_i, f(x_i))$.

We hypothesise that a model trained on symbolic descriptions and executions for a subset of procedures F_{paired} can, through the standard autoregressive training objective, learn to execute other procedures F_{unpaired} from their symbolic descriptions alone.

Let F be a universe of procedures with symbolic descriptions S_F (e.g., Python functions, grammar rules). Crucially, this universe can span multiple domains and may include novel compositions of procedures, allowing us to test for more complex forms of generalisation. We partition this universe into two disjoint sets: a **paired set** F_{paired} and an **unpaired set** F_{unpaired}. For each procedure $f \in F_{\text{paired}}$, we have access to its symbolic description s_f and a corresponding dataset of N execution examples $\mathcal{D}_f = \{(x_i, f(x_i))\}_i^N$. For procedures in F_{unpaired}, we possess only their symbolic descriptions. The test task is executing procedures from F_{unpaired} on a set of inputs.

This setup frames PBB as a **meta-learning problem**: the model learns the skill of interpreting symbolic descriptions from the paired set (F_{paired} and its associated examples) and then generalises this skill to execute the procedures from the unpaired set (F_{unpaired}), which it has only seen as descriptions. We propose and evaluate two distinct finetuning strategies designed to elicit this capability.

Proactive PBB: Priming for Interpretation

The first approach, which we call **Proactive PBB**, is a two-stage supervised finetuning (SFT) pipeline designed to first teach the model the general correspondence between descriptions and execution, and then expose it to new, unpaired procedures.

1. **Stage 1 (Meta-Learning):** A pretrained base model $\mathcal{M}_{\text{base}}$ is finetuned on the paired procedures' symbolic descriptions $S_{F_{\text{paired}}}$ and their corresponding execution examples $\mathcal{D}^{\text{train}}_{\text{paired}}$.

This stage explicitly trains the model to associate a symbolic description with its behaviour, priming it to interpret new procedures it encounters.

2. **Stage 2 (Acquisition):** The resulting model, $\mathcal{M}_{\text{stage-1}}$ is further finetuned on the symbolic descriptions $S_{F_{\text{unpaired}}}$. *During this stage, the model is expected to implicitly internalise these new procedures by leveraging the interprative skill learned in Stage 1.*

$\mathcal{M}_{\text{stage-2}}$ is then evaluated on a test set of execution tasks $\mathcal{D}^{\text{test}}_{\text{unpaired}}$ for the procedures F_{unpaired}.

Retroactive PBB: Activating Latent Procedures

Our second approach, **Retroactive PBB**, reverses the training sequence. It first exposes the model to all symbolic procedures and then "activates" the ability to execute them by finetuning on a subset of paired examples.

1. **Stage 1 (Exposure):** The base model $\mathcal{M}_{\text{base}}$ is first finetuned on the full set of symbolic descriptions S_F. In this stage, the model learns representations of all procedures without any explicit signal that they are executable.
2. **Stage 2 (Activation):** The resulting model $\mathcal{M}_{\text{stage-1}}$ is then finetuned on the execution examples $\mathcal{D}^{\text{train}}_{\text{paired}}$ corresponding to the paired set F_{paired}. *The hypothesis is that this will retroactively teach the model to treat all procedures that it learned in Stage 1 as executable, including those from* F_{unpaired}.

4 Experimental Setup

To investigate Programming by Backprop (PBB), we conduct experiments across two distinct domains: algorithmic reasoning in Python and formal grammar generation. Our setup is designed to test whether models can internalise and execute procedures from symbolic descriptions they have not seen demonstrated.

4.1 Datasets and Tasks

We create three synthetic datasets to provide a controlled environment for studying PBB. We additionally experiment with a real-world coding dataset and corresponding execution task. For each dataset, we define a universe of procedures F, which is partitioned into a paired and unpaired set.

Algorithmic Reasoning. These tasks test whether models can learn to execute Python functions from their source code.

- **Random Arithmetic:** This dataset contains 1,000 unique Python functions that map integers to integers. The functions are synthetically generated by composing basic control flow (`for` loops, `if` / `else` conditionals) and arithmetic operators (`+`, `-`, `*`, `//`, `%`, `>`, `<`, `exp`, `abs`). This allows us to control for procedural complexity (i.e., the number of operations). For our main experiments, we use 100 functions for F_{paired} and 100 for F_{unpaired}.
- **Leetcode:** This dataset consists of 702 real-world algorithmic problems and their Python solutions, sourced from the competitive programming platform. This tests PBB on more complex and naturalistic procedures. We use 500 problems for F_{paired} and 100 for F_{unpaired}.
- **Ciphers:** To test generalisation to novel, OOD procedures, we create three custom ciphers (Alice, Bob, Kevin) that are variations of standard ciphers (Caesar, Atbash, Vigenère). We assume these novel ciphers are absent from the model's pretraining data, allowing for controlled experimentation. The same 500 Leetcode problems are used for F_{paired} and the three custom ciphers form F_{unpaired}. For this task, we also generate demonstrations from an imbalanced distribution (Appendix E), reflecting the greater occurrence of examples with specific shift values in pretraining data (McCoy et al., 2024). This allows us to compare learning ciphers via PBB to learning from imbalanced demonstration data, thus revealing whether PBB can provide a data-efficient way of overcoming biases from pretraining.

