Emergent Representations of Program Semantics in Language Models Trained on Programs

Charles Jin [1] Martin Rinard [1]

Abstract

We present evidence that language models (LMs) of code can learn to represent the formal semantics of programs, despite being trained only to perform next-token prediction. Specifically, we train a Transformer model on a synthetic corpus of programs written in a domain-specific language for navigating 2D grid world environments. Each program in the corpus is preceded by a (partial) specification in the form of several input-output grid world states. Despite providing no further inductive biases, we find that a probing classifier is able to extract increasingly accurate representations of the *unobserved, intermediate* grid world states from the LM hidden states over the course of training, suggesting the LM acquires an emergent ability to *interpret* programs in the formal sense. We also develop a novel interventional baseline that enables us to disambiguate what is represented by the LM as opposed to learned by the probe. We anticipate that this technique may be generally applicable to a broad range of *semantic* probing experiments. In summary, this paper does not propose any new techniques for training LMs of code, but develops an experimental framework for and provides insights into the acquisition and representation of formal semantics in statistical models of code.

1. Introduction

As LMs continue to improve on a range of downstream tasks, their capabilities in the domain of *programming languages* have drawn increasing attention (Bommasani et al., 2023; Zan et al., 2023). The advancement of recent models such as GPT-4 (OpenAI et al., 2023), Code Llama (Rozière et al., 2023), Gemini (Gemini Team et al., 2023), and Claude 3 (Anthropic, 2024) has also spurred the widespread adoption of LMs in developer workflows via mainstream commercial products, offering functionality such as code completion, debugging assistance, generating documentation and commit messages, and writing test cases (Fan et al., 2023).

Despite these results, however, a major open question is whether current LMs capture any information about the *semantics* of the text that they consume and generate (Mitchell & Krakauer, 2023). Indeed, one hypothesis—which takes a unified view of both natural and programming language domains—is that LMs trained purely on form (e.g., to model the conditional distribution of tokens in a training corpus) produce text only according to surface statistical correlations gleaned from the training data (Bender & Koller, 2020), with any apparently sophisticated behavior attributable to the scale of the model and training data.

This work studies whether LMs of code learn aspects of semantics when trained using standard textual pretraining. We empirically evaluate the following hypothesis (**MH**):

Main Hypothesis. LMs of code trained only to perform next token prediction on text do not model the formal semantics of the underlying programming language.

To investigate **MH**, we apply language modeling to the task of *program synthesis*, or generating a program that implements a given (partial) specification, which we take to be a set of input-output examples. Specifically, we explore whether an LM trained on text that encodes only the *input-output behavior of programs* also learns to model the *intermediate program states* specified by the *small-step operational semantics* of the synthesized program (Plotkin, 1981). We train an LM on example programs preceded by several input-output examples, then use small classifiers to *probe* the LM's hidden states for (abstractions of) the intermediate states in the program execution. Despite the text of the training corpus encoding only input-output behavior, we find the probe's ability to extract intermediate states undergoes a phase transition during training, with the phase transition strongly correlated with the LM's ability to generate a correct program in response to previously unseen specifications. We also present results from a novel interventional experiment, which indicate that the semantics are represented by the LM (rather than learned by the probe).

[1]CSAIL, MIT, Cambridge, MA, USA. Correspondence to: Charles Jin <ccj@csail.mit.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

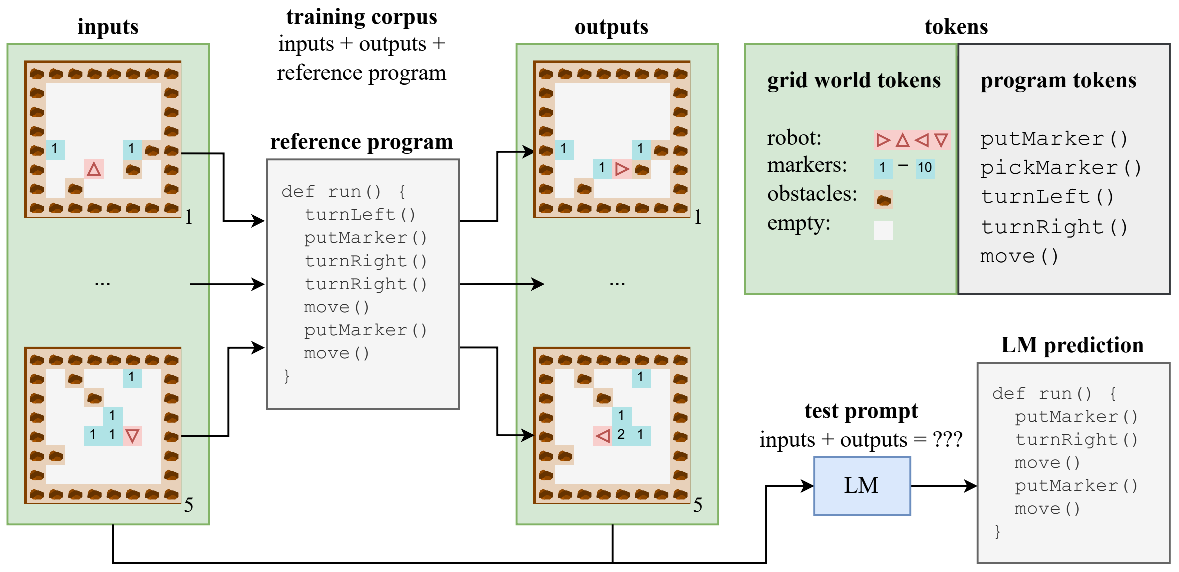


Figure 1: An overview of the experimental setting. We construct training examples by sampling a random reference program, then sampling 5 random inputs and executing the program to obtain the corresponding 5 outputs. The LM is trained for next-token prediction on a corpus of examples consisting of the interleaved inputs and outputs, then the reference program. At test time, we provide an unseen input-output specification to the LM, and use greedy decoding to predict a program.

We summarize our main contributions as follows:

Emergence of meaning We present results that are consistent with the emergence of representations of formal semantics in LMs trained to perform next token prediction (Section 3). In particular, we use the trained LM to generate programs given input-output examples, then train small probing classifiers to extract information about the intermediate program states from the hidden states of the LM. We find that the LM states encode (1) an abstract semantics—specifically, an *abstract interpretation*—that tracks the intermediate states of the program through its execution and (2) predictions of *future* program states corresponding to program tokens that have yet to be generated. During training, these representations of semantics emerge in lockstep with the LM's ability to generate correct programs.

Semantic probing interventions We present a novel interventional technique for disentangling the contributions of the LM and the probe when probing for semantics (Section 4). Specifically, one possible explanation for the results in Section 3 is that the LM states contain a (purely syntactic) record of the inputs and generated program, from which the probe learns to generate the abstract interpretation. Our key insight is that, if this were true, we should be able to supervise a new probe to interpret the (hypothetical) syntactic record according to an appropriately chosen set of *alternative* semantics and achieve accuracies similar to the original semantics. However, we find that probes trained on alternative semantics achieve lower accuracies, which is consistent with the proposition that LM states are aligned with the original semantics and inconsistent with the proposition that the LM states simply encode a syntactic record.

Taken together, Sections 3 and 4 present evidence that rejects **MH**: we find that, contrary to **MH**, representations of formal semantics emerge via next token prediction in our setting. We therefore conclude that training LMs of code solely to predict the next token does not imply that they cannot develop accurate models of the underlying domain's semantics. More broadly, we see programs and their precise formal semantics as a promising direction for working toward a deeper understanding of the behavior of LMs, such as whether or how LMs acquire and use semantic representations of the underlying domain more generally.

2. Background and setting

This section provides brief background of the *program trace* as our chosen model of formal program semantics, introduces the language modeling task and setting, and presents qualitative results from our training run.

2.1. Program tracing as meaning

A foundational topic in the theory of programming languages, formal semantics (Winskel, 1993) is the study of how to formally specify the meaning of programs. In this work, we use the *small step semantics* (Plotkin, 1981) to generate *program traces* (Cousot, 2002): given an input (i.e, an assignment of values to input variables), the trace is the sequence of intermediate program states traversed by the program as it executes over the input. A (syntactic) program can then be formally assigned a (semantic) meaning, given by the collection of all of its traces.

Beyond its amenability to formal analysis, tracing is attractive as a model of program semantics for several reasons. In novice programmers, the ability to accurately trace a piece a code has been directly linked to the ability to explain the code (Lopez et al., 2008; Lister et al., 2009), and computer science education has emphasized tracing as a method of developing program understanding (Hertz & Jump, 2013) and localizing reasoning errors (Sorva, 2013). Expert programmers also rely on tracing, both as a mental process (Letovsky, 1987) and via trace-based debuggers.

Abstract interpretation *Abstract interpretation* (Cousot & Cousot, 1977) is a technique for producing sound approximations of concrete program semantics. For example, given the multiplication operator $\times$ over the integers $\mathbb{Z}$, we could define an abstract interpretation α by abstracting each integer to its sign $\alpha : \mathbb{Z} \mapsto \{-, 0, +\}$. In this paper, we use abstract interpretation to establish a precise, formal connection between the concrete program states and the abstract program states measured in our experiments.

2.2. Language modeling task and training

Karel domain Karel is an educational programming language (Pattis, 1994) developed at Stanford in the 1970s, which is still in use in their introductory programming course today (Piech & Roberts, January 2019; CS106A, 2023). The domain features a robot (named Karel) navigating a 2D grid world with obstacles while leaving and picking up markers. Since being introduced by Devlin et al. (2017), Karel has been adopted by the program synthesis community as a standard benchmark (Bunel et al., 2018; Shin et al., 2018; Sun et al., 2018; Chen et al., 2019; 2021b), in which input-output examples are provided, and the task is to produce a program which maps each input grid to its corresponding output grid.

Figure 1 gives an overview of our domain. Each 8x8 grid world contains 4 types of tokens: the robot controlled by the program, which we represent graphically with an arrow in the direction that the robot currently faces (red); markers (blue); obstacles (brown); or an empty space (gray). We use a subset of the language consisting of the following 5 operations: `move` advances the robot by one space in the facing direction if there is not an obstacle ahead (otherwise, the robot does not move); `turnRight` and `turnLeft` turn the robot right and left, respectively; `putMarker` and `pickMarker` increment and decrement the number of markers on the space occupied by the robot (with no effect if there are 10 and 0 markers), respectively. The robot also obscures the number of markers on the space it currently occupies; the obscured markers have no effect on the correctness of the program. Note that there is no control flow and all programs consist of straight line programs, so that each operation produces exactly one program state when the program is traced.

Synthetic dataset construction Our training set consists of 500,000 randomly sampled Karel programs of lengths between 6 and 10, inclusive. For each program, we randomly sample 5 grid worlds as input, then evaluate the program to obtain 5 output grids. We create textual representations for Karel grid worlds by scanning the grid in row order, with one token per grid space. Each training sample consists of the concatenation of the 5 input-output grid world states (the *specification*), followed by the *reference program*. The language modeling task thus consists of predicting a program from a (partial) specification in the form of input-output grid world states. Note that (1) the training set consists only of programs which correctly implement the preceding specification and (2) the intermediate states of the trace are not observed in the training data. We also generate a test set of 10,000 specifications in the same manner, except that we sample reference programs of length between 1 and 10.

Training an LM to synthesize programs We train an off-the-shelf Transformer (Vaswani et al., 2017) to perform next token prediction on our dataset. Specifically, we train a 350M parameter variant of the CodeGen architecture (Nijkamp et al., 2023) in the HuggingFace Transformers library (Wolf et al., 2020) from initialization for approximately 2.5 billion tokens. Appendix A contains further details.

2.3. Results

Figure 2 plots the main results from our training run. To measure the ability of the LM to synthesize programs, we use the LM to generate text starting from a specification using greedy decoding constrained to program tokens, i.e., the generated text is guaranteed to be a syntactically well-formed program. The program is correct if it maps each input in the specification to its corresponding output; we define the **generative accuracy** as the percentage of correct programs over the test set. The LM reaches 92.4% generative accuracy at the end of training.

We also track two additional metrics related to the *syntax* (or form) of the LM outputs: the number of unique programs generated by the LM over the test set (black) and the perplexity of the LM over different subsets of tokens in the test set (orange). In particular, the solid orange line plots the perplexity on the reference programs, the dashed orange line plots the perplexity on the programs generated by the LM, and the dotted orange line plots the perplexity over all tokens. Note that overall perplexity improves over the entire run, indicating that the training dynamics remain stable.

