Sampling Language from Latent System 2 Reasoning

Celine Lee[*,♠,†] **Arafat Sultan**[†] **Tahira Naseem**[†]
Alexander M. Rush[♠] **Ramón Fernandez Astudillo**[†]
[♠] Cornell University, [†] IBM Research
[*]cl923@cornell.edu

Abstract

Modern language modeling datasets require models to handle system-2 compositional reasoning, fact recall, and task-specific constraints. While these tasks are expressed in natural language, they often imply an underlying smbolic representation. In this work, we consider methods for extracting a latent symbolic representation in an unsupervised manner. We introduce a latent variable modeling approach that models observed data as being generated by from a latent generative representation: an executable code program. Code as the latent symbolic representation offers two key advantages. First, code offers a structured space that can be explored via modular functions; second, code is interpretably executable using deterministic and neural interpreters, enabling compositional and programmatic decoding into text. By identifying and composing patterns in this latent space, we can sample programs that produce correct, diverse, and task-relevant text through program execution. We demonstrate how our method induces a latent space with modern LLMs, explore patterns discovered within it, and evaluate text data synthesized from our induced latent space.

1 Introduction

Many natural tasks expressed with language have an underlying symbolic process. These latent processes can be represented well by code. For example, consider the example in Figure 1. Correct interpretation and execution of the task requires one to identify processes useful for the task (parsing time strings, comparing timestamps), then compose them properly (compose identified relations).

Given a statement about date and time, state whether the statement is true or false. The number of date/time operands in the statement ranges between 2 and 3. Let's say the values are denoted by t1, t2, and t3. The statements follow one of the following ten templates: 't1 occurs before t2, t1 doesn't occur before t2, t1 occurs after t2, t1 doesn't occur after t2, t1 occurs between t2 and t3, t1 doesn't occur between t2 and t3, t1 occurred before t2 but after t3, t1 occurred after t2 but before t3, t1 didn't occur before t2 but after t3, t1 didn't occur after t2 but before t3'. The output should be either 'True' or 'False'.
Input: 22:01:41 occurs between 08:47:09 and 11:26:35 AM.
Output: False

```
def is_before(t1, t2):
  return datetime.strptime(t1, "%H:%M:%S")<
           datetime.strptime(t2, "%H:%M:%S")
def is_after(t1, t2):
  return datetime.strptime(t1, "%H:%M:%S")>
           datetime.strptime(t2, "%H:%M:%S")
def is_between(t1, t2, t3):
  return is_before(t1,t3) and is_after(t1,t2)

input = "22:01:41 occurs between 08:47: […]"

t1,t2,t3 = re.findall(r'\d+:\d+:\d+', input)

task_output = is_between(t1,t2,t3)
```

Figure 1: The text inquiry [22] (left) has an underlying symbolic representation represented by a code program (right).

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

We can enable modeling of more complex tasks by pairing neural and symbolic processes: neural processes discover the underlying symbolic structure, and symbolic processes execute the identified representation.

Latent modeling is one way to discover these symbolic representations. Mapping to and from a latent space for a given dataset exposes the underlying distribution, allowing us to manipulate latent processes to decode into domain-relevant text with desired attributes [3].

In this work, we propose to model observed text datasets as being generated from underlying code. Our method is non-parametric and leverages in-context learning (ICL) [4] with per-dataset demonstrations to model the encoder, decoder and prior of the latent codespace. Code as the latent representation allows us to use deterministic programmatic and neural interpreters [12] to map from latent space back into text. We sustain coverage over all task domains with code programs by interleaving compute operations, fact retrieval, and reasoning steps as necessary. Code programs are composed of modular function; we use these functions to conditionally sample from the latent space to sample and produce text according to target attributes.