For all tasks, input-output examples are framed as word problems. We generate ground-truth solutions by executing the corresponding Python code. To test the benefits of intermediate reasoning, we also generate chain-of-thought (CoT) solutions for each problem using `GPT-4o` in a post-rationalisation step (Zelikman et al., 2022).

Example Context-Free Grammar and Generations

'VenShi' Grammar (terminals omitted for brevity):

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \mid \text{VP NP} \\
\text{VP} &\rightarrow \text{V NP} \mid \text{V Adv} \mid \text{V} \\
\text{NP} &\rightarrow \text{Det Adj N} \mid \text{N}
\end{aligned}
$$

Sample 'VenShi' generations:

```
-> sleeps car the ancient tree
-> the blue wizard shines dog
```

Formal Grammar Generation. To test PBB beyond code, we construct a procedurally generated suite of artificial grammars that define syntactic constraints on sentence formation. Concretely, we generate a universe of 200 unique context-free grammars (CFGs). Each grammar is produced by sampling from a compact, interpretable parameter space that controls typological properties (word order families such as SVO, SOV, VSO, etc.), modifier placement (adjectives pre/post-nominal, determiners pre/post-nominal, adverbs pre/post-verbal), and optional structural features. Each grammar is represented symbolically as a small set of production rules (nonterminals and productions). We use a single, shared lexicon when sampling example strings.

We sample deriviation trees from each CFG by repeatedly expanding nonterminals according to that grammar until a depth cutoff. A sampled derivation tree is used to produce a sentence by instantiating terminals from the shared lexicon. From the 200 grammars, we designate 100 as F_{paired} (grammars paired with example sentences) and 100 as F_{unpaired} (grammars for which symbolic specification is shown during training, but not example sentences). We evaluate model generations with a strict, grammar-based validity test: a candidate sentence is accepted if it can be parsed by the requested CFG (we use a chart/Earley parser). Accuracy on a grammar is thus the fraction of model outputs that produce at least one valid parse under that grammar. For this task, we do not employ CoT during training or evaluation.

4.2 Training Details

We use instruction-tuned `Llama-3` models (1B, 3B, 8B) (Dubey et al., 2024) as our primary set of base models and conduct additional experiments with `GPT-4o` (OpenAI et al., 2024) via the OpenAI finetuning API to investigate PBB in a large frontier model. We also repeat two core experiments with instruction-tuned `Qwen-3` models (4B, 8B) (Qwen et al., 2025) to see if trends are consistent across model families (Appendix A). All training runs use a single epoch. For RL runs, we use GRPO (Shao et al., 2024) with a group size of 8 and no KL regularisation ($\beta = 0$). A positive reward ($+1$) is given if the final answer is properly formatted and correct; a neutral reward (0) is given if the answer is properly formatted but incorrect; a negative reward (-1) is given if the model fails to produce a final answer in the required format. For SFT and RL runs, the training batch size is set to 32 and we use a constant learning rate of 1×10^{-6}. We use a sampling temperature $t = 0.8$ during RL training and for evaluation. All evaluations are averaged over 16 samples and 95% confidence intervals are reported.

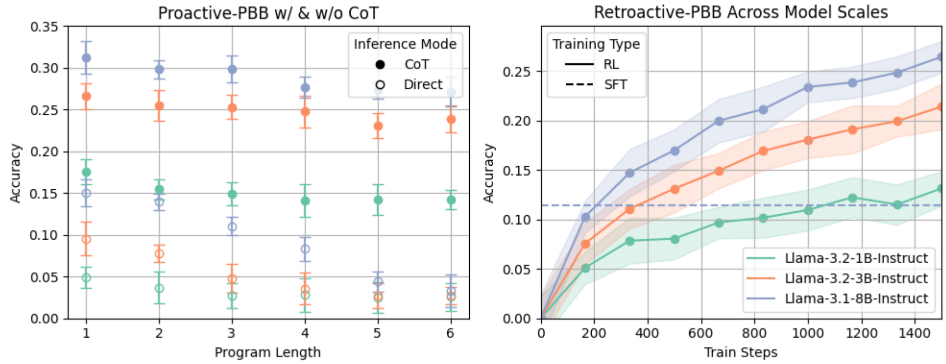

Figure 2: **Left**: Accuracy following proactive PBB on executing unpaired random arithmetic programs of different lengths. **Right**: Accuracy following retroactive PBB.

5 Results

5.1 Random Arithmetic

We first test whether models of different scales can use proactive PBB to execute programs for which no input–output examples are available. Results are shown in Figure 2 (left). We find that eliciting PBB is highly scale-dependent: 1B models show little to no ability to execute unseen programs, while 3B and 8B models display measurable improvements, particularly when using chain-of-thought (CoT). Explicit reasoning allows these models to handle longer programs with more operations. Implicit execution in a forward pass is significantly less reliable and performance more notably declines as program complexity increases. However, the fact that an 8B model can recall and execute programs of up to 3 successive operations, knowledge of which comes exclusively from having encountered them as code during finetuning, with an accuracy greater than 10% is promising.