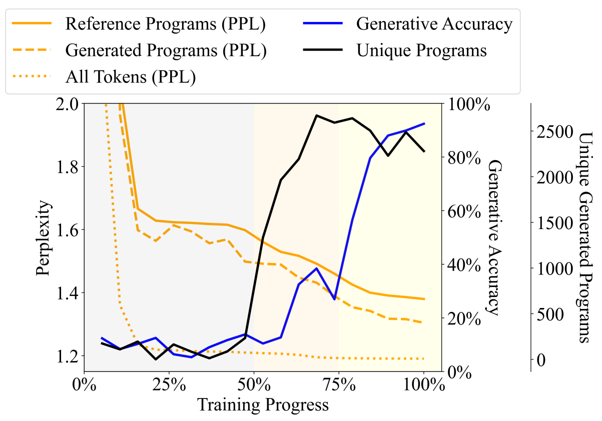


Figure 2: Three distinct phases during training: babbling (gray), syntax acquisition (orange), and semantics acquisition (yellow), based on qualitative differences in the evolution of perplexity (orange), generative accuracy (blue), and diversity of output (black). The number of unique programs is measured over the test set, which contains 10,000 specifications and 6,473 unique reference programs.

We observe 3 distinct phases during training: in the **babbling** phase (up to 50% of training, gray background), the generated programs are often highly repetitive, with a plateau in perplexity on the reference programs starting around 20%. The generative accuracy stays flat at around 10%. The next phase (50-75% of training, orange background) exhibits a sharp increase in the diversity of the generated outputs with a corresponding decrease in perplexity on the reference programs—i.e., the LM begins to model the program tokens—with a modest increase in generative accuracy (from 10% to 25%). In the final phase (from 75% to the end of training, yellow background), the diversity of the generated programs stays roughly constant, and the perplexity on the reference programs continues to improve at the previous rate. Conversely, the generative accuracy of the LM increases rapidly, from 25% to over 90%. As such, the middle phase sees the most significant change in the syntactic properties of the LM's generation, while the final phase is characterized by a rapid improvement in the LM's ability to generate semantically correct output. We hence identify these two phases primarily with **syntax acquisition** and **semantics acquisition**, respectively.

Finally, while it is natural for the perplexity of the generated programs to be lower than the reference programs (due to the use of greedy decoding), the constant margin between the two perplexities suggests that the LM consistently *underfits* the program tokens in the training data, despite producing increasingly correct programs; we refer to Appendix B for further analyses. These dynamics suggest that the increase in generative accuracy over the course of LM training cannot be attributed entirely to the LM's ability to model the surface distribution of program tokens in the training corpus.

3. Emerging representations of semantics

We train small probing classifiers to extract information about the program state from the hidden states of the LM. The idea is to prompt the LM to generate a program given some inputs, and check whether the LM states contain a representation of the *intermediate program states as it generates the program*. A positive result is consistent with the LM having learned to model the underlying semantics of the programs it generates, constituting evidence against **MH**.

3.1. Probing for representations of the program trace

Trace dataset construction Every 4000 steps (or roughly 5%) of training, we take a snapshot of (1) the hidden states of the LM as it generates programs using next-token prediction and (2) the corresponding program states after evaluating the partial program on each of the 5 specified inputs. Specifically, starting from an input-output specification, we generate according to the standard autoregressive loop:

$$(\text{state}_{LM})_i = LM(\text{input}, \text{output}, \{(\text{state}_{LM})_j\}_{j=1}^{i-1}) \quad (1)$$
$$\text{token}_i = \texttt{greedy}(LM_{\text{head}}((\text{state}_{LM})_i)) \quad (2)$$
$$(\text{state}_{\text{prog}})_i = \texttt{exec}(\text{token}_i, (\text{state}_{\text{prog}})_{i-1}) \quad (3)$$

where each $(\text{state}_{LM})_i$ is a hidden state, $(\text{state}_{\text{prog}})_0$ is the inputs from the specification, `greedy` performs greedy decoding (constrained to program tokens), and `exec` executes a program token on the program state. We generate up to 14 tokens, or until the model outputs a special end-of-sequence token (`<EOS>`). Figure 3 illustrates this process.

We average the hidden state over the layer dimension, so that the snapshot is a 1-dimensional tensor of size 1024, and call this the **model state**. We repeat this process for each of the training and test sets, producing two *trace datasets* consisting of aligned pairs of $(\text{state}_{LM})_i$, $(\text{state}_{\text{prog}})_i$ from all the programs generated from the specifications in the training and test sets, respectively

Probe training For each training trace dataset, we train a set of **probes** (ranging from linear to 2-layer MLPs) to predict features of the program state given the model state, using standard supervised learning. The features consist of (1) the facing direction of the robot, (2) the position of the robot as an offset from its starting position, and (3) whether the robot is current facing an obstacle, i.e.:

$$\alpha : \text{state}_{\text{prog}} \mapsto (\text{position}, \text{direction}, \text{obstacle}) \quad (4)$$

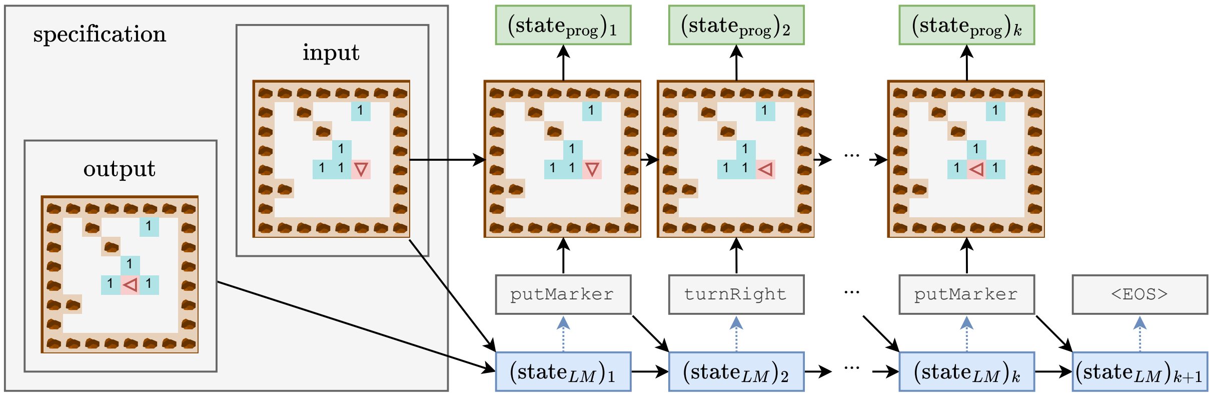


Figure 3: An overview of the trace dataset construction for the probe task. Given a specification consisting of input and output for some (unobserved) reference program, we use the trained LM to generate a program using next-token prediction (dotted blue arrows), yielding a sequence of $(\text{state}_{LM})_i$. At the same time, each token is an operation that induces a transition in the program state to $(\text{state}_{\text{prog}})_i$. The probe is trained to predict $(\text{state}_{\text{prog}})_i$ given $(\text{state}_{LM})_i$. Note that, while the depicted generation is correct as the final $(\text{state}_{\text{prog}})_k$ is equal to the specified output state, this need not be the case in general (i.e., the LM may generate incorrect programs). For clarity, we depict the specification as a single input-output example (rather than 5); autoregressive edges are also hidden.

As the features are an abstraction of the full program state, we refer to them collectively as the **abstract state**.[1] We then evaluate the accuracy of the probe over the corresponding test trace dataset, and define the **semantic content** as the geometric mean (over the 3 features). As tracing the abstract state is formally equivalent to performing an abstract interpretation of the program, the semantic content measures, in a precise sense, the extent to which the model states encode an abstract interpretation of the formal semantics.

3.2. Results

This section presents a summary of the main results; Appendix B contains additional results, including results for individual features and all 3 probes. We also evaluate several additional hypothesis, including that the semantic content is due to a retrieval process (i.e., the LM is simply recalling the abstract states from previously seen data), which can be viewed as a variation on **MH**. The main idea is that the training corpus contains only programs of length 6 or greater, so the LM cannot use retrieval for shorter programs and must learn to infer (rather than retrieve) the semantics.

Emergence of semantics is correlated with generative accuracy Figure 4 plots the results of our probing experiments. Our first observation is that the semantic content during the babbling phase is extremely noisy, which can be attributed to a lack of diversity in the outputs of the LM,

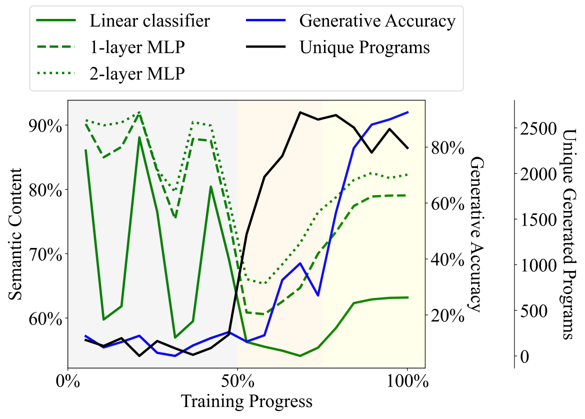


Figure 4: The semantic content (green) measured by different probing classifiers.

so that the probe only needs to fit a trivial set of semantics. For instance, about 20% of the way through training, the LM degenerates to generating a single program of 9 PICKMARKER tokens, regardless of the specification. Conversely, all 3 probes reach a minimum during the syntax acquisition phase (where the LM outputs grow more diverse), and steadily increase over the semantics acquisition phase of training. This result is consistent with the proposition that the hidden states of the LM do in fact contain (relatively) shallow encodings of the abstract state, and crucially these representations emerge within an LM trained purely to perform next token prediction on text. Regressing generative accuracy against semantic content during the second half of training also yields strong, statistically sig-

[1]To avoid introducing more notation, we use state_{LM} and $\text{state}_{\text{prog}}$ to refer to both (1) the full hidden states of the LM and program state during trace dataset generation, and (2) the averaged model state and abstract program state during probing, respectively, whenever the distinction is clear from context.

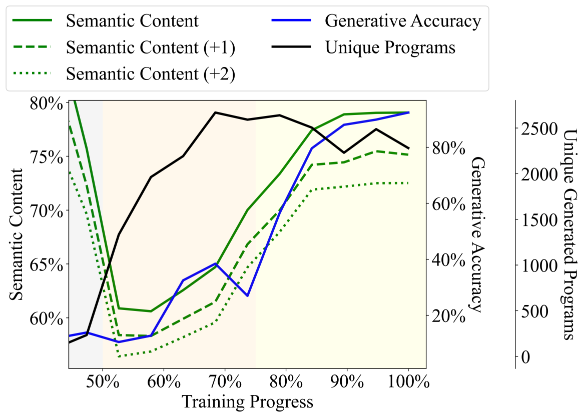


Figure 5: The semantic content (1-layer MLP) for the current and next two abstract states over the second half of training.

	0	+1	+2
linear classifier	63.2	61.1	60.1
1-layer MLP	79.1	75.1	72.5
2-layer MLP	82.3	79.2	77.3
baseline (current state)	100.0	83.9	75.6

Table 1: Probing accuracy for current and future abstract states at the end of training vs. a baseline of predicting the current abstract state for future abstract states.

nificant linear correlations ($R^2 = 0.904, 0.886, 0.821$ with $p < 0.001$ for the linear, 1-layer MLP, and 2-layer MLP probes, respectively).

Representations are predictive of future program states We next explore whether the trained LM encodes the semantics of text that has yet to be generated. Specifically, we train probes to predict *future* abstract states from model states. Figure 5 displays how well a 1-layer MLP is able to predict abstract states 1 and 2 steps into the future. As with the previous results, the probe's performance reaches a minimum during syntax acquisition, then increases for the remainder of training. We also find a strong correlation between the semantic content of future states and the generative accuracy in the second half of training; regressing the semantic content against the generative accuracy yields an R^2 of 0.878 and 0.874 ($p < .001$) for 1 and 2 abstract states into the future, respectively.

Table 1 compares the probing results at the end of training against a baseline which simply predicts the current abstract state for all future abstract states (which is the Bayes-optimal predictor absent any information about future states). We observe that (1) the accuracy of using the baseline degrades more rapidly than the probe, which suggests that the probes are not simply using encodings of the current state to predict future states, and (2) the absolute accuracy at 2 states into the future is greater using the 2-layer MLP probe than the baseline. These results suggest that the LM encodes information about what it *intends* to say ahead of its generation.

4. Semantic probing interventions

We next evaluate the possibility that semantics are learned by the probe instead of the LM. For example, because the probe is explicitly supervised on intermediate states, the model states may encode the inputs and a record of the generated program tokens, with the probe learning to interpret the tokens one-by-one. More generally, the semantic content could be attributed to (1) the LM encoding only lexical and syntactic structure while (2) the probe learns to derive the semantics from the lexical and syntactic structure encoded in the LM state (because it is explicitly supervised to predict the semantics from the LM state). We refer to this as the **syntactic record** hypothesis, which offers an explanation for the results in Section 3 consistent with **MH**.

To test this hypothesis, we design a novel interventional experiment that preserves the lexical and syntactic structure of the language and intervenes only on the semantics. Then, we re-execute the program with the alternative semantics to obtain a *new* trace with *new* abstract states, and train a *new* probe to predict the new abstract states using the *original* model states. Our key insight is that, if the model states encode only syntactic information, then the new probe should be capable of extracting the new semantics from the original syntactic record equally well, leaving the semantic content unchanged.