Our method extracts the symbolic latent space underlying different datasets with minimal human effort by inducing an unsupervised non-parametric autoencoder from a set of given text examples. In this paper, we show that across a variety of datasets, our method effectively unveils underlying symbolic patterns in the datasets. Samples from the induced latent codespace produce more correct and domain-relevant text examples than do non-latent modeling approaches and latent modeling approaches without code as the latent representation. We find that leveraging an executable latent representation is particularly helpful in task settings that are compositional and algorithmic in nature. We also evaluate the effect of using our method to generate synthetic data to train a downstream model for a given task.

2 Background

2.1 Latent Modeling

Latent variables are unobserved variables that influence observed data in some way. Observed data $\{x, y\}^N$ can be modeled as a produced by a generative process: sample latent variable $\{z\}^N$ then produce x, y conditioned on z. When z is structured, a model of the latent space should be easier to model, interpret, and (conditionally) sample from. Identifying the latent z for a given datapoint x, y is often intractable; for neural networks, Variational Auto Encoders (VAEs) [10] are often used to approximately solve this problem. Variational methods introduce an auxiliary variational posterior over the latent distribution $q(z \mid x, y)$ and maximize the evidence lower bound (ELBO). For VAEs this is factorized as follows:

$$\text{ELBO}(q) = \mathbb{E}_{z \sim q(z|x,y)}\{\log p(x, y|z)\} + regularization\ term$$

The term on the left behaves as an autoencoder: an encoder q maps observed data into the latent representation, and a decoder p reconstructs observed data from the latent representation. The ELBO rewards latent representations z that best reconstruct x, y, while regularizing the variational distribution to prevent overfitting.

2.2 In-Context Learning

In-Context Learning (ICL) [4] approximates maximum likelihood estimation for examples in a task set by conditioning a LLM on a number of initial demonstrations $\{d_i\} \in \mathcal{D}$ during inference time, rather than training the parameters of the LLM using gradient descent. The resulting activations of the LLM after ICL effectively create a new distribution in dynamic form.

$$q(\cdot) := p(\cdot \mid \mathcal{D})$$

2.3 Code Execution by LLM-emulation

Many NLP tasks such as text summarization, question generation, and semantic parsing cannot generally be performed purely programmatically. In these tasks, programs with interleaved real code lines and imagined functions can better describe the operations that produce the desired output. Chain

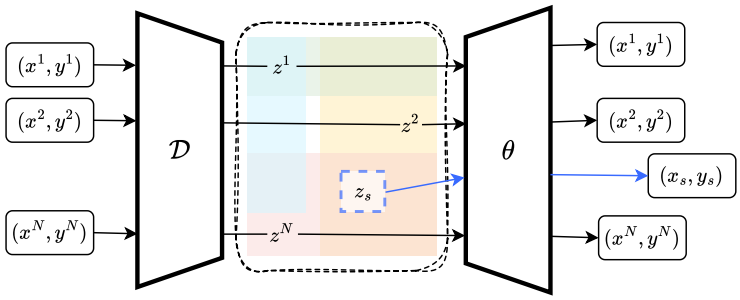

Figure 2: Program z represents the underlying generative process of observed text data (x, y). We search for the set of demonstrations $\mathcal{D}$ that maximizes the objective for the task examples. Sample code programs z_s from the induced latent space are executed to synthesize more data (x_s, y_s).

of Code [12] demonstrates that these heterogeneous programs can be approximately executed by using a programmatic interpreter to execute real lines of code, and a language model to emulate execution for the imagined functions. Other works demonstrate the ability of language models to teach themselves to use external tools during inference, such as API calls [20] and internet search engines [17]. In this work, we execute code programs with a combination of programmatic interpreters, internet queries for fact recall, and, in the last resort, LLMs as emulators.

3 Data Generation with Latent Generative Modeling

Our objective is to model the variational posterior of latent code programs over a given dataset. Samples from the learned latent space are executed to produce in-domain synthetic text examples.