The two-stage proactive PBB pipeline also has clear benefits over a single mixed finetuning stage (Appendix C). Because the initial meta-learning in stage 1 teaches a general code-execution mapping, the same program source code needs to appear fewer times in stage 2 for the model to succeed. This shows that a curriculum which first builds the interpretive skill makes learning from symbolic descriptions more efficient compared to combining the meta-learning and acquisition stages.

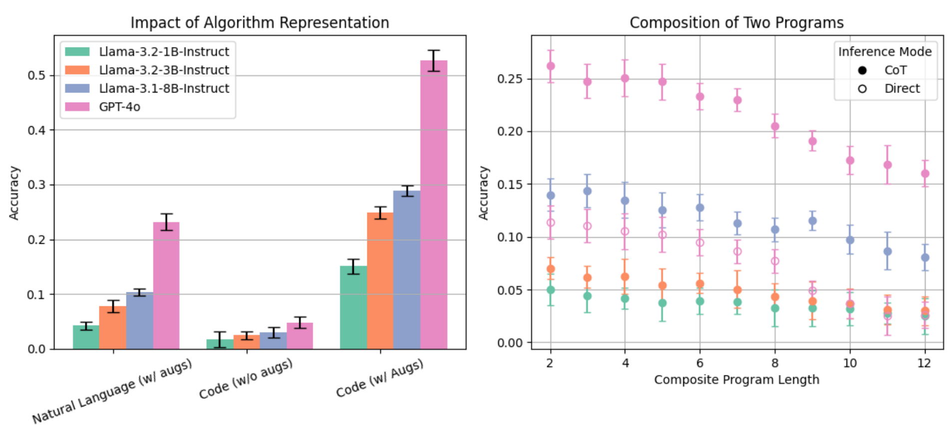

Figure 3: **Left**: Accuracy following proactive PBB on executing unpaired random arithmetic programs represented as natural language or code. **Right**: Accuracy following proactive PBB for compositions of two programs that have been trained on independently.

Next, we examine how the representation of procedures affects proactive PBB. Substituting source code for semantically equivalent natural language descriptions of functions markedly reduces performance (Figure 3, left). This suggests that the formal structure of code provides scaffolding that LLMs can more readily internalise as algorithmic abstractions. Prompt/response preamble augmentations further improve performance by diversifying training examples, echoing findings from

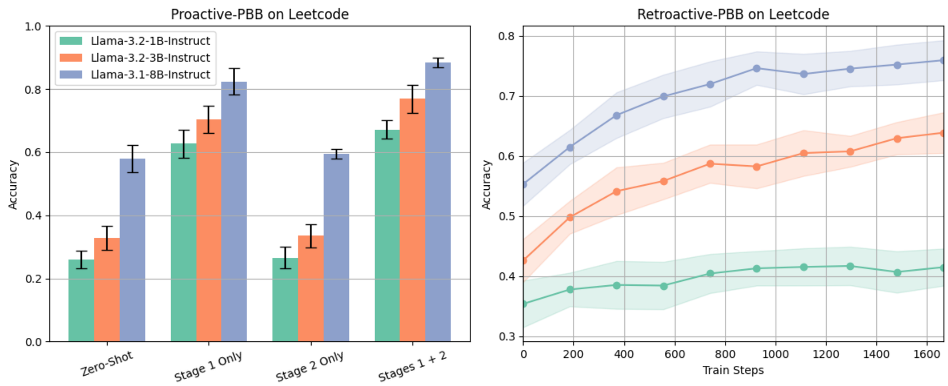

Figure 4: **Left**: Accuracy of following each stage of proactive PBB on executing unpaired Leetcode programs. **Right**: Accuracy following retroactive PBB.

OOCR (Berglund et al., 2023), though we find that augmenting surrounding text (i.e., prompt and answer preamble) is sufficient; no modification of the source code itself is required.

We then probe compositional generalisation by holding out composite functions (i.e., definitions built from the composition of two unpaired random arithmetic functions). Here, `Llama-3` models fail to perform implicit execution. However, they sometimes succeed with CoT reasoning (Figure 3, right). `GPT-4o` achieves partial success at composition even without CoT, demonstrating retrieval and sequential execution of independently learned subroutines within a single forward pass. Performance nevertheless declines with increasing program length, consistent with cumulative recall and/or execution error likelihood across successive operations.

Finally, we evaluate retroactive PBB. Results in Figure 2 (right) show that elicitation strength depends on both model scale and training algorithm. Notably, reinforcement learning (RL) in stage 2 substantially outperforms supervised finetuning (SFT): even a 1B model with RL surpasses the final performance of an 8B model trained with SFT. Under SFT, models tended to memorise the surface form of demonstrations rather than linking them back to the symbolic descriptions learned in stage 1. RL with verifiable rewards more effectively teaches the model to associate the symbolic representations with their execution, thereby "activating" executable behaviour. In contrast, SFT suffices for proactive PBB because stage 1 jointly exposes symbolic descriptions and demonstrations, allowing the correspondence to be learned directly. This indicates that online learning plays a crucial role in activating latent procedure representations, consistent with prior work on the benefits of RL for generalisation (Held & Hein, 1963; Ostrovski et al., 2021; Kirk et al., 2024; Chu et al., 2025). Appendix D further isolates the contributions of negative samples versus on-policy data by using DPO, showing that both are beneficial but GRPO remains the most effective.