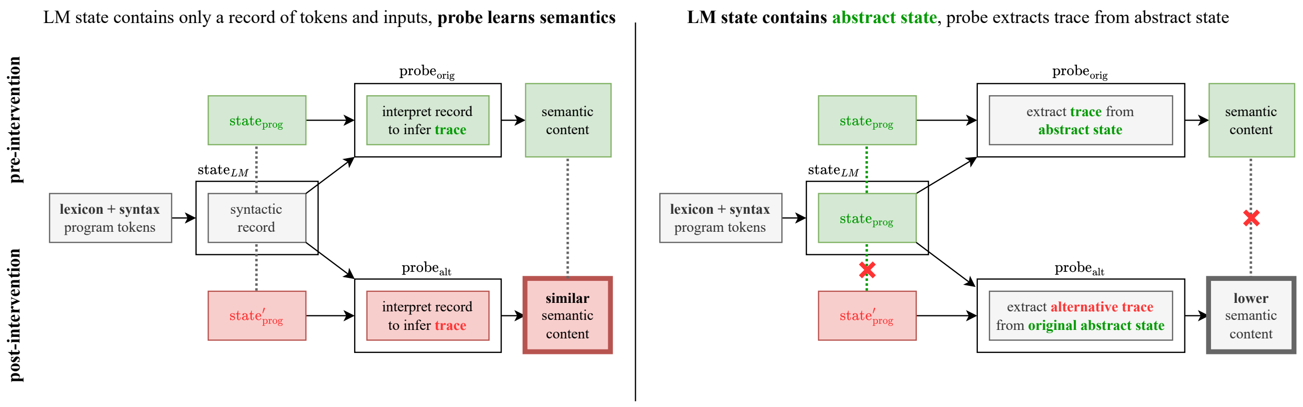
4.1. Intervening on semantics

Concretely, we define two **alternative semantics** by reassigning the semantics of individual operations as follows:

original	flip	adversarial
`pickMarker`	`pickMarker`	`turnRight`
`putMarker`	`putMarker`	`turnLeft`
`turnRight`	`turnLeft`	`move`
`turnLeft`	`turnRight`	`turnRight`
`move`	`move`	`turnLeft`

For instance, $\texttt{exec}(\texttt{turnRight}, \cdot)$ in the original semantics would have the robot turn 90 degrees clockwise, while $\texttt{exec}_{\text{adversarial}}(\texttt{turnRight}, \cdot)$ advances the robot by a space (i.e., according to the original semantics of `move`).

Next, for each sequence of program tokens in Equation (1) from the construction of the original trace dataset, we use



LM state contains only a record of tokens and inputs, probe learns semantics
LM state contains abstract state, probe extracts trace from abstract state
pre-intervention
post-intervention
lexicon + syntax program tokens
state$_{LM}$
syntactic record
state$_{prog}$
state$'_{prog}$
probe$_{orig}$
probe$_{alt}$
interpret record to infer trace
semantic content
similar semantic content
extract trace from abstract state
extract alternative trace from original abstract state
lower semantic content

Figure 6: The proposed semantic intervention baseline. We use green for the original semantics, red for the alternative semantics, and gray for non-semantic components (such as syntax). We aim to distinguish between two hypotheses: (left) the LM only maintains a syntactic record (e.g., a list of the inputs and program tokens generated thus far), while probe$_{\text{orig}}$ learns to infer semantics from the record; and (right) the LM learns to represent state$_{\text{prog}}$, while probe$_{\text{orig}}$ just extracts the semantics. We mark the emergent connection between the original semantics and the LM representations in the latter case by a dashed green line. The top half depicts how, pre-intervention, both cases can lead to the high semantic content measured in Section 3. The bottom half displays why intervening on the semantics while preserving the form of programs separates the two hypotheses: if the LM representations contain only syntax (bottom left), then it should be possible to train probe$_{\text{alt}}$ to learn to interpret the record according to the alternative state$'_{\text{prog}}$ (bold red outcome); however, if the LM representations encode the original abstract state (bottom right), then probe$_{\text{alt}}$ needs to extract the alternative state$'_{\text{prog}}$ from the original state$_{\text{prog}}$, yielding a lower semantic content (bold gray outcome).

Table 2: The results of our semantic intervention experiments. For each of the original, flip, and adversarial semantics, we report the semantic content (SC) at the end of training for 2 abstract states into the past (-2, -1), the current state (0), and 2 abstract states into the future (+1, +2), using linear, 1-layer MLP, and 2-layer MLP probes. We also regress the SC against the generative accuracy over the second half of training ($R^2(p)$). For each of the alternative semantics, we additionally compute the difference with respect to the original semantics (Δ) and regress the difference against the generative accuracy over the second half of training as ($R^2(p)$ of Δ). Highlighted cells are statistically significant at a level of $p < 0.05$ with an R^2 of at least 50%; all such correlations are positive. We reject the syntactic record hypothesis due to the magnitude (Δ and R^2 of Δ) and statistical significance (p of Δ) of the drop in semantic content when probing for the alternative semantics.

		original		flip				adversarial			
		SC	$R^2(p)$	SC	$R^2(p)$	Δ	$R^2(p)$ of Δ	SC	$R^2(p)$	Δ	$R^2(p)$ of Δ
linear	-2	64.2	86.0 (<.001)	60.5	55.5 (0.021)	3.7	72.2 (0.004)	55.8	7.4 (0.478)	8.4	60.5 (0.014)
	-1	64.4	87.1 (<.001)	59.9	66.1 (0.008)	4.5	81.4 (<.001)	54.8	2.6 (0.680)	9.7	70.7 (0.005)
	0	63.2	90.4 (<.001)	57.3	52.7 (0.027)	5.9	86.9 (<.001)	53.1	0.4 (0.878)	10.1	71.9 (0.004)
	1	61.1	91.3 (<.001)	55.2	38.4 (0.075)	5.9	89.3 (<.001)	52.6	0.6 (0.846)	8.5	59.4 (0.015)
	2	60.1	92.3 (<.001)	54.3	23.2 (0.189)	5.7	90.7 (<.001)	52.6	0.7 (0.827)	7.5	49.4 (0.035)
MLP-1	-2	82.8	83.8 (<.001)	81.2	87.1 (<.001)	1.6	23.2 (0.190)	73.8	66.3 (0.008)	9.0	66.1 (0.008)
	-1	83.6	83.9 (<.001)	81.6	86.9 (<.001)	1.9	30.3 (0.125)	73.4	65.4 (0.008)	10.1	72.9 (0.003)
	0	79.1	88.6 (<.001)	76.7	90.3 (<.001)	2.3	61.4 (0.012)	66.7	44.7 (0.049)	12.4	76.5 (0.002)
	1	75.1	87.8 (<.001)	72.2	87.2 (<.001)	2.9	83.3 (<.001)	61.8	29.0 (0.135)	13.4	75.0 (0.003)
	2	72.5	87.4 (<.001)	69.1	86.9 (<.001)	3.4	84.7 (<.001)	59.1	12.2 (0.356)	13.4	75.1 (0.002)
MLP-2	-2	85.4	75.5 (0.002)	84.0	76.7 (0.002)	1.4	48.5 (0.037)	83.1	74.5 (0.003)	2.3	5.2 (0.556)
	-1	85.6	78.0 (0.002)	83.9	79.3 (0.001)	1.7	64.3 (0.009)	82.5	72.6 (0.004)	3.2	20.7 (0.218)
	0	82.3	82.1 (<.001)	80.6	84.3 (<.001)	1.7	63.0 (0.011)	76.1	68.1 (0.006)	6.2	25.2 (0.169)
	1	79.2	82.5 (<.001)	77.6	84.5 (<.001)	1.7	62.0 (0.012)	69.8	51.7 (0.029)	9.5	48.0 (0.039)
	2	77.3	83.7 (<.001)	75.6	86.0 (<.001)	1.6	58.2 (0.017)	65.6	48.0 (0.039)	11.6	59.3 (0.015)

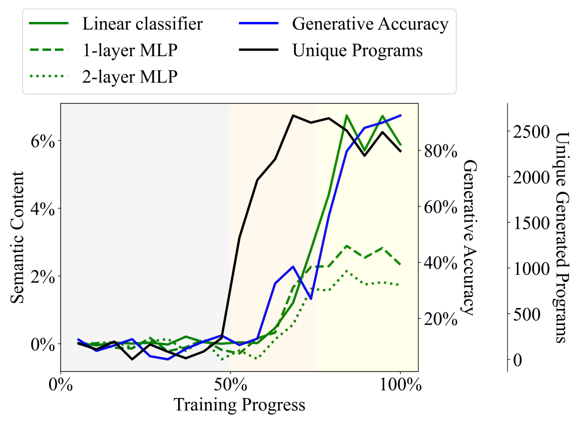


Figure 7: Excess of original over flip semantic content using different probing classifiers.

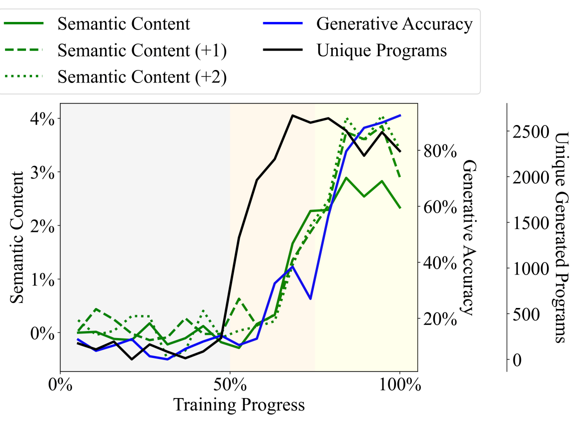


Figure 8: Excess of original over flip semantic content for current and future abstract states (1-layer MLP) .

the *same* tokens to define a corresponding alternative trace:

$$(\text{state}'_{\text{prog}})_i = \texttt{exec}_{\text{alt}}(\text{token}_i, (\text{state}'_{\text{prog}})_{i-1}), \qquad (5)$$

where we also start from the *same* initial program state: $(\text{state}_{\text{prog}})'_0 = (\text{state}_{\text{prog}})_0$ (i.e., the specification inputs).

Finally, with the *new* traces from Equation (5) and the *original* state_{LM} from Equation (1), we train a new probe

$$\text{probe}_{\text{alt}} : \text{state}_{LM} \mapsto \text{state}'_{\text{prog}}, \qquad (6)$$

and compare its accuracy against the original probe:

$$\text{probe}_{\text{orig}} : \text{state}_{LM} \mapsto \text{state}_{\text{prog}}. \qquad (7)$$

Figure 6 illustrates our setup.

For an intervention to be a proper control, we identify two critical properties: (1) the alternative semantics should be limited to *reassigning* the semantics of individual operations in the language (as opposed to inventing completely new semantics, e.g., "jump three spaces diagonally") and (2) the intervention must preserve the syntactic structure of programs (i.e., how the individual operations are composed when interpreting a full program). Then assuming the syntactic record hypothesis is true, $\text{probe}_{\text{alt}}$ should be able to interpret the record according to an analogous procedure as $\text{probe}_{\text{orig}}$, yielding comparable measurements of semantic content. Hence, *rejecting the syntactic record hypothesis reduces to measuring a statistically significant degradation in the alternative semantic content* (relative to the original semantic content).

4.2. Results

Table 2 displays the results of our semantic intervention baseline, where we trained probes to predict up to two abstract states into the past and future using the original and alternative semantics. In all 5 cases, the semantic content for the alternative semantics is significantly degraded when compared to the original semantics, which supports rejecting the hypothesis that the model states only encode a syntactic record (i.e., lexical and syntactic information only) while the probe learns to interpret the record (i.e., semantics).

Note that the flip semantics are strongly related to the original semantics: absent any obstacles in the robot's path, they only require reflecting the original path of the robot across an axis; in contrast, the adversarial semantics completely changes the semantics of each operator. Hence, if the syntactic record hypothesis is false, then we would expect the semantic content to be lower for adversarial vs. flip, since it should be more challenging to map from the original to the adversarial semantics; our results support this interpretation.

We also plot the excess of the original over the flip semantic contents in Figures 7 and 8. Note that the apparently high semantic content of the babbling phase—which was observed pre-intervention in Figures 4 and 5, respectively, and attributed to the probe being able to learn the semantics of a small number of unique generated programs (black)—disappears post-intervention. This is consistent with the probe learning the semantics equally well for both the original and flip semantics during the babbling phase, and demonstrates the ability of the semantic intervention baseline to control for the ability of the probe to learn semantics. We conclude that a statistically significant portion of the observed semantic content from Section 3 cannot be explained as the probe learning semantics, refuting **MH**.