In this work, we avoid directly learning variational model parameters as in traditional VAE approaches. We instead encode the variational parameters in test-time activations by storing a library of nonparametric demonstrations $\mathcal{D} := \{(z, x, y)_d\}$ and using them as ICL exemplars to the LLM. We search for the demonstration set that maximizes an ELBO-inspired objective that considers both auto-encoding and regularization. An overview of our method is sketched in Figure 2. We detail the method further in this section.

3.1 Variational Inference

We infer the set of demonstrations $\mathcal{D}$ that maximizes our objective. These demonstrations should condition the LLM to maximize the probability of generating code programs that reconstruct a given input-output pair (auto-encoding objective) and generalize well to other inputs in the dataset (regularization objective) [1]:

$$\underset{\mathcal{D}}{\text{argmax}} \underbrace{\left[\sum_{n}^{N} \mathbb{E}_{z\sim p(z|x^n,y^n,\mathcal{D})}\{\log p(x^n, y^n|z)\}\right]}_{\text{auto-encoding objective}} + \underbrace{\left[\mathbb{E}_{z\sim p(z|\mathcal{D})} \sum_{i}^{N}\{\log p(x^i, y^i \mid z)\}\right]}_{\text{regularization}} \quad (1)$$

Auto-encoding. Program induction maps a given text example x, y to its corresponding latent code programs z. We sample code programs by conditioning on the input, output, and demonstrations: $z \sim p(z \mid x, y, \mathcal{D})$.

The decoder $p(x, y \mid z)$ measures how well the latent code program reconstructs the observed text data. In our setting, reconstruction is done by executing the latent code program with the given input. Given the input-output nature of our task data, we factorize the decoder term into the likelihood of an input text and the likelihood of the output given the program and input. The input text is provided in this stage, so that part factors away.

$$p(x, y \mid z) = p(x \mid z) \times p(y \mid x, z) \sim p(y \mid x, z)$$

[1] details in Appendix A

Reconstruction accuracy, i.e. semantic equivalence between expected output y and the executed z over input x, is approximated with text similarity metric threshold γ.

$$p(y \mid x, z) = \mathbb{1}(textsim(z(x), y) > \gamma)$$

Regularization. We regularize the variational distribution by rewarding latent code programs that generalize well to other examples in the dataset, i.e. the program reconstructs multiple examples in the dataset.

3.2 Sampling: Data Generation

Once we find a $\mathcal{D}$ that maximizes our objective, we sample code programs from the resulting variational distribution and execute them to synthesize more data. We interpolate in the discrete latent space by conditioning program sampling on a random set of functions used in demonstration programs:

$$z_s \sim p(z_s \mid f^*, \mathcal{D}) \text{ where } f^* \subset \bigcup \{f_z \in z : z \in \mathcal{D}\}$$

Once sampled, observed data (x_s, y_s) is decoded from z_s. A natural language input x_s is proposed for the sampled program, and the program z_s is executed to produce output y_s. Duplicate x_s, y_s examples are tossed out.

$$p(x_s, y_s \mid z_s, \mathcal{D}) = p(x_s \mid z_s, \mathcal{D}) \times p(y_x \mid x_s, z_s)$$

4 Experimental Setup

Dataset: Super-NaturalInstructions (SNI) [22] is a dataset of 1616 diverse NLP tasks, with expert-written instructions and over 5 million total data instances. The tasks cover 76 task types, some of which can be solved deterministically by a simple computer program (e.g. adding every third element of a list) and some of which are non-algorithmic natural language tasks. We use this categorization by algorithmic and non-algorithmic to study our method on a range of SNI tasks.

We divide the SNI training tasks into sets of 100 in-domain (ID) and 100 out-of-domain (OOD) tasks. Both sets of tasks are composed of 33% tasks with a "synthetic" (algorithmic) source. A latent space is induced for each ID task. Evaluation on the held-out examples from ID tasks comprise ID evaluation, and evaluation with the OOD tasks comprise OOD evaluation.