5.2 LEETCODE

We test whether PBB extends to more naturalistic, real-world programs in Figure 4. For proactive PBB, stage 1 alone (i.e., training on paired descriptions and input-output examples) yields strong gains on unpaired programs, likely reflecting the presence of code similar, or equivalent, to the unpaired Leetcode programs in the base model's training data. Prior knowledge of these programs means that improved execution ability could generalise to them in a manner similar to stage 2 of retroactive PBB. However, proactive PBB's second stage *further* improves performance by exposing the model to the held-out program source code, confirming that PBB can be elicited even in domains with substantial prior familiarity, improving a model's ability to execute previously encountered procedures. For retroactive PBB, we again see that larger models benefit more during the RL activation stage: while

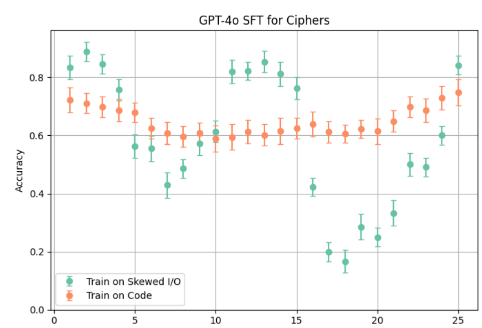

Figure 5: Accuracy of `GPT-4o` when encrypting text with ciphers trained on only as code, or when trained on as demonstrated execution traces with unevenly distributed shifts.

all models begin with nonzero zero-shot performance, only the 3B and 8B models show strong gains from stage 2 of finetuning.

5.3 CIPHERS

We investigate transfer to novel, synthetic ciphers exclusively with `GPT-4o` because the open-source models we consider struggle with accurate encryption even if the ciphers are provided in-context. `GPT-4o`'s zero-shot accuracy, when the custom ciphers are only referenced by name in-context, is at chance, confirming that these procedures are absent from pretraining. Remarkably, Figure 5 shows that after stage 1 of proactive PBB on Leetcode, further finetuning on cipher source code alone is sufficient for the model to learn and apply these ciphers with reasonable accuracy, demonstrating that PBB can support the acquisition of OOD behaviours. Compared to training on demonstrations with imbalanced shifts, PBB yields more uniform accuracy, demonstrating the benefits of learning from descriptions of procedures for generalisation. As functional source code abstracts away the values of input parameters, such as shift in this case, PBB enables the model to learn an executable, input-general representation of the procedure, leading to robustness across inputs and suggesting a path towards overcoming the "embers of autoregression" (McCoy et al., 2024).

5.4 FORMAL GRAMMAR GENERATION

To test whether PBB can be applied beyond code, we evaluate its ability to acquire behaviours from abstract grammar rules. The task is to generate sentences that comply with a given context-free grammar (CFG) that the model has not seen demonstrated, but that has appeared in its symbolic form in training data. For this task, we only train and evaluate without CoT, meaning that the model must parametrically retrieve the rules of the grammar and immediately generate a valid sentence. Results in Figure 6 show that PBB successfully extends to this non-coding symbolic domain, but only when the full proactive PBB pipeline is used. Zero-shot performance is negligible, as models have never seen these synthetic grammars before. Crucially, neither training phase is indpendently effective: training on paired grammars alone or on unpaired grammar rules alone provides minimal benefit.

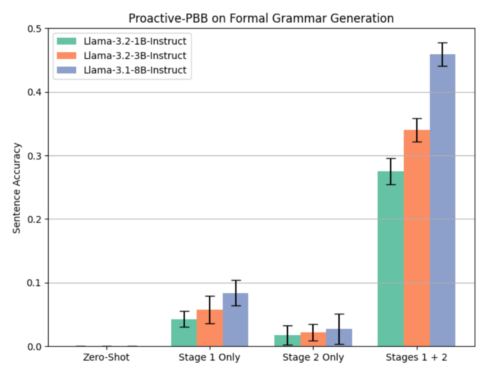

Figure 6: Compliance to unpaired grammars following each stage of proactive PBB.

However, the complete two-stage pipeline yields a dramatic increase in sentence accuracy. This demonstrates that the meta-learning phase (Stage 1) successfully teaches the model a general skill of interpreting symbolic rules, which it then leverages in Stage 2 to internalise and execute the novel, unpaired grammars. Consistent with our findings in algorithmic tasks, this capability is highly scale-dependent, with the 8B model substantially outperforming the smaller models.

6 CONCLUSION

In this paper, we show that LLMs can learn to execute novel procedures from their symbolic descriptions, a capability we term Programming by Backprop (PBB). We demonstrate this by eliciting the generalisation through targeted finetuning pipelines. We introduce two such pipelines that successful induce this skill. Proactive PBB uses a two-stage curriculum that first uses descriptions paired with demonstrations to meta-learn a general interpretation skill before acquiring new, unpaired procedures from descriptions alone. In contrast, retroactive PBB first exposes a model to all symbolic descriptions and then "activates" its ability to execute these procedures, a process we found is substantially more effective when driven by RL than SFT. Across multiple domains — from synthetic arithmetic and out-of-distribution ciphers to formal grammars — our results reveal a consistent set of principles governing PBB. First, the capability is highly scale-dependent, with larger models proving significantly more adept at internalising and executing symbolic rules. Second, representation matters; source code provides a far more effective learning signal than semantically equivalent natural language, suggesting that its syntax acts as a crucial scaffold for building algorithmic abstractions. Third, PBB on algorithmic tasks benefits considerably from explicit reasoning, indicating that while models can internalise procedures for implicit execution in their forward pass, using the internalised procedure as a guide for explicit computation is more effective.