5. Related work

LMs, semantics, and interpretability While many works have evaluated the external behavior of LMs on a range of semantically meaningful tasks (Austin et al., 2021; Toshniwal et al., 2022; Patel & Pavlick, 2022; Liu et al., 2023), our work explores the internal state of the LM, falling under the broad umbrella of efforts toward LM interpretability. For instance, Abdou et al. (2021) find that pretrained LMs' representations of color terms are geometrically aligned with CIELAB, a color space based on human perception. Li et al. (2021) fine-tune pretrained LMs on text that describes evolving situations, then probe whether the model states track entity states. Unlike this research, we study an LM trained from scratch, which (1) yields insights into how semantics emerge in the representations of LMs over time and (2) allows us to rigorously evaluate alternative explanations of the LM behavior. Li et al. (2023) train a Transformer on transcripts of Othello, then use probes to intervene on the LM's internal representations; they find that the LM's subsequent generations are consistent with the edited version of the extracted board state. In contrast, our semantic probing interventions introduce a novel method of distinguishing the contributions of the LM and probe. To the best of our knowledge, we are also the first to apply probing to find evidence that the LM encodes the meaning of text ahead of generation.

Grounding programs from text The specific question of whether LMs of *code* can ground programs from text has received prior attention in the literature. Merrill et al. (2021) show that there exist programs whose semantics provably cannot be learned from text, albeit under strong assumptions not satisfied by our setting. Bender & Koller (2020) concede that meaning could be learned from programs paired with unit tests, but assert this requires a "learner which has been equipped by its human developer with the ability to identify and interpret unit tests." Our research, in contrast, provides empirical evidence that LMs trained only to predict the next token can, in fact, learn aspects of the program semantics.

Program synthesis with LMs There is a growing body of work on training large-scale, Transformer-based LMs for program synthesis (Austin et al., 2021; Chen et al., 2021a; Li et al., 2022; Nijkamp et al., 2023; Fried et al., 2023; Rozière et al., 2023), as well as program synthesis as a benchmark for LMs (Hendrycks et al., 2021; Liang et al., 2022). Several of these works have observed that the BLEU score with respect to a reference solution is not a good predictor of the LM's competency, which complements our results regarding the LM's perplexity on the training corpus.

Probing Probing (Shi et al., 2016; Belinkov & Glass, 2019) is widely used as a technique to investigate the inner workings of LMs. A key challenge is controlling for what is learned by the probe rather than represented in the LM (Belinkov, 2022). The standard methodology is to establish a baseline measurement on a task for which the model states are assumed to be meaningless. Hewitt & Liang (2019) develop *control tasks* for word-level properties in the context of probing for parts of speech in LM representations. They compare against the performance of a probe that maps from the model states to a dataset with a *random* part of speech assigned to each word. In our case, the control task approach would assign random features to each program state; however, this also destroys the syntactic structure of the program, and hence cannot be used to test the syntactic record hypothesis. To address this, we introduce *semantic* probing interventions, a control framework for probing that intervenes on the semantics of individual operations while preserving the overall syntax of programs. As our techniques specifically advance the study of semantics in LMs, we believe our contributions can be broadly applicable to future interpretability research, particularly for investigations involving meaning (rather just than syntactic structure).

6. Conclusion

This paper presents empirical evidence that **LMs of code can acquire the formal semantics of programs from next token prediction**. We find that, when training an LM to model text consisting of examples of input-output specifications followed by programs, the learning process of the LM appears to undergo 3 distinct phases, with the second half of training characterized by a strong, linear correlation between the emerging representations of the semantics and the ability of the LM to synthesize programs that correctly implement unseen specifications. We also find representations of *future* semantics, suggesting a notion of intent during generation. Further explorations of these dynamics could yield deeper insights into the behavior of LMs.

We also present **semantic probing interventions**, a framework for the application of probes—a standard tool for interpreting the learned representations of, e.g., neural models—to understanding whether representations capture information related to the underlying semantics of a domain. Specifically, we design experiments capable of distinguishing whether the probe's measurement is indicative of (1) the presence of semantic information intrinsic to the representations or (2) the ability of the probe to perform the task itself, with purely syntactic information encoded in the representations. This also allows us to justify the use of nonlinear probes that, absent our technique, are more likely yield false positives due to having more capacity to learn the task; we see moving beyond shallow probes as a way to progress toward understanding whether (and how) LMs represent more complex concepts.

More broadly, the question of what exactly LMs are learning from text has garnered considerable interest in recent years, driven by the increasingly impressive performance of frontier models. We believe the techniques and insights presented in this work can serve as a principled foundation for future studies of the capabilities and limitations of LMs.

Acknowledgements

We would like to thank Jacob Andreas, Omar Costilla-Reyes, Kai Jia, Jason Kim, Armando Solar-Lezama, and Yichen Yang for their helpful comments and discussions on an earlier version of this paper. We also gratefully acknowledge support from DARPA Grants HR001120C0015, HR001120C0191, and N6600120C4025. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

Impact statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

To aid in reproducibility, we open source all our code, including the code we use to generate the training data, train the LM, and conduct the probing experiments, at https://github.com/charlesjin/emergent-semantics.

References

Abdou, M., Kulmizev, A., Hershcovich, D., Frank, S., Pavlick, E., and Søgaard, A. Can language models encode perceptual structure without grounding? a case study in color. In *Proceedings of the 25th Conference on Computational Natural Language Learning*, pp. 109–132, 2021.

Anthropic, A. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 2024.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Belinkov, Y. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, March 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7.

Belinkov, Y. and Glass, J. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72, 2019. doi: 10.1162/tacl_a_00254. URL https://aclanthology.org/Q19-1004.

Bender, E. M. and Koller, A. Climbing towards nlu: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 5185–5198, 2020.

Bommasani, R., Liang, P., and Lee, T. Holistic evaluation of language models. *Annals of the New York Academy of Sciences*, 2023.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Xw62kRZ.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Chen, X., Liu, C., and Song, D. Execution-guided neural program synthesis. In *International Conference on Learning Representations*, 2019.

Chen, X., Song, D., and Tian, Y. Latent execution for neural program synthesis beyond domain-specific languages. *Advances in Neural Information Processing Systems*, 34: 22196–22208, 2021b.

Cousot, P. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1-2):47–103, 2002.

Cousot, P. and Cousot, R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 238–252, 1977.

CS106A. CS106A: Programming Methodologies (spring 2023). https://web.archive.org/web/20230515003120/https://web.stanford.edu/class/cs106a/, 2023. URL https://web.stanford.edu/class/cs106a/. Accessed: 2023-05-14.

Devlin, J., Bunel, R. R., Singh, R., Hausknecht, M., and Kohli, P. Neural program meta-induction. *Advances in Neural Information Processing Systems*, 30, 2017.

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., and Zhang, J. M. Large language models for software engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533*, 2023.

Fried, D., Aghajanyan, A., Lin, J., Wang, S., Wallace, E., Shi, F., Zhong, R., Yih, S., Zettlemoyer, L., and Lewis, M. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hQwb-lbM6EL.