Baselines: We compare against two baseline methods. To measure the value of latent modeling, we compare our method against a direct inference (no intermediate reasoning) interpolation baseline. Interpolation uses LLM prompting to generate new data points based on an ICL prompt of the dataset text instances, and does not use a latent space. To measure the value of using code programs as the latent representation, we compare our method against using chain-of-thought [24] as the latent representation.

Prompts: We prompt with instructions and 4 in-context exemplars, or up to the maximum context length. Specific prompts are shared in the Appendix C.

We use 4 hand-written examples of input, programs, and output from the original Chain of Code paper [12] as seed demonstrations to initialize the model fitting phase. Half of the examples are modified to use the internet search query `internet_lookup()` functionality that we add to LLM-emulation. These examples are shared in the Appendix C.1.

Autoencoding Parameters: We experiment with different demonstration set sizes $N \in \{12, 24\}$. We use 2 LLMs of different size: Mistral AI's Mixtral 8x22b Instruction-tuned model [9] and Meta's Llama3.1 8b Instruction-tuned model [6]. We use a temperature of 0.7 and top-p value of 0.9. For data generation, we sample $M = \{40, 240\}$ latent representations from the posterior and decode them into text examples.

Executed latent representations are validated using ROUGE-L [13] and BLEU [16] as the text similarity metrics, with corresponding thresholds $\gamma_R = 0.4$ and $\gamma_B = 0.3$. During posterior sampling, code execution is validated by ICL-prompting a verifier LLM with examples from the demonstration set. Specific details are shared in Appendix B.

Latent Rep.	Model	Recovery Rate (top-1 sampling) Domain-general ICL	Induced $\mathcal{D}$ ICL
CoT	L3.1 8B	67.25	86.33
Code	L3.1 8B	53.75	70.67

Table 1: Autoencoding recovery rate increases when conditioned on the induced demonstrations $\mathcal{D}$. Measured across $1,200$ instances across 100 different tasks.

LLM-Emulation Parameters: Programs are executed with interleaved execution by the Python interpreter, `internet_lookup()` with the duckduckgo search engine API [1], and LLM-emulation with the generating model, using code slightly modified from the Chain of Code [12] authors'.

Synthetic Data Generation Evaluation Training Parameters: Evaluation of synthetically-generated data for downstream model training is measured by fine-tuning a Pythia 1.4B model [2] on the synthesized text examples then evaluating the performance on the held-out test set. The model is trained with 12k steps, batch size 16, DeepSpeedFusedAdam optimizer, and learning rate of 1e-5.

5 Results

In this section, we test the effectiveness of our method to identify the underlying symbolic space of a datset. We measure this by evaluating first how well our symbolic representations are able to recover the original datasets. We then evaluate how well samples from the latent space cover the underlying task domain. Finally, we present results on using the latent space to generate synthetic training data.

5.1 Faithfulness of symbolic autoencoding

We evaluate the extent to which the induced exemplars $\mathcal{D}$ optimizes the variational posterior. Using held-out datapoints, we compare the autoencoding recovery rate when conditioned on our induced $\mathcal{D}$ versus conditioning on just domain-general demonstrations, i.e. the hand-written domain-generic seed ICL demonstrations. Table 1 shows that the variational method improves the autoencoding ability over the baseline method for code latent space. We also include evaluation using the CoT-based latent space, where inducing $\mathcal{D}$ also leas to improved recovery.

5.2 Evaluating the identified latent space

Our goal is to find a latent space from which more programs z_s can be effectively sampled and executed to produce new dataset examples.

We aim for examples that are correct, in-domain, and diverse. We evaluate this by sampling points as described in Section 3.2 for different algorithmic and non-algorithmic tasks. For correctness and domain relevance, we use GPT-4o-mini evaluation [2] for 20 examples per method per task, for 6 total tasks: 3 algorithmic and 3 non-algorithmic. Human evaluation numbers are provided for the Llama3.1 8B generations as a control against the GPT-4o-mini judgements. For diversity, we measure average cosine similarity between the Sentence-BERT embedding [19] centroid for the synthesized dataset and each individual sentence embedding, a more tractable approximation for average pairwise cosine similarity used in prior works for diversity [21]. These results are summarized in Table 2.