The implications of PBB are significant. For capability acquisition, it points towards a more data-efficient and interpretable learning paradigm, where new skills can be imparted through concise, formal descriptions rather than extensive demonstrations. PBB also offers a promising new avenue for alignment; our results with formal grammars show that models can internalise abstract rules governing their output. While performance can be considerably improved, particularly on complex compositional tasks, this work establishes PBB as a distinct and viable learning mechanism within LLMs. Future work should focus on integrating these principles into pretraining and exploring other "code-like" formalisms for alignment. Ultimately, PBB reframes our approach to teaching models, suggesting a future where learning is driven as much by abstract instruction as by concrete example, moving us closer to models that don't just mimic patterns, but internalise and execute explicit principles.

7 Reproducibility Statement

We are open-sourcing all code and datasets needed to reproduce our experiments at `https://anonymous.4open.science/r/Programming-by-Backprop`. This includes data generation scripts and training code.

References

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.2, knowledge manipulation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=oDbiL9CLoS.

Viraat Aryabumi, Yixuan Su, Raymond Ma, Adrien Morisot, Ivan Zhang, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. To code, or not to code? exploring impact of code in pre-training, 2024. URL https://arxiv.org/abs/2408.10914.

Lukas Berglund, Asa Cooper Stickland, Mikita Balesni, Maximilian Kaufmann, Meg Tong, Tomasz Korbak, Daniel Kokotajlo, and Owain Evans. Taken out of context: On measuring situational awareness in llms. *CoRR*, abs/2309.00667, 2023. URL https://doi.org/10.48550/arXiv.2309.00667.

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a". In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=GPKTIktA0k.

Jan Betley, Xuchan Bao, Martín Soto, Anna Sztyber-Betley, James Chua, and Owain Evans. Tell me about yourself: LLMs are aware of their learned behaviors. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=IjQ2Jtemzy.

Jiaao Chen, Xiaoman Pan, Dian Yu, Kaiqiang Song, Xiaoyang Wang, Dong Yu, and Jianshu Chen. Skills-in-context: Unlocking compositionality in large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pp. 13838–13890. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.findings-emnlp.812.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on*

Machine Learning Research, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. *CoRR*, abs/2501.17161, 2025. URL https://doi.org/10.48550/arXiv.2501.17161.

Zoltan Dienes and Josef Perner. A theory of implicit and explicit knowledge. *Behavioral and Brain Sciences*, 22(5):735–808, 1999.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://doi.org/10.48550/arXiv.2407.21783.

Richard Held and Alan Hein. Movement-produced stimulation in the development of visually guided behavior. *Journal of comparative and physiological psychology*, 56:872–6, 10 1963. doi: 10.1037/h0040546.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of RLHF on LLM generalisation and diversity. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024. URL https://openreview.net/forum?id=PXD3FAVHJT.

Dmitrii Krasheninnikov, Egor Krasheninnikov, Bruno Kacper Mlodozeniec, Tegan Maharaj, and David Krueger. Implicit meta-learning may lead language models to trust more reliable sources. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Fzp1DRzCIN.

Andrew K. Lampinen, Arslan Chaudhry, Stephanie C.Y. Chan, Cody Wild, Diane Wan, Alex Ku, Jörg Bornschein, Razvan Pascanu, Murray Shanahan, and James L. McClelland. On the generalization of language models from in-context learning and finetuning: a controlled study, 2025. URL https://arxiv.org/abs/2505.00661.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=vKtomqlSxm.

R. Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D. Hardy, and Thomas L. Griffiths. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41):e2322420121, 2024. doi:

10.1073/pnas.2322420121. URL https://www.pnas.org/doi/abs/10.1073/pnas.2322420121.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=O9YTt26r2P.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe

Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=nPHA8fGicZk.

Davide Paglieri, Bartlomiej Cupial, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Lukasz Kucinski, Lerrel Pinto, Rob Fergus, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. BALROG: benchmarking agentic LLM and VLM reasoning on games, 2024. URL https://doi.org/10.48550/arXiv.2411.13543.

Jackson Petty, Sjoerd van Steenkiste, and Tal Linzen. How does code pretraining affect language model task performance?, 2024. URL https://doi.org/10.48550/arXiv.2409.04556.

Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. Training language models on synthetic edit sequences improves code synthesis. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AqfUa08PCH.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Nasim Rahaman, Muhammad Waleed Gondal, Shruti Joshi, Peter V. Gehler, Yoshua Bengio, Francesco Locatello, and Bernhard Schölkopf. Dynamic inference with neural interpreters. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 10985–10998, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5b4e9aa703d0bfa11041debaa2d1b633-Abstract.html.