Gemini Team, Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Petrov, S., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., Lillicrap, T., Lazaridou, A., Firat, O., Molloy, J., Isard, M., Barham, P. R., Hennigan, T., Lee, B., Viola, F., Reynolds, M., Xu, Y., Doherty, R., Collins, E., Meyer, C., Rutherford, E., Moreira, E., Ayoub, K., Goel, M., Tucker, G., Piqueras, E., Krikun, M., Barr, I., Savinov, N., Danihelka, I., Roelofs, B., White, A., Andreassen, A., von Glehn, T., Yagati, L., Kazemi, M., Gonzalez, L., Khalman, M., Sygnowski, J., Frechette, A., Smith, C., Culp, L., Proleev, L., Luan, Y., Chen, X., Lottes, J., Schucher, N., Lebron, F., Rrustemi, A., Clay, N., Crone, P., Kocisky, T., Zhao, J., Perz, B., Yu, D., Howard, H., Bloniarz, A., Rae, J. W., Lu, H., Sifre, L., Maggioni, M., Alcober, F., Garrette, D., Barnes, M., Thakoor, S., Austin, J., Barth-Maron, G., Wong, W., Joshi, R., Chaabouni, R., Fatiha, D., Ahuja, A., Liu, R., Li, Y., Cogan, S., Chen, J., Jia, C., Gu, C., Zhang, Q., Grimstad, J., Hartman, A. J., Chadwick, M., Tomar, G. S., Garcia, X., Senter, E., Taropa, E., Pillai, T. S., Devlin, J., Laskin, M., de Las Casas, D., Valter, D., Tao, C., Blanco, L., Badia, A. P., Reitter, D., Chen, M., Brennan, J., Rivera, C., Brin, S., Iqbal, S., Surita, G., Labanowski, J., Rao, A., Winkler, S., Parisotto, E., Gu, Y., Olszewska, K., Zhang, Y., Addanki, R., Miech, A., Louis, A., Shafey, L. E., Teplyashin, D., Brown, G., Catt, E., Attaluri, N., Balaguer, J., Xiang, J., Wang, P., Ashwood, Z., Briukhov, A., Webson, A., Ganapathy, S., Sanghavi, S., Kannan, A., Chang, M.-W., Stjerngren, A., Djolonga, J., Sun, Y., Bapna, A., Aitchison, M., Pejman, P., Michalewski, H., Yu, T., Wang, C., Love, J., Ahn, J., Bloxwich, D., Han, K., Humphreys, P., Sellam, T., Bradbury, J., Godbole, V., Samangooei, S., Damoc, B., Kaskasoli, A., Arnold, S. M. R., Vasudevan, V., Agrawal, S., Riesa, J., Lepikhin, D., Tanburn, R., Srinivasan, S., Lim, H., Hodkinson, S., Shyam, P., Ferret, J., Hand, S., Garg, A., Paine, T. L., Li, J., Li, Y., Giang, M., Neitz, A., Abbas, Z., York, S., Reid, M., Cole, E., Chowdhery, A., Das, D., Rogozińska, D., Nikolaev, V., Sprechmann, P., Nado, Z., Zilka, L., Prost, F., He, L., Monteiro, M., Mishra, G., Welty, C., Newlan, J., Jia, D., Allamanis, M., Hu, C. H., de Liedekerke, R., Gilmer, J., Saroufim, C., Rijhwani, S., Hou, S., Shrivastava, D., Baddepudi, A., Goldin, A., Ozturel, A., Cassirer, A., Xu, Y., Sohn, D., Sachan, D., Amplayo, R. K., Swanson, C., Petrova, D., Narayan, S., Guez, A., Brahma, S., Landon, J., Patel, M., Zhao, R., Villela, K., Wang, L., Jia, W., Rahtz, M., Giménez, M., Yeung, L., Lin, H., Keeling, J., Georgiev, P., Mincu, D., Wu, B., Haykal, S., Saputro, R., Vodrahalli, K., Qin, J., Cankara, Z., Sharma, A., Fernando, N., Hawkins, W., Neyshabur, B., Kim, S., Hutter, A., Agrawal, P., Castro-Ros, A., van den Driessche, G., Wang, T., Yang, F., yiin Chang, S., Komarek, P., McIlroy, R., Lučić, M., Zhang, G., Farhan, W., Sharman, M., Natsev, P., Michel, P., Cheng, Y., Bansal, Y., Qiao, S., Cao, K., Shakeri, S., Butterfield, C., Chung, J., Rubenstein, P. K., Agrawal, S., Mensch, A., Soparkar, K., Lenc, K., Chung, T., Pope, A., Maggiore, L., Kay, J., Jhakra, P., Wang, S., Maynez, J., Phuong, M., Tobin, T., Tacchetti, A., Trebacz, M., Robinson, K., Katariya, Y., Riedel, S., Bailey, P., Xiao, K., Ghelani, N., Aroyo, L., Slone, A., Houlsby, N., Xiong, X., Yang, Z., Gribovskaya, E., Adler, J., Wirth, M., Lee, L., Li, M., Kagohara, T., Pavagadhi, J., Bridgers, S., Bortsova, A., Ghemawat, S., Ahmed, Z., Liu, T., Powell, R., Bolina, V., Iinuma, M., Zablotskaia, P., Besley, J., Chung, D.-W., Dozat, T., Comanescu, R., Si, X., Greer, J., Su, G., Polacek, M., Kaufman, R. L., Tokumine, S., Hu, H., Buchatskaya, E., Miao, Y., Elhawaty, M., Siddhant, A., Tomasev, N., Xing, J., Greer, C., Miller, H., Ashraf, S., Roy, A., Zhang, Z., Ma, A., Filos, A., Besta, M., Blevins, R., Klimenko, T., Yeh, C.-K., Changpinyo, S., Mu, J., Chang, O., Pajarskas, M., Muir, C., Cohen, V., Lan, C. L., Haridasan, K., Marathe, A., Hansen, S., Douglas, S., Samuel, R., Wang, M., Austin, S., Lan, C., Jiang, J., Chiu, J., Lorenzo, J. A., Sjösund, L. L., Cevey, S., Gleicher, Z., Avrahami, T., Boral, A., Srinivasan, H., Selo, V., May, R., Aisopos, K., Hussenot, L., Soares, L. B., Baumli, K., Chang, M. B., Recasens, A., Caine, B., Pritzel, A., Pavetic, F., Pardo, F., Gergely, A., Frye, J., Ramasesh, V., Horgan, D., Badola, K., Kassner, N., Roy, S., Dyer, E., Campos, V., Tomala, A., Tang, Y., Badawy, D. E., White, E., Mustafa, B., Lang, O., Jindal, A., Vikram, S., Gong, Z., Caelles, S., Hemsley, R., Thornton, G., Feng, F., Stokowiec, W., Zheng, C., Thacker, P., Çağlar Ünlü, Zhang, Z., Saleh, M., Svensson, J., Bileschi, M., Patil, P., Anand, A., Ring, R., Tsihlas, K., Vezer, A., Selvi, M., Shevlane, T., Rodriguez, M., Kwiatkowski, T., Daruki, S., Rong, K., Dafoe, A., FitzGerald, N., Gu-Lemberg, K., Khan, M., Hendricks, L. A., Pellat, M., Feinberg, V., Cobon-Kerr, J., Sainath, T., Rauh, M., Hashemi, S. H., Ives, R., Hasson, Y., Li, Y., Noland, E., Cao, Y., Byrd, N., Hou, L., Wang, Q., Sottiaux, T., Paganini, M., Lespiau, J.-B., Moufarek, A., Hassan, S., Shivakumar, K., van Amersfoort, J., Mandhane, A., Joshi, P., Goyal, A., Tung, M., Brock, A., Sheahan, H., Misra, V., Li, C., Rakićević, N., Dehghani, M., Liu, F., Mittal, S., Oh, J., Noury, S., Sezener, E., Huot, F., Lamm, M., Cao, N. D., Chen, C., Elsayed, G., Chi, E., Mahdieh, M., Tenney, I., Hua, N., Petrychenko, I., Kane, P., Scandinaro, D., Jain, R., Uesato, J., Datta, R., Sadovsky, A., Bunyan, O., Rabiej, D., Wu, S., Zhang, J., Vasudevan, G., Leurent, E., Alnahlawi, M., Georgescu, I., Wei, N., Zheng, I., Chan, B., Rabinovitch, P. G., Stanczyk, P., Zhang, Y., Steiner, D., Naskar, S., Azzam, M., Johnson, M., Paszke, A., Chiu, C.-C., Elias, J. S., Mohiuddin, A., Muhammad, F., Miao, J., Lee, A., Vieillard, N., Potluri, S., Park, J., Davoodi, E., Zhang, J., Stanway, J., Garmon, D., Karmarkar, A., Dong, Z., Lee, J., Kumar, A., Zhou, L., Evens, J., Isaac, W., Chen, Z., Jia, J., Levskaya, A., Zhu, Z., Gorgolewski, C., Grabowski, P., Mao, Y., Magni, A., Yao, K., Snaider, J., Casagrande, N., Suganthan, P., Palmer, E., Irving, G., Loper, E., Faruqui, M., Arkatkar, I., Chen, N., Shafran, I., Fink, M., Castaño, A., Giannoumis, I., Kim, W., Rybiński, M., Sreevatsa, A., Prendki, J., Soergel, D., Goedeckemeyer, A., Gierke, W., Jafari, M., Gaba, M., Wiesner, J., Wright, D. G., Wei, Y., Vashisht, H., Kulizhskaya, Y., Hoover, J., Le, M., Li, L., Iwuanyanwu, C., Liu, L., Ramirez, K., Khorlin, A., Cui, A., LIN, T., Georgiev, M., Wu, M., Aguilar, R., Pallo, K., Chakladar, A., Repina, A., Wu, X., van der Weide, T., Ponnapalli, P., Kaplan, C., Simsa, J., Li, S., Dousse, O., Yang, F., Piper, J., Ie, N., Lui, M., Pasumarthi, R., Lintz, N., Vijayakumar, A., Thiet, L. N., Andor, D., Valenzuela, P., Paduraru, C., Peng, D., Lee, K., Zhang, S., Greene, S., Nguyen, D. D., Kurylowicz, P., Velury, S., Krause, S., Hardin, C., Dixon, L., Janzer, L., Choo, K., Feng, Z., Zhang, B., Singhal, A., Latkar, T., Zhang, M., Le, Q., Abellan, E. A., Du, D., McKinnon, D., Antropova, N., Bolukbasi, T., Keller, O., Reid, D., Finchelstein, D., Raad, M. A., Crocker, R., Hawkins, P., Dadashi, R., Gaffney, C., Lall, S., Franko, K., Filonov, E., Bulanova, A., Leblond, R., Yadav, V., Chung, S., Askham, H., Cobo, L. C., Xu, K., Fischer, F., Xu, J., Sorokin, C., Alberti, C., Lin, C.-C., Evans, C., Zhou, H., Dimitriev, A., Forbes, H., Banarse, D., Tung, Z., Liu, J., Omernick, M., Bishop, C., Kumar, C., Sterneck, R., Foley, R., Jain, R., Mishra, S., Xia, J., Bos, T., Cideron, G., Amid, E., Piccinno, F., Wang, X., Banzal, P., Gurita, P., Noga, H., Shah, P., Mankowitz, D. J., Polozov, A., Kushman, N., Krakovna, V., Brown, S., Bateni, M., Duan, D., Firoiu, V., Thotakuri, M., Natan, T., Mohananey, A., Geist, M., Mudgal, S., Girgin, S., Li, H., Ye, J., Roval, O., Tojo, R., Kwong, M., Lee-Thorp, J., Yew, C., Yuan, Q., Bagri, S., Sinopalnikov, D., Ramos, S., Mellor, J., Sharma, A., Severyn, A., Lai, J., Wu, K., Cheng, H.-T., Miller, D., Sonnerat, N., Vnukov, D., Greig, R., Beattie, J., Caveness, E., Bai, L., Eisenschlos, J., Korchemniy, A., Tsai, T., Jasarevic, M., Kong, W., Dao, P., Zheng, Z., Liu, F., Yang, F., Zhu, R., Geller, M., Teh, T. H., Sanmiya, J., Gladchenko, E., Trdin, N., Sozanschi, A., Toyama, D., Rosen, E., Tavakkol, S., Xue, L., Elkind, C., Woodman, O., Carpenter, J., Papamakarios, G., Kemp, R., Kafle, S., Grunina, T., Sinha, R., Talbert, A., Goyal, A., Wu, D., Owusu-Afriyie, D., Du, C., Thornton, C., Pont-Tuset, J., Narayana, P., Li, J., Fatehi, S., Wieting, J., Ajmeri, O., Uria, B., Zhu, T., Ko, Y., Knight, L., Héliou, A., Niu, N., Gu, S., Pang, C., Tran, D., Li, Y., Levine, N., Stolovich, A., Kalb, N., Santamaria-Fernandez, R., Goenka, S., Yustalim, W., Strudel, R., Elqursh, A., Lakshminarayanan, B., Deck, C., Upadhyay, S., Lee, H., Dusenberry, M., Li, Z., Wang, X., Levin, K., Hoffmann, R., Holtmann-Rice, D., Bachem, O., Yue, S., Arora, S., Malmi, E., Mirylenka, D., Tan, Q., Koh, C., Yeganeh, S. H., Põder, S., Zheng, S., Pongetti, F., Tariq, M., Sun, Y., Ionita, L., Seyedhosseini, M., Tafti, P., Kotikalapudi, R., Liu, Z., Gulati, A., Liu, J., Ye, X., Chrzaszcz, B., Wang, L., Sethi, N., Li, T., Brown, B., Singh, S., Fan, W., Parisi, A., Stanton, J., Kuang, C., Koverkathu, V., Choquette-Choo, C. A., Li, Y., Lu, T., Ittycheriah, A., Shroff, P., Sun, P., Varadarajan, M., Bahargam, S., Willoughby, R., Gaddy, D., Dasgupta, I., Desjardins, G., Cornero, M., Robenek, B., Mittal, B., Albrecht, B., Shenoy, A., Moiseev, F., Jacobsson, H., Ghaffarkhah, A., Rivière, M., Walton, A., Crepy, C., Parrish, A., Liu, Y., Zhou, Z., Farabet, C., Radebaugh, C., Srinivasan, P., van der Salm, C., Fidjeland, A., Scellato, S., Latorre-Chimoto, E., Klimczak-Plucińska, H., Bridson, D., de Cesare, D., Hudson, T., Mendolicchio, P., Walker, L., Morris, A., Penchev, I., Mauger, M., Guseynov, A., Reid, A., Odoom, S., Loher, L., Cotruta, V., Yenugula, M., Grewe, D., Petrushkina, A., Duerig, T., Sanchez, A., Yadlowsky, S., Shen, A., Globerson, A., Kurzrok, A., Webb, L., Dua, S., Li, D., Lahoti, P., Bhupatiraju, S., Hurt, D., Qureshi, H., Agarwal, A., Shani, T., Eyal, M., Khare, A., Belle, S. R., Wang, L., Tekur, C., Kale, M. S., Wei, J., Sang, R., Saeta, B., Liechty, T., Sun, Y., Zhao, Y., Lee, S., Nayak, P., Fritz, D., Vuyyuru, M. R., Aslanides, J., Vyas, N., Wicke, M., Ma, X., Bilal, T., Eltyshev, E., Balle, D., Martin, N., Cate, H., Manyika, J., Amiri, K., Kim, Y., Xiong, X., Kang, K., Luisier, F., Tripuraneni, N., Madras, D., Guo, M., Waters, A., Wang, O., Ainslie, J., Baldridge, J., Zhang, H., Pruthi, G., Bauer, J., Yang, F., Mansour, R., Gelman, J., Xu, Y., Polovets, G., Liu, J., Cai, H., Chen, W., Sheng, X., Xue, E., Ozair, S., Yu, A., Angermueller, C., Li, X., Wang, W., Wiesinger, J., Koukoumidis, E., Tian, Y., Iyer, A., Gurumurthy, M., Goldenson, M., Shah, P., Blake, M., Yu, H., Urbanowicz, A., Palomaki, J., Fernando, C., Brooks, K., Durden, K., Mehta, H., Momchev, N., Rahimtoroghi, E., Georgaki, M., Raul, A., Ruder, S., Redshaw, M., Lee, J., Jalan, K., Li, D., Perng, G., Hechtman, B., Schuh, P., Nasr, M., Chen, M., Milan, K., Mikulik, V., Strohman, T., Franco, J., Green, T., Hassabis, D., Kavukcuoglu, K., Dean, J., and Vinyals, O. Gemini: A family of highly capable multimodal models, 2023.

Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

Hertz, M. and Jump, M. Trace-based teaching in early programming courses. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pp. 561–566, 2013.

Hewitt, J. and Liang, P. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing* *and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2733–2743, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1275. URL https://aclanthology.org/D19-1275.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

LeBrun, B., Sordoni, A., and O'Donnell, T. J. Evaluating distributional distortion in neural language modeling. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=bTteFbU99ye.

Letovsky, S. Cognitive processes in program comprehension. *Journal of Systems and software*, 7(4):325–339, 1987.

Li, B. Z., Nye, M., and Andreas, J. Implicit representations of meaning in neural language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1813–1827, 2021.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Lister, R., Fidge, C., and Teague, D. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin*, 41(3):161–165, 2009.

Liu, R., Wei, J., Gu, S. S., Wu, T.-Y., Vosoughi, S., Cui, C., Zhou, D., and Dai, A. M. Mind's eye: Grounded language model reasoning through simulation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=4rXMRuoJlai.

Lopez, M., Whalley, J., Robbins, P., and Lister, R. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research*, pp. 101–112, 2008.

Meister, C. and Cotterell, R. Language model evaluation beyond perplexity. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5328–5339, 2021.

Merrill, W., Goldberg, Y., Schwartz, R., and Smith, N. A. Provable limitations of acquiring meaning from ungrounded form: What will future language models understand? *Transactions of the Association for Computational Linguistics*, 9:1047–1060, 2021.

Mitchell, M. and Krakauer, D. C. The debate over understanding in AI's large language models. *Proceedings of the National Academy of Sciences*, 120(13), mar 2023. doi: 10.1073/pnas.2215907120. URL https://doi.org/10.1073%2Fpnas.2215907120.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., and Xiong, C. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=iaYcJKpY2B_.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. GPT-4 technical report, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Patel, R. and Pavlick, E. Mapping language models to grounded conceptual spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gJcEM8sxHK.

Pattis, R. E. *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons, 1994.

Piech, C. and Roberts, E. Karel Reader: Python version. https://compedu.stanford.edu/karel-reader/docs/python/en/intro.html, January 2019. Accessed May 8, 2023.

Plotkin, G. D. A structural approach to operational semantics. *Aarhus University, Report DAIMI FN-19*, 1981.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code Llama: Open foundation models for code, 2023.

Shi, X., Padhi, I., and Knight, K. Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1526–1534, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1159. URL https://aclanthology.org/D16-1159.

Shin, E. C., Polosukhin, I., and Song, D. Improving neural program synthesis with inferred execution traces. *Advances in Neural Information Processing Systems*, 31, 2018.

Sorva, J. Notional machines and introductory programming education. *ACM Trans. Comput. Educ.*, 13(2), jul 2013. doi: 10.1145/2483710.2483713. URL https://doi.org/10.1145/2483710.2483713.

Sun, S.-H., Noh, H., Somasundaram, S., and Lim, J. Neural program synthesis from diverse demonstration videos. In *International Conference on Machine Learning*, pp. 4790–4799. PMLR, 2018.