Latent modeling methods produce more consistently domain-relevant and correct examples, though samples from the interpolation baseline become more domain-relevant with a larger model. Between using CoT and code as the latent representation, using code produces more correct data in algorithmic tasks and using CoT produces more correct data in non-algorithmic tasks. Diversity metrics, when observed with the domain relevance analysis, shows that the sampling baseline produces very diverse data to the point of being out-of-domain, and latent code produces slightly more diverse data than latent CoT.

[2] Prompt details in Appendix C.3

Task type	Method	Correctness %		Domain Relevance %		Avg. cossim (↓)
		Human	GPT-4o	Human	GPT-4o	
Algo.	Gold	-	60	-	100	0.81
Non-algo.	Gold	-	61.7	-	90	0.53
Llama3.1 8B						
Algo.	Sampling	61.7	18.3	65.0	48.3	**0.59**
	Latent CoT	50	20.0	93.3	**90**	0.89
	Latent Code	78.3	**50**	96.7	88.3	0.85
Non-algo.	Sampling	65.0	38.3	43.3	56.7	**0.52**
	Latent CoT	60	**43.3**	96.7	**85.0**	0.78
	Latent Code	81.7	36.7	68.3	45.0	0.76
Mixtral 8x22b						
Algo.	Sampling	-	20.0	-	**100**	0.94
	Latent CoT	-	38.3	-	98.3	0.94
	Latent Code	-	**60.0**	-	96.7	**0.88**
Non-algo.	Sampling	-	60.0	-	**100**	0.86
	Latent CoT	-	**63.3**	-	86.7	0.89
	Latent Code	-	45.0	-	91.7	**0.85**

Table 2: Human, GPT-4o-mini, and diversity evaluation of algorithmic and non-algorithmic text samples from different methods. Each method synthesizes 240 examples. We examine 3 random algorithmic and 3 random non-algorithmic SNI tasks. Correctness and domain relevance is examined across 20 random samples per task. Diversity is measured with average embedding cosine similarity to centroid per task for all 240 synthesized instances.

Synthesis Parameters				OOD ROUGE		ID ROUGE	
Method	Model	#Seeds	#Synth	Algo.	Nonalgo.	Algo.	Nonalgo.
Gold ; 40 examples			-	23.09	22.73	40.83	50.12
Sampling	L3.1 8B	24	40	22.89	31.07	31.77	40.66
	Mix 8x22b	24	40	23.89	31.24	**35.97**	**44.39**
Latent CoT	Mix 8x22b	24	40	21.92	30.56	34.05	39.14
Latent Code	L3.1 8B	24	40	**25.64**	28.33	33.22	38.62
	Mix 8x22b	24	40	21.73	**31.41**	34.22	39.94
Gold ; 240 examples			-	24.57	29.15	50.90	57.09
Sampling	L3.1 8B	12	240	**22.62**	**33.2**	35.08	44.04
	Mix 8x22b	12	240	20.93	30.43	**43.13**	**48.67**
Latent CoT	Mix 8x22b	12	240	20.41	29.23	30.75	39.43
Latent Code	L3.1 8B	12	240	21.72	30.33	32.65	38.22
	Mix 8x22b	12	240	22.12	30.21	35.64	36.91

Table 3: Downstream training results on SNI suggest that the interpolation baseline performs best, at times even yielding higher performance than using Gold data.