Laura Ruis, Maximilian Mozes, Juhan Bae, Siddhartha Rao Kamalakara, Dwaraknath Gnaneshwar, Acyr Locatelli, Robert Kirk, Tim Rocktäschel, Edward Grefenstette, and Max Bartolo. Procedural knowledge in pretraining drives reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=1hQKHHUsMx.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. URL https://openreview.net/forum?id=rylNH20qFQ.

Johannes Treutlein, Dami Choi, Jan Betley, Samuel Marks, Cem Anil, Roger B. Grosse, and Owain Evans. Connecting the dots: Llms can infer and verbalize latent structure from disparate training data. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/fe489a28a54583ee802b8e2955c024c2-Abstract-Conference.html.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization, 2024. URL https://doi.org/10.48550/arXiv.2405.15071.

Yujun Yan, Kevin Swersky, Danai Koutra, Parthasarathy Ranganathan, and Milad Hashemi. Neural execution engines: Learning to execute subroutines. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8b9abffb45bf79a630fb613dcd23449-Abstract.html.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. Do large language models perform latent multi-hop reasoning without exploiting shortcuts?, 2024. URL https://doi.org/10.48550/arXiv.2411.16679.

Wojciech Zaremba and Ilya Sutskever. Learning to execute, 2015. URL https://arxiv.org/abs/1410.4615.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

A QWEN RESULTS

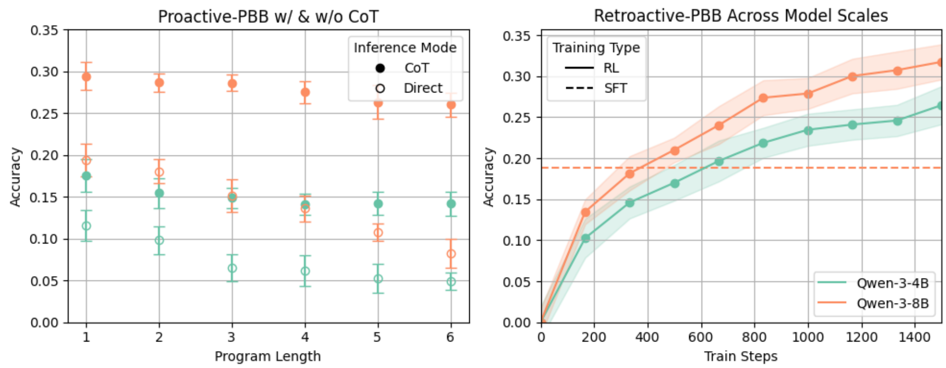

Figure 7: **Left**: Accuracy following proactive PBB on executing unpaired random arithmetic programs of different lengths. **Right**: Accuracy following retroactive PBB.

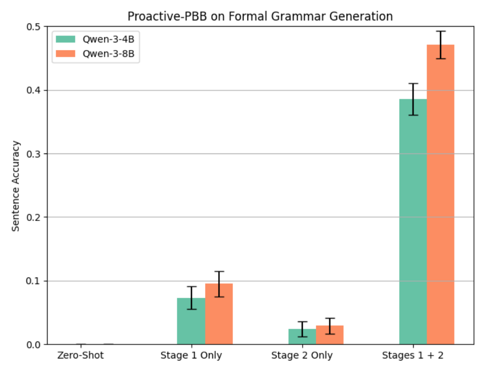

Figure 8: Compliance to unpaired grammars following each stage of proactive PBB.

B Data Scaling

B.1 Ablation over Dataset Size

Figure 9 compares the performance of Llama models (1B, 3B and 8B parameters) for varying dataset size on the evaluation of Random Arithmetic programs. Here, ‘dataset size’, refers specifically to the amount of unique code functions included in the dataset. Performance is evaluated on three separate sets:

- The *w/ IO Train* set: both the function and the IO pairs are observed during training
- The *w/ IO Test* set: uses the same functions as *w/ IO Train* but different IO pairs, not included in the training data
- The *w/o IO Test* set: evaluates IO pairs for functions seen only as code during training

The results show that accuracy on both *w/ IO* and *w/o IO* sets generally increases with larger dataset sizes and larger model scales. Notably, model performance is strongly tied to parameter count; for example, the 8B model trained on only 100 unique functions achieves comparable performance on the *w/o IO* set to the 1B model trained on 800 functions. The number of unpaired (*w/o IO*) functions is fixed at 200. The upper limit of 800 paired (*w/ IO*) functions reflects the total of 1000 available functions that we generated for Random Arithmetic.