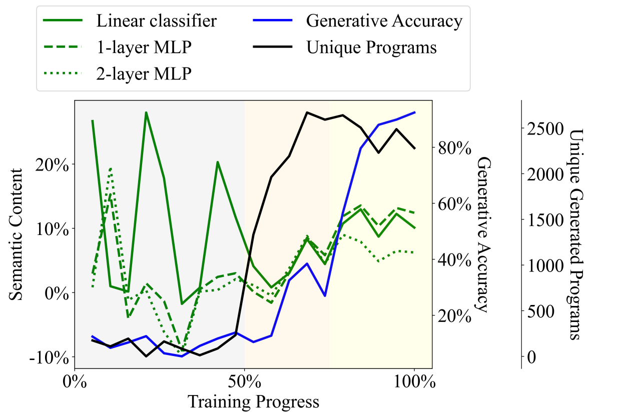
Toshniwal, S., Wiseman, S., Livescu, K., and Gimpel, K. Chess as a testbed for language model state tracking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 11385–11393, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

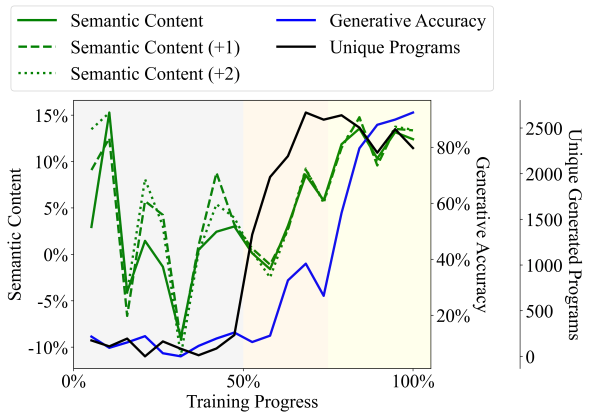
Winskel, G. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Zan, D., Chen, B., Zhang, F., Lu, D., Wu, B., Guan, B., Yongji, W., and Lou, J.-G. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7443–7464, 2023.



(a) Excess of original over adversarial semantic content using different probing classifiers.



(b) Excess of original over adversarial semantic content for current and future abstract states (1-layer MLP) .

Figure 9: Excess of original over adversarial semantic content.

A. Experimental details

A.1. Karel grammar specification

We use the same grammar as Devlin et al. (2017), except with loops and conditionals removed.

$$\begin{aligned}
\text{Prog } p &:= \texttt{def run():} s \\
\text{Stmt } s &:= s_1; s_2 \mid a \\
\text{Action } a &:= \texttt{move()} \mid \texttt{turnRight()} \mid \texttt{turnLeft()} \mid \texttt{pickMarker()} \mid \texttt{putMarker()}
\end{aligned}$$

A.2. Language model and probe details

We train the 350M-parameter variant of the CodeGen architecture (Nijkamp et al., 2023) from the HuggingFace Transformers library (Wolf et al., 2020), implemented in PyTorch (Paszke et al., 2019). We use the Adam optimizer, a learning rate of 5e-5, a block size of 2048, and a batch size of 32768 tokens. We train for 2.5 billion tokens, which was close to 6 passes over our training corpus; we did not observe any instabilities with training (see the results in the main text). We use a warm up over roughly the first 3000 batches, then linearly decay the learning rate to 0 after 80000 batches (training runs over 76000 batches). On a single NVIDIA A100 GPU with 80GB of VRAM, training takes around 8 days.

The linear probe consists of a single linear layer. The MLP probes consist of stacked linear, batch norm, and ReLU layers, in that order. The first linear layer in both MLP probes projects to a hidden dimension of 256, and the second linear layer (in the 2-layer MLP) projects to a hidden dimension of 1024. Probes are trained on the first 100000 aligned traces in the training trace dataset. All probes are trained using the same recipe, which was tuned to saturate (or nearly saturate) their performance: we use the AdamW optimizer with weight decay of 1e-4 and a learning rate of 0.01 that decays by .1 at 75% and 90% of the way through training; and train for 10000000 steps using a batch size of 1024.

B. Additional experimental results

B.1. Plots for adversarial semantics

Figures 9a and 9b provide plots for the adversarial semantics that are analogous to those provided in the main text for the flip semantics (specifically Figures 7 and 8, respectively). We observe that, in contrast to the flip semantics (which achieve close to zero excess semantic content in the babbling phase), the excess of the original over adversarial semantics exhibit significant noise during the babbling phase of the LM training, which is often negative; we attribute this to weaker relationship between the adversarial and original semantics conjectured in Section 4: certain distributions of programs may be easier for the adversarial vs. original semantics, and vice versa. We also note that the excess semantic content is largely constant across the current and future states in Figure 9b, particularly in the final phases of training, which is consistent with

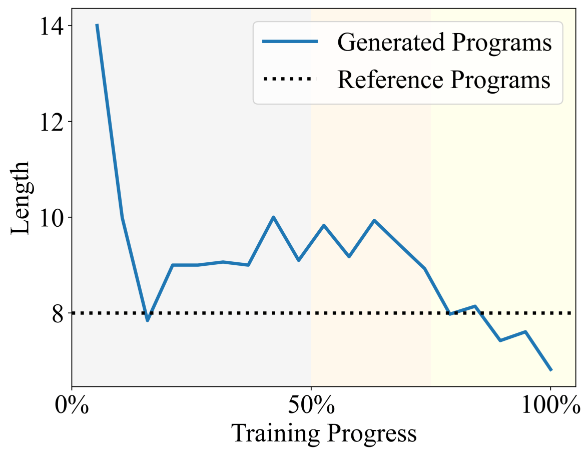


Figure 10: The average length of generated programs over time generated from specifications sampled according to the training distribution and compared with the corresponding reference programs.

the probe learning the same map from the original abstract state to the adversarial abstract state (and thus having roughly the same absolute error rate, independent of how far into the future the abstract states are relative to the model state).

B.2. Generated programs become shorter than reference programs over training

Figure 10 plots the average length of programs generated by the LMs over specifications sampled from the training distribution (i.e., where the reference programs are of length 6 to 10, inclusive). The results indicate that the LM learns to generate programs which are, on average, shorter than the reference program length. While this can be partially explained by the fact that we use greedy decoding to generate the outputs, we emphasize that the generated programs are also grow increasingly correct over the second half of training, i.e., the quality of the LM's generation continues to improve despite diverging from the distribution of the training corpus.

These results complements our findings in the main text, which show that the LM tends to underfit the distribution of tokens in the training data. Although prior work has explored the differences between an LM's output and its training corpus based on surface statistics (Meister & Cotterell, 2021; LeBrun et al., 2022), we are, to the best of our knowledge, the first to present an account of how such syntactic divergence relate to the semantic properties of the LM generation and training data.

B.3. Semantic content of reference programs

This section explores a possible implication of the finding in Section 3 that representations contain predictions of *future* program states. In particular, these future program states are defined by the tokens produced by greedy decoding, i.e., the program is generated by taking the token that the LM models as most likely at each step in the autoregressive loop of Equation (1). However, it has been observed in other settings that sampling from the tokens according to a distribution derived from the logits of the LM head can improve the quality of the LM's outputs, e.g., with multinomial or nucleus sampling (Holtzman et al., 2020). However, if the LM states do in fact encode the future program states according to greedy decoding, then any deviation could, in fact, force the LM states to visit "unanticipated" program states.

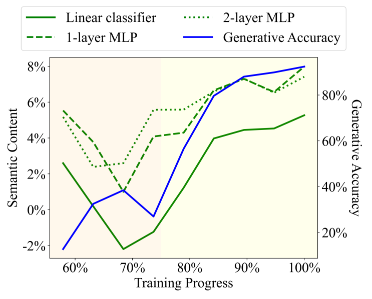
We perform a preliminary exploration of this hypothesis. Specifically, we measure the semantic content when decoding from the LM according to the *reference* programs used to generate the specifications in our train and test datasets: we create new **reference trace datasets** by replacing the greedy decoding step of Equation (2) with the tokens from the reference programs, i.e.,

$$(\text{state}_{LM})_i = LM(\text{input}, \text{output}, \{(\text{state}_{LM})_j\}_{j=1}^{i-1}) \tag{8}$$

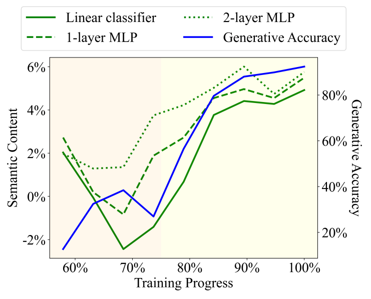
$$\text{token}_i = \text{prog}_{\text{ref}}[i] \tag{9}$$

$$(\text{state}_{\text{prog}})_i = \text{exec}(\text{token}_i, (\text{state}_{\text{prog}})_{i-1}) \tag{10}$$

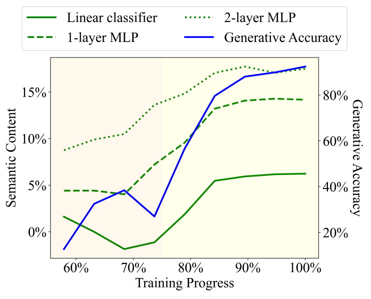
where prog_{ref} is the reference program and $\text{prog}_{\text{ref}}[i]$ is the i^{th} program token in the reference program. The idea is that,



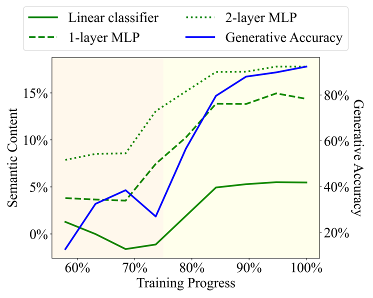
(a) Probing 2 states into the past.



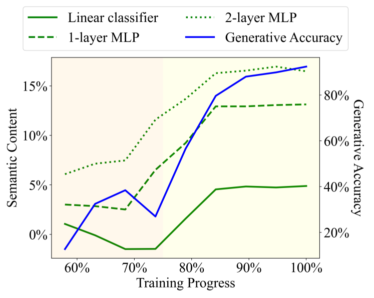
(b) Probing 1 state into the past.



(c) Probing the current state.



(d) Probing 1 state into the future.



(e) Probing 2 states into the future.

Figure 11: Plotting the difference in the semantic content when decoding according to the reference programs vs. greedy decoding. A positive difference indicates that greedy decoding yields a higher semantic content.

because the LM is specifically trained to minimize its perplexity on the reference programs, we might expect the LM representations to also be closely aligned with the semantics of the training corpus.

Figure 11 plots the results of this experiment. For the linear probe, extracting both past and future semantic states does appear easier up until the beginning of the last phase of training. However, by the end of training, probing for the abstract states from greedy decoding all exhibit higher semantic content than the reference programs, which suggests that the LM representations are indeed better aligned with greedy decoding. We also observe that the difference is particularly large for the present and future semantic states. This can be explained by the fact that the future program states are essentially a random process (due to the program itself being generated according to a random sampling procedure), and hence, there is a fundamental limit to how informative the LM's representation can be. Note that even the "current" abstract state is a random process, as the the current abstract state also results from a token that the LM has yet to see: the probe's task for the current abstract state is to predict $(\text{state}_{\text{prog}})_i$ from $(\text{state}_{LM})_i$ in Equation (8).

Finally, we note that the difference (between the semantic contents with respect to the reference programs vs. generated programs) grows as the LM's generative accuracy improves; we conjecture that the LM's preference for greedy decoding may increase as it becomes better calibrated to the training data as a general principle. We leave a more detailed exploration of how different decoding strategies affect the coherence of the LM's internal state to future work.

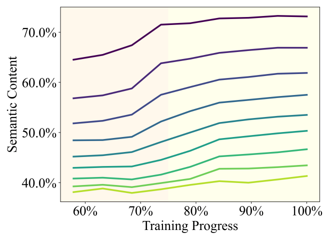
B.4. Semantics are inferred, not retrieved

In this section, we evaluate the **retrieval hypothesis**: the semantic content can be attributed entirely to the LM recalling previously seen training data. Similar to the syntactic record hypothesis, this hypothesis offers another potential explanation for the results in Section 3 which is consistent with **MH**. For instance, if we test the LM on (A) "`turnRight`, `move`, `turnLeft`" and the LM was trained on (B) "`turnRight`, `move`", then the LM may have seen the second entry in the trace of (A) as the output of (B).

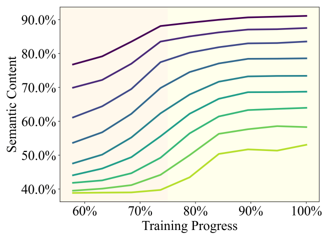
We design our experimental setting specifically to test this hypothesis: as the training corpus contains only programs of length 6 or greater, while the test set contains programs of length 1 to 10, it is impossible for the LM to "retrieve" program states corresponding to the first 5 entries in any trace. Hence, we argue that any representations of unseen program states

Table 3: The semantic content at the end of training, separated by the depth of the program state and the 3 features in the abstract state. The LM only observes program states at depth 6 or greater in the training corpus. We display depths consisting of at least 1% of the training set.