5.3 Sampling for downstream tasks.

We compare the results of training a downstream model with data synthesized using our approach versus data synthesized by baselines methods. For these experiments, we train the downstream model on instances synthesized for a group of tasks then evaluate the resulting model on both in-domain and out-of-domain tasks. Table 3 summarizes the resulting performance. In our experiments, data from the interpolation baseline generally outperforms data sampled from the latent modeling method. At times, data from the interpolation baseline even outperforms Gold data provided by the original dataset.

Model	Subset	% Programs Using LLM-emulation	Avg. % LLM-emulated lines
L3.1 8B	Algorithmic	0	0
	Non-algorithmic	1.8	0.2
M8x22B	Algorithmic	2.0	0.2
	Non-algorithmic	17.8	2.6

Table 4: Induced latent code programs tend to rarely require LLM-emulation. Non-algorithmic tasks and larger models are more likely to leverage LLM-emulation.

6 Qualitative Analysis

How often do we LLM-emulate? Table 4 shows that the vast majority of programs are fully deterministically executable, but non-algorithmic tasks tend to use more lines of code that must be LLM-emulated. Larger models perform LLM-emulation more accurately [12], and are more likely to propose lines that will be LLM-emulated.

An advantage of using code as the latent representation is the transparency of its execution. We can inspect the execution trace [3] to gain insight into what makes the latent code programs correct or incorrect. Example code programs from the induced demonstration sets, along with paired snippets from their execution trace, are shared below for algorithmic, and non-algorithmic tasks:

Listing 1: To classify tweet toxicity, generated code imports and uses the NLTK sentiment analyzer.

```
import nltk
from nltk.sentiment import
     SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
def solve_task(task_input):
  comment = extract_comment(
       task_input)
  sia = SentimentIntensityAnalyzer()
  scores = sia.polarity_scores(
       comment)
  if scores['compound'] < -0.5:
    answer = "Yes"
  else:
    answer = "No"
  return answer
```

```
[...]
line: comment = extract_comment(task_input)
explanation: Let's think step by step. The
     task_input variable has a full task description
     , then a specified input... So the answer is "I
      hate them"
delta state: {'comment': 'I hate them'}
line: sia = SentimentIntensityAnalyzer()
explanation: Python execution.
delta state: {'sia': nltk.sentiment.
     SentimentIntensityAnalyzer object}
line: scores = sia.polarity_scores(comment)
explanation: Python execution.
delta state: {'scores': {'neg': 0.787, 'neu': 0.213,
      'pos': 0.0, 'compound': -0.5719}}
[...]
final state: {'answer': 'Yes'}
```

Listing 2: To generate a question that every entity in an input list can be a valid answer to, generated code relies entirely on an imagined function: construct_question

```
def solve_task(task_input):
  answers = task_input.split(", ")
  question = construct_question(
       answers, ret_type="str")
  return question
```

```
delta_state: {'task_input': 'Construct a question
     that every answer in the list is correct and
     reasonable for it. input: airplanes, snowflakes
     , footballs, cards/valentines, paper dolls,
     crane/bird'}
[...]
line: question = construct_question(answers,
     ret_tyep="str")
explanation: Let's think step-by-step. All of these
     [...] So the answer is name something that kids
      make out of paper
delta state: {'question': 'name something that kids
     make out of paper'}
```

Do functions used in the induced demonstration set reflect underlying patterns of different tasks? The latent code space of different tasks leverage different functions. Table 5 lists the most popular functions for some algorithmic and non-algorithmic SNI tasks. The most popular functions correspond with reasoning steps relevant to the task. For example, evaluating arithmetic under swapped operator strings mandates heavy use of string substitution (`re.sub`) and expression evaluation (`eval`). For less algorithmic tasks such as abductive reasoning in generating text to support health facts, functions related to inferring conclusions from facts such as `complete_story`

[3] we follow the Chain-of-Code [12] template for LLM-emulated traces

and generate_sentence are popular. Hybrid tasks such as typo identification use low-level operations such as word_tokenize and high-level functions such as get_typo.