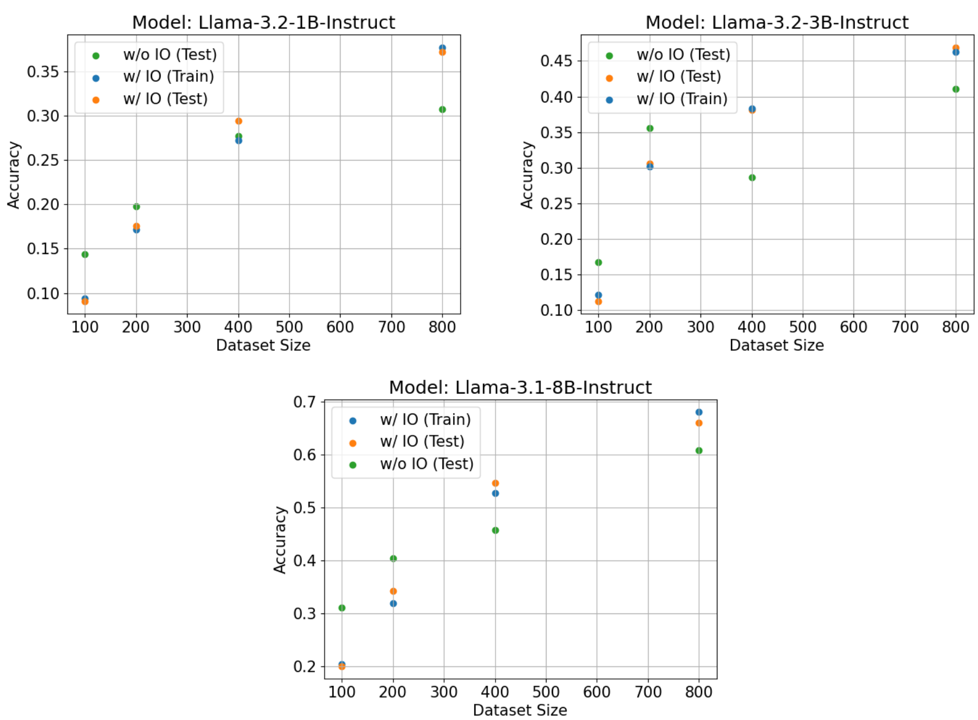

Figure 9: Performance comparison of Llama models across 1B, 3B and 8B on paired (*w/ IO*) and unpaired (*w/o IO*) Random Arithmetic program evaluation. Each model is trained and tested across varying dataset sizes. Dataset size refers to the number of unique functions present in the dataset.

B.2 Ablation over Number of IO Pairs

In Figure 10 we vary the number of IO training pairs (per program) provided for the *w/ IO* set, and examine the results. This analysis specifically uses the `Llama-3.2-3B-Instruct` model on the Random Arithmetic dataset, which for this experiment consists of 200 distinct functions. Performance is reported across the same sets as the ones described in Appendix B.1. The results show

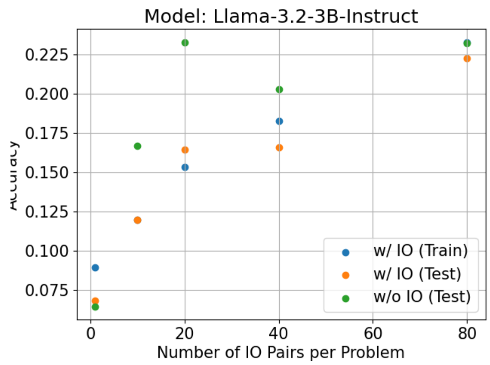

Figure 10: Impact of varying the number of IO training pairs for paired (*w/ IO*) programs and unpaired (*w/o IO*) sets evaluation accuracy. Results are shown for the `Llama-3.2-3B-Instruct` model using a Random Arithmetic dataset comprising 200 distinct functions.

how increasing the quantity of IO examples for each program affects not only direct generalisation in the *w/ IO* Test set, but also the model's ability to accurately execute *w/o IO* programs.

C SINGLE-STAGE PROGRAMMING BY BACKPROP

In Figure 11, we show the accuracy of Llama-3.1-8B-Instruct on unpaired (*w/o IO*) Random Arithmetic program evaluation following proactive PBB in comparison to a single SFT stage with all training data in a single mixture. As we scale the number of times the same piece of unpaired (*w/o IO*) source code appears in the dataset, with prompt and response preamble augmentations, single-stage SFT approaches the performance of proactive PBB. The greater sample efficiency of proactive PBB is likely because initial train steps on source code are waisted in single-stage SFT, as a code-I/O relationship has not yet been learned.

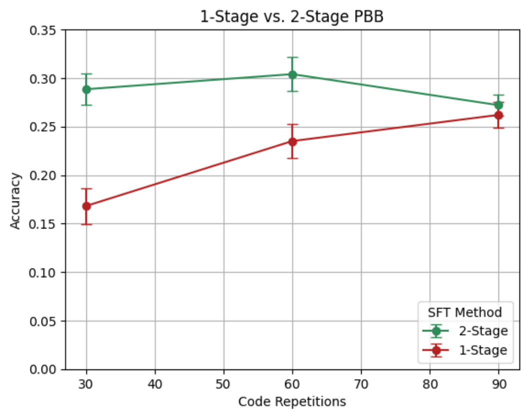

Figure 11: Comparing two-stage proactive PBB to a single SFT stage on the full Random Arithmetic training data mixture for different numbers of repeated source code samples. The base model is Llama-3.1-8B-Instruct.

D ONLINE VS. OFFLINE RETROACTIVE-PBB

In Figure 12, we compare different finetuning algorithms for the second stage of retroactive PBB with Llama-3.1-8B-Instruct on Random Arithmetic. DPO allows for learning from both positive and negative samples, considerably outperforming SFT. GRPO is an online RL algorithm, meaning that the model learns from on-policy data, which could be why it yields further improvements.