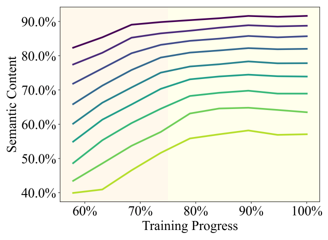
	depth	linear			MLP-1			MLP-2		
		direction	position	obstacle	direction	position	obstacle	direction	position	obstacle
unseen	1	46.7	85.0	73.7	76.7	92.9	78.6	89.1	93.3	79.5
	2	47.9	71.3	73.0	76.4	85.2	77.7	87.3	85.7	78.2
	3	49.8	64.0	73.5	76.7	80.1	78.2	86.8	80.3	78.5
	4	50.5	61.4	74.8	78.8	76.9	79.1	87.3	77.3	79.4
	5	51.5	60.1	76.0	79.7	74.9	80.2	87.1	74.9	80.4
seen	6	59.9	60.9	74.6	82.1	73.7	79.9	87.6	73.6	80.5
	7	59.2	58.8	74.3	79.1	68.6	79.3	84.3	68.2	79.3
	8	51.7	57.1	76.0	69.8	62.2	79.9	75.0	62.9	80.2



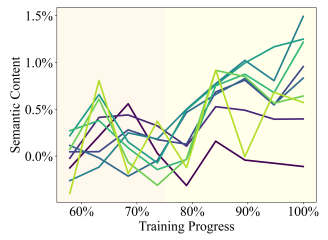
(a) Probing with a linear classifier.



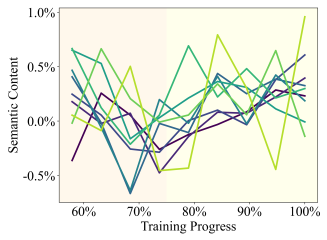
(b) Probing with a 1-layer MLP.



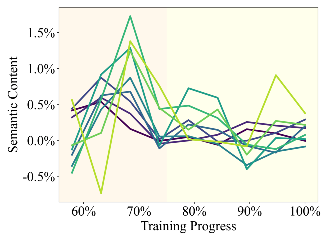
(c) Probing with a 2-layer MLP.



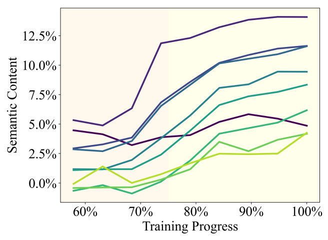
(d) Probing with a linear classifier, excess over flip semantics.



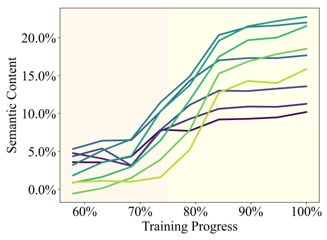
(e) Probing with a 1-layer MLP, excess over flip semantics.



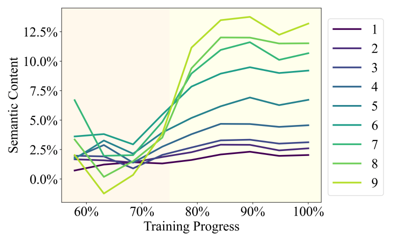
(f) Probing with a 2-layer MLP, excess over flip semantics.



(g) Probing with linear classifier, excess over adversarial semantics.



(h) Probing with a 1-layer MLP, excess over adversarial semantics.



(i) Probing with a 2-layer MLP, excess over adversarial semantics.

Figure 12: Plotting the semantic content separated by depth over time. Note that the excess over the two alternative semantics starts at zero during the middle phase of training, then becomes significantly positive around the final phase for all depths, including those corresponding to unseen programs.

must be *inferred* by the LM according to the semantics of the programming language.

Table 3 displays the accuracy of the probes at the end of training; as the probe achieves non-trivial accuracy when tracing *unseen* programs (of length 5 or less), we argue that the observed semantic content cannot be fully attributed to a retrieval-like process, and instead requires the LM to perform some degree of generalization over the semantics. We also remark that Appendix B.3 offers another piece of evidence in support of this claim, as the semantic content is lower when using the reference programs (whereas we would expect the ability of the LM to perform retrieval to *increase* when following the training distribution more closely).

We make 2 additional observations. First, the effect of retrieval differs by feature, and appears more pronounced the "simpler" the feature is to compute. For instance, with the direction feature, there is a local optimum in the accuracy of the linear and 1-layer MLP probes at depth 6, which is indicative that some amount of retrieval is occurring in the representations of the LM: as the direction of the robot depends only on the initial direction and program, the LM has likely seen the answer previously in its training data. Conversely, the feature corresponding to the position of the robot decreases monotonically with depth for all 3 probes (with a very small exception at depth 6 for the linear probe), which is consistent with execution (deeper program states are harder to trace without a mistake due to compounding errors). Second, deeper probes exhibit lower sensitivity to retrieval across all 3 features. As deeper probes can extract more complex representations, this suggests that (1) the results of retrieval are represented more shallowly in the LM states, while (2) benefits of the shallow retrieval representations are less important once the probe is able to extract the deeper semantic representations.

Figure 12 also plots, for all 3 probes and separated by depth across all training steps, (1) the semantic content for the original semantics and (2) the excess semantic content with respect to the flip and adversarial semantics. The results show that the excess over the flip and adversarial semantics begins around zero in the middle phase (for all depths), then becomes positive by the end of training. This behavior is consistent with the results in the main text that suggest that the LM learns semantics over the latter half of training. We also point out the excess semantic content for the flip semantics as shown in Figures 12d to 12f increase as the depth increases from 1 to 5 (starting from close to 0 at depth 1), which we attribute to the high degree of correlation between the flip and original semantics (which degrades as the program length increases); beyond depth 5, the difference actually decreases, whereas we would expect the difference to be greater if the probe is simply relying on representations of retrieved program states in the original semantics. To summarize, while we do find evidence that the LM is performing some retrieval process, our results indicate that this process coexists with semantics in the representations of the LM.

B.5. Interpreting programs without outputs

We conduct another series of experiments for explore whether the LM is capable of interpreting programs when only inputs are provided in the specification, and the outputs in the specification are obscured. Specifically, for every reference program and specification in the original trace datasets, we generate an **input-only trace dataset** according to the following loop (cf. Equation (1)):

$$(\text{state}_{LM})_i = LM(\text{input}, \text{empty}, \{(\text{state}_{LM})_j\}_{j=1}^{i-1}) \tag{11}$$

$$\text{token}_i = \text{prog}_{\text{ref}}[i] \tag{12}$$

$$(\text{state}_{\text{prog}})_i = \text{exec}(\text{token}_i, (\text{state}_{\text{prog}})_{i-1}) \tag{13}$$

where prog_{ref} is the reference program and $\text{prog}_{\text{ref}}[i]$ is the i^{th} program token in the reference program; input $= (\text{state}_{\text{prog}})_0$ is the input state from the specification; and we replace every output in the specification with an *empty* Karel grid. In other words, we force the LM to generate the reference program given *only inputs and no outputs*.

We emphasize that this task is firmly outside of the distribution of the training data, and there is no guarantee that the LM states will even be coherent. Hence, any ability to interpret programs in this setting could be considered *emergent* behavior on the part of the LM. Additionally, as the text still contains the necessary information for *interpreting* the program (as the output is not necessary for this task), we also expect to find that probing for the alternative semantics yields accuracies that are no higher than the original semantics.

Table 4 displays the results. As expected, the semantic content for the original abstract states is significantly degraded (compared to Table 2), especially when probing into the future, which we attribute mainly to obscuring the outputs. We also see that there the performance of the 3 probes is highly compressed when probing into the future, i.e., deeper probes do not perform much better than shallow probes. This can be explained by the same reasoning as in Appendix B.3, i.e.,

Table 4: The results of our input-only probing experiments. For each of the original, flip, and adversarial semantics, we report the semantic content (SC) at the end of training for 2 abstract states into the past (-2, -1), the current state (0), and 2 abstract states into the future (+1, +2), using linear, 1-layer MLP, and 2-layer MLP probes. We also regress the SC against the generative accuracy over the second half of training ($R^2(p)$). For each of the alternative semantics, we additionally compute the difference with respect to the original semantics (Δ) and regress the difference against the generative accuracy over the second half of training as ($R^2(p)$ of Δ). Highlighted cells are statistically significant at a level of $p < 0.05$ with an R^2 of at least 50%; all such correlations are positive.

		original		flip				adversarial			
		SC	$R^2(p)$	SC	$R^2(p)$	Δ	$R^2(p)$ of Δ	SC	$R^2(p)$	Δ	$R^2(p)$ of Δ
linear	-2	57.7	89.6 (<.001)	57.5	88.2 (<.001)	0.3	55.8 (0.021)	50.5	68.6 (0.006)	7.2	92.9 (<.001)
	-1	60.5	83.8 (<.001)	59.8	80.4 (0.001)	0.7	84.1 (<.001)	51.5	55.5 (0.021)	9.1	87.4 (<.001)
	0	54.6	85.9 (<.001)	54.2	82.7 (<.001)	0.4	92.9 (<.001)	47.6	6.3 (0.514)	7.0	86.7 (<.001)
	1	52.4	81.1 (<.001)	52.2	80.7 (0.001)	0.2	75.7 (0.002)	47.5	79.0 (0.001)	4.9	84.8 (<.001)
	2	51.7	70.3 (0.005)	51.6	69.1 (0.006)	0.1	62.1 (0.012)	47.6	12.0 (0.362)	4.1	74.8 (0.003)
MLP-1	-2	77.3	81.2 (<.001)	77.2	81.8 (<.001)	0.0	0.8 (0.824)	62.6	54.7 (0.023)	14.7	89.8 (<.001)
	-1	79.8	81.7 (<.001)	79.5	80.4 (0.001)	0.3	38.3 (0.076)	64.7	56.1 (0.020)	15.1	88.4 (<.001)
	0	62.3	82.4 (<.001)	62.3	81.4 (<.001)	0.1	5.1 (0.558)	50.2	70.6 (0.005)	12.2	83.6 (<.001)
	1	55.3	80.8 (<.001)	55.3	80.1 (0.001)	0.0	22.8 (0.194)	49.1	39.3 (0.071)	6.2	85.7 (<.001)
	2	52.9	70.7 (0.005)	52.7	73.0 (0.003)	0.1	0.0 (0.974)	48.5	55.7 (0.021)	4.4	76.3 (0.002)
MLP-2	-2	81.6	68.5 (0.006)	81.7	69.5 (0.005)	-0.1	2.1 (0.710)	76.1	43.1 (0.055)	5.5	48.3 (0.038)
	-1	82.5	70.1 (0.005)	82.4	70.5 (0.005)	0.1	18.7 (0.245)	77.7	52.3 (0.028)	4.8	84.8 (<.001)
	0	63.8	67.6 (0.006)	63.7	69.6 (0.005)	0.2	28.3 (0.140)	51.3	59.2 (0.015)	12.5	69.0 (0.006)
	1	56.7	76.1 (0.002)	56.8	69.1 (0.005)	-0.1	0.4 (0.872)	49.3	47.8 (0.039)	7.4	80.5 (0.001)
	2	53.9	59.4 (0.015)	53.7	59.7 (0.015)	0.2	4.6 (0.580)	48.8	49.4 (0.035)	5.1	60.8 (0.013)

Table 5: The excess semantic content at the end of training when comparing the original and alternative semantics using a linear probe on the input-only trace datasets. We separate the semantic content by the depth of the program state and the 3 features in the abstract state. The LM only observes program states at depth 6 or greater in the training corpus. We display depths consisting of at least 1% of the training set. A positive value indicates that the original semantic content is greater than the alternative semantic content.

	depth	flip				adversarial			
		direction	position	obstacle	all	direction	position	obstacle	all
unseen	1	-0.51	0.15	0.17	-0.11	9.30	1.19	1.63	4.86
	2	0.36	0.31	0.50	0.40	25.35	3.58	0.97	14.08
	3	1.53	0.39	0.65	0.96	20.97	4.25	-0.12	11.63
	4	1.92	-0.14	0.17	0.83	20.09	4.80	-0.17	11.60
	5	3.12	0.32	-0.00	1.49	16.07	4.52	-0.09	9.44
seen	6	2.33	0.55	0.00	1.25	12.66	5.78	0.40	8.34
	7	2.01	0.39	0.71	1.22	9.37	4.30	-0.04	6.17
	8	1.46	-0.17	0.15	0.64	5.96	3.47	-0.33	4.17
	9	1.04	-0.04	0.50	0.57	4.32	4.06	2.64	4.26

the present and future program states are subject to an inherent degree of randomness (recall that the probe's task for the "current" abstract state is to predict $(\text{state}_{\text{prog}})_i$ from $(\text{state}_{LM})_i$ in Equation (11)). Indeed, we note that the semantic content is maximized at 1 abstract state into the past across all settings.