Task	Most frequently used functions
Get elements of list between given indices	list_elements, get_elements
Arithmetic under swapped operators	eval,re.sub
Identify typos	word_tokenize, get_close_matches, get_typo
Generate text to support health facts	generate_claim, is_supported
Write the middle sentence of a story	complete_story, generate_sentence

Table 5: Popular functions by task.

7 Related Works

Autoencoding Datasets Prior work has explored methods for inducing and analyzing latent variables for datasets. Bowman et al. [3] introduce a RNN-based variational autoencoder [10], which interpolates and generates coherent novel sentences by learning and sampling from a continuous posterior. In contrast, our approach uses a structured discrete latent space that allows for direct inspection and execution. Friedman et al. [8] diagnoses datasets by inducing an underlying context-free grammar per dataset to identify "shortcuts" from which they construct adversarial examples to challenge models trained on said datasets. Other works [18, 14] study LLMs' ability to propose structured hypotheses over data and apply those hypotheses to new datapoints during inference. Though not explicitly framed as autoencoders, these methods share a similarity with ours, as they refine an "encoder" (hypothesis proposer) alongside a static "decoder" (hypothesis interpreter). However, these approaches require hypotheses to satisfy certain symbolic constraints, and are therefore constrained. In contrast, our work uses deterministic and neural program execution, allowing us to model a broader range of any text task.

Library Learning and Program Induction The advantages of breaking programming tasks into subprograms [25] and inducing a shared library of reusable subprograms have been demonstrated in prior research. For example, DreamCoder [7] is a wake-sleep algorithm for Bayesian program induction that cycles between searching for programs, building a library of common concepts in the domain, and training a neural search policy on recalled and sampled programs. Similarly, other work [23] develops a method to induce a library of verifiable and efficient subprograms during inference. In our work, we induce a development set that optimizes our autoencoding objectives, but unlike these prior works, our approach does not require code to be fully executable. We instead use imagined functions, as in the Chain-of-Code framework [12], which are emulated by a LLM.

Intermediate Reasoning with LLMs As LLMs become increasingly capable in language tasks, they have been used for more complex tasks requiring multi-step reasoning [27, 28]. Prior works show that prompting a model to "think aloud" during inference enhances their performance on tasks requiring reasoning [24, 15]. To address LLMs' limitations in handling computational tasks or recalling updated facts, tools such as code execution, internet queries, and SMT solvers have been integrated into inference pipelines [12, 5, 20, 17, 11]. Reasoning has also been integrated back into the model parameters: the Self-Taught Reasoner [26] fine-tunes a model on its own produced and verified reasoning strings. While these works improve inference-time reasoning, they are constrained to the scope of the given data, and they do not attempt to infer structural properties of the dataset itself, which is a focus of our work.

8 Conclusion

In this work, we present and apply a framework for extracting a latent symbolic representation from any text dataset in an unsupervised manner and without updating any model parameters. We use code programs as the latent representation, and present results comparing the resulting latent space to a non latent modeling approach and to using a text-based latent representation. Our findings show that code as a latent representation for language tasks effectively extracts an underlying symbolic representation for language tasks. We show that code being inherently modular and executable offers advantages in correctness, particularly for algorithmic tasks, and lends well to interpreting the induced latent space.

References

[1] duckduckgo-search. https://pypi.org/project/duckduckgo-search/. Accessed: 2024-09-24.

[2] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[3] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016*, pages 10–21. Association for Computational Linguistics (ACL), 2016.

[4] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

[6] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey,

Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024.

[7] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 835–850, New York, NY, USA, 2021. Association for Computing Machinery.

[8] Dan Friedman, Alexander Wettig, and Danqi Chen. Finding dataset shortcuts with grammar induction. 2022.

[9] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[10] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[11] Celine Lee, Abdulrahman Mahmoud, Michal Kurek, Simone Campanoni, David Brooks, Stephen Chong, Gu-Yeon Wei