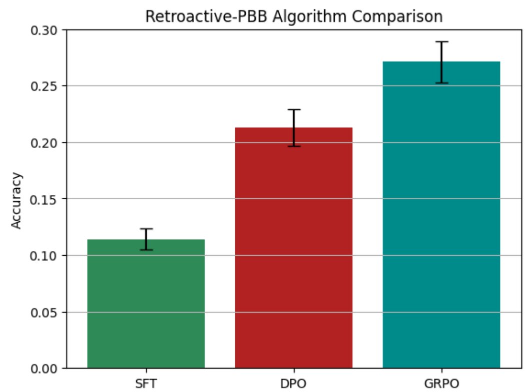

Figure 12: Comparing finetuning algorithms for the second stage of retroactive PBB on Random Arithmetic with Llama-3.1-8B-Instruct. DPO is an offline method, but allows for learning from positive and negative examples. GRPO is online and thus has the added benefit of learning from on-policy data.

E CIPHERS DATA

A plot showing the distribution of IO pairs used in Figure 5 is provided in Figure 13.

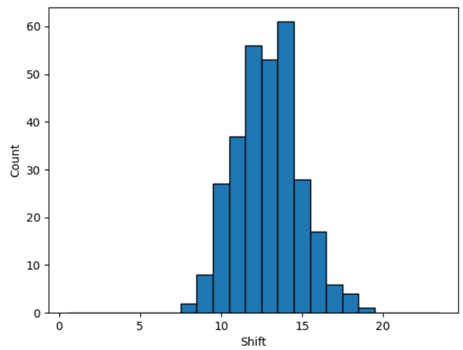

Figure 13: Sampled shifts for cipher I/O pairs.

F Natural Language Descriptions

Here, we include an example of a random arithmetic program and its natural language description.

Program:

```
def Blaankle(x):
    t0 = x + x
    t1 = 1 * abs(t0)
    return t1
```

Description: *A Blaankle is a process that takes an input value, doubles it, and then returns the absolute value of the doubled result.*

F.1 Discussion on Natural Language PBB

Current models exhibit a strong dependence on the structure of the description. Code and grammars provide explicit, unambiguous abstractions that models can internalise more reliably. Natural language (NL) lacks this regular structure and therefore poses a more challenging learning signal for PBB.

However, we see this as a scaling and representation issue rather than a fundamental limitation:

- **Positive scale-dependence:** NL PBB improves with model size, suggesting that future models may be substantially more capable in this regime.
- **Intermediate formalisms:** Many practical workflows already translate NL specifications to structured representations (pseudocode, planning languages). PBB could be applied at these intermediate stages; our work provides evidence that the formal end of this spectrum works well.
- **Synthetic datasets at scale:** Because code-like descriptions are easy to generate programmatically, PBB could be elicited in earlier LLM training stages, potentially allowing models to internalise general symbolic-interpretation skills before encountering downstream NL descriptions.
- **General mechanisms:** Our CFG experiments demonstrate that PBB extends outside programming entirely, to abstract formal systems. This indicates that what matters is symbolic structure, not code specifically.

Thus, while NL PBB is currently weaker, the paradigm itself is not restricted to code, and our results point towards concrete avenues for future work on making NL PBB practical.

G Compute Requirements

All experiments with Llama models can be run on two GPUs with 40GB vRAM. We used data parallelism over 4 NVIDIA L40s GPUs to run these experiments.

Experiments with GPT-4o made use of the OpenAI finetuning API. Data generation (Leetcode word problems and post-rationalised chain-of-thought ground truth outputs for all datasets) and finetuning runs came to a total cost just over 500 USD.

H Additional Ablations

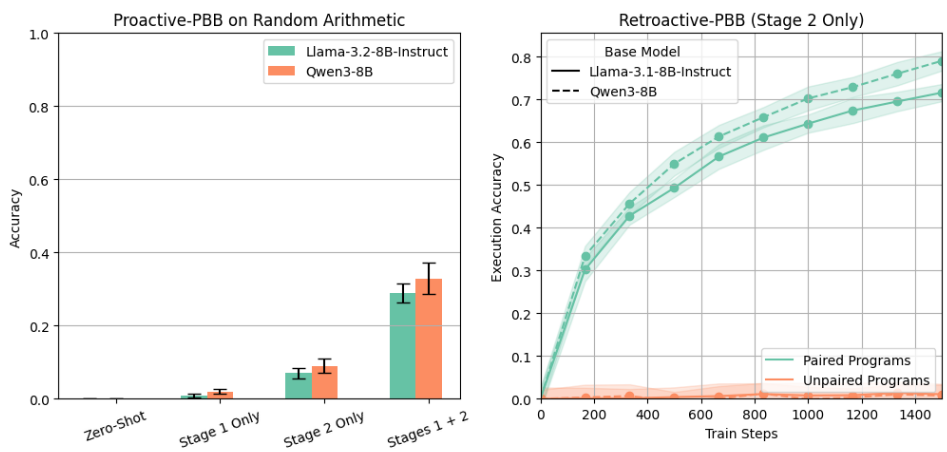

Figure 14: **Left**: Accuracy following each stage of proactive PBB on executing unpaired Random Arithmetic programs. **Right**: Accuracy for test inputs following only stage 2 ("activation") of retroactive PBB on paired vs. unpaired Random Arithmetic programs. This training corresponds to only doing RL on execution problems with train inputs on paired programs.