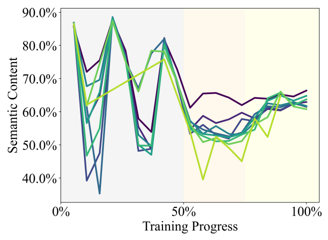
Nonetheless, our results suggest the LM still maintains the ability to interpret programs, even in this challenging setting; moreover, the interventional baseline offers strong support that the representations of the LM are aligned with the original semantics (rather than being learned by the probe). In particular, we see statistically significant correlations for all three probes and all 5 abstract states when comparing the original semantic content against the adversarial semantic content. Note that, even though predicting future states is difficult, the ability to predict future states is correlated with knowing the current state, so that it is not surprisingly that the adversarial semantic content for future states would be lower, despite the lack of information about future states.

We also observe a statistically significant positive correlation when regressing the excess of the original semantic content over the flip semantic content over the second half of training, when using the linear probe, though this effect disappears as we move to deeper probes. We emphasize that the flip semantics presents a very strong baseline as the semantics can often be inferred directly from the result of the original semantics (i.e., by reflecting the robot across an axis).

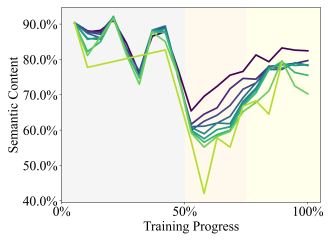
Could these results could be due to retrieval? To test this hypothesis, we also reproduce the experiments from Appendix B.4 on the input-only trace dataset. We focus on 1 abstract state into the past, as it is where the semantic content is maximized across all settings; and the linear probe, as the original semantic content and excess over the alternative semantics are all correlated with the generative accuracy to a statistically significant degree over the course of training. Table 5 displays the excess semantic content of the original over the flip and adversarial semantics, respectively, using a linear probe on the input-only trace datasets when probing for 1 abstract state into the past. We see that almost every semantic content of *unseen* features (and overall abstract state) is greater when probing for the original semantics compared to the alternative semantics, confirming that the observed difference in semantic content cannot be entirely attributed to the LM performing retrieval. We thus conclude that the LM is able to abstractly interpret programs even without seeing the final outputs of the program, which is emergent behavior on out-of-distribution text. For completeness, Figure 13 plots results over the entire LM training.

B.6. Selected regression and residual plots

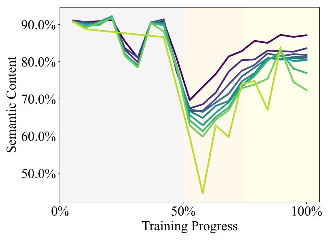
Figures 14 and 15 display selected regression and residual plots from Tables 2 and 4, respectively. Specifically, we provide the regression and residual plots corresponding to the semantic content of the current state as measured by a linear classifier. In all cases (including those omitted for brevity and not explicitly shown in the figures), the residual plots confirm a linear relationship between the semantic content and generative accuracy.



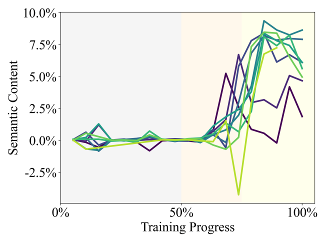
(a) Probing with a linear classifier.



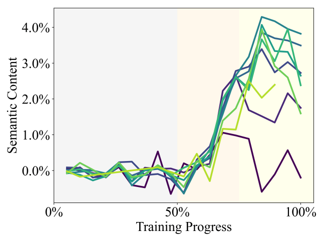
(b) Probing with a 1-layer MLP.



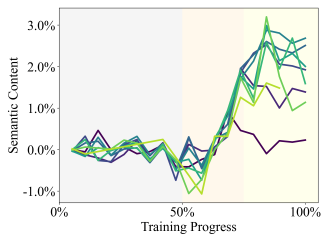
(c) Probing with a 2-layer MLP.



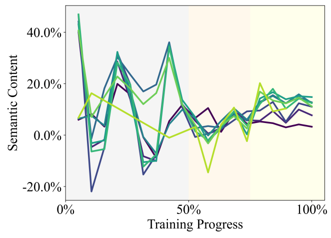
(d) Probing with a linear classifier, excess over flip semantics.



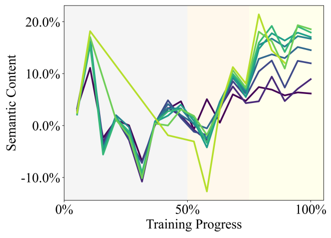
(e) Probing with a 1-layer MLP, excess over flip semantics.



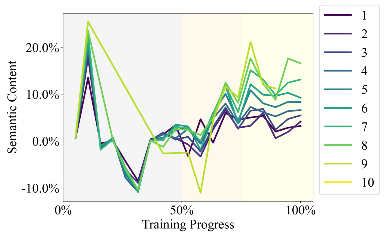
(f) Probing with a 2-layer MLP, excess over flip semantics.



(g) Probing with linear classifier, excess over adversarial semantics.

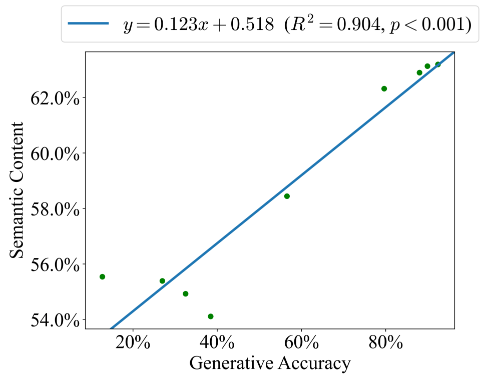


(h) Probing with a 1-layer MLP, excess over adversarial semantics.

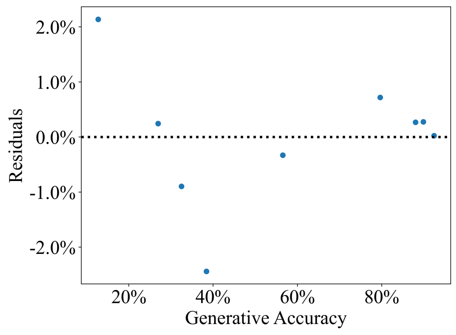


(i) Probing with a 2-layer MLP, excess over adversarial semantics.

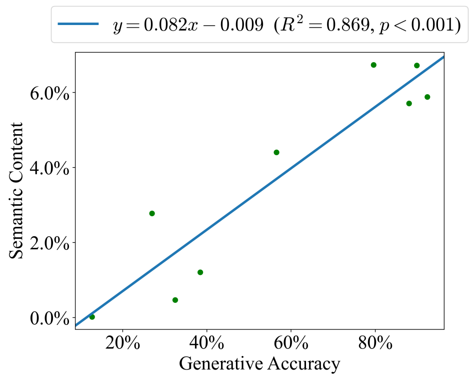
Figure 13: The semantic content on the input-only trace datasets, separated by depth over the full LM training run. Note that all plots show trends which, when aggregated over depth, exhibit a statistically significant linear correlation with the generate accuracy, except for Figures 13e and 13f.



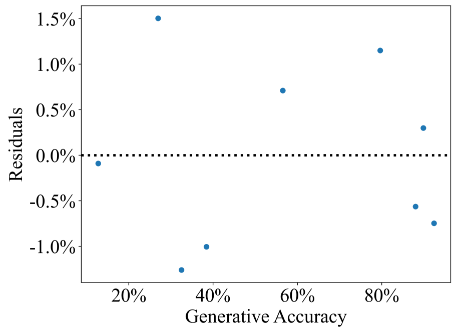
(a) Regressing semantic content vs. generative accuracy.



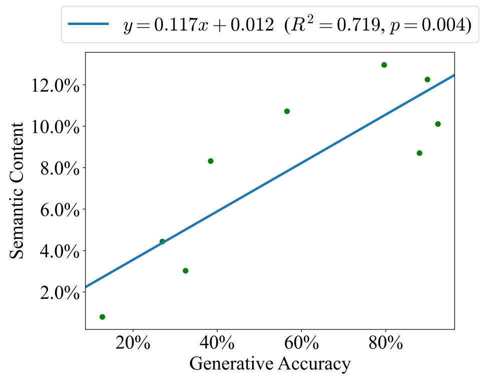
(b) Residuals for semantic content vs. generative accuracy.



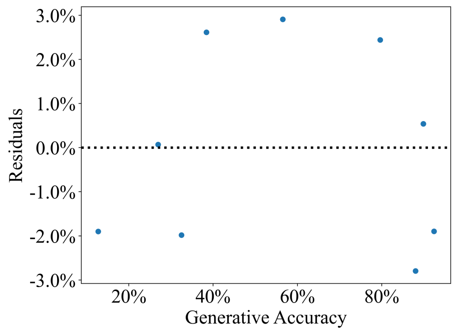
(c) Regressing excess of original over flip semantic content vs. generative accuracy.



(d) Residuals for excess of original over flip semantic content vs. generative accuracy.

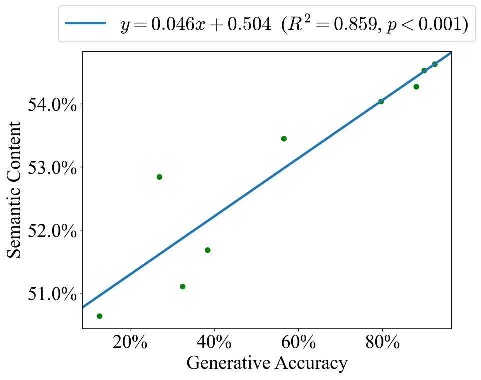


(e) Regressing excess of original over adversarial semantic content vs. generative accuracy.

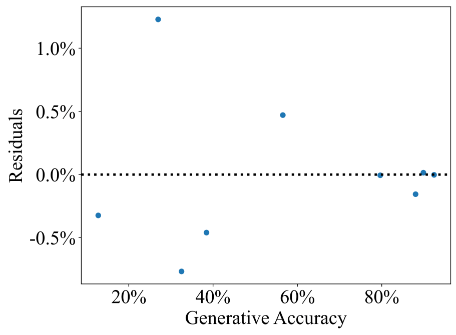


(f) Residuals for excess of original over adversarial semantic content vs. generative accuracy.

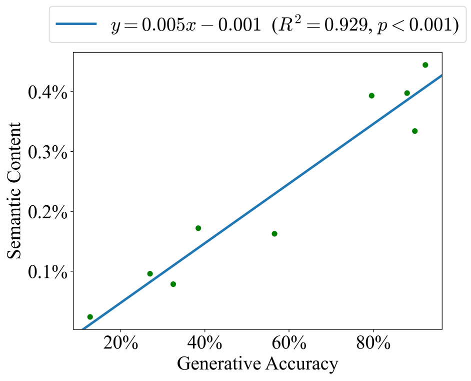
Figure 14: Regression and residual plots for semantic content (measured by a linear classifier) vs. generative accuracy over the second half of training.



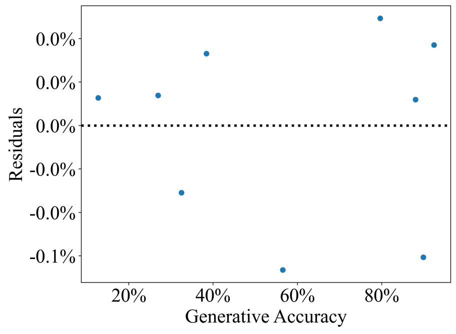
(a) Regressing semantic content vs. generative accuracy.



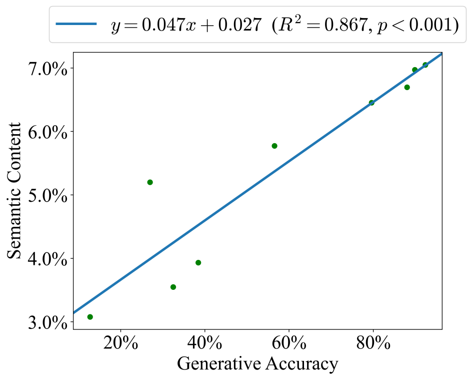
(b) Residuals for semantic content vs. generative accuracy.



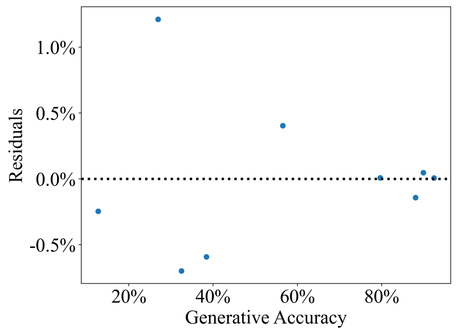
(c) Regressing excess of original over flip semantic content vs. generative accuracy.



(d) Residuals for excess of original over flip semantic content vs. generative accuracy.



(e) Regressing excess of original over adversarial semantic content vs. generative accuracy.



(f) Residuals for excess of original over adversarial semantic content vs. generative accuracy.

Figure 15: Regression and residual plots for semantic content (measured by a linear classifier on the input-only trace datasets) vs. generative accuracy over the second half of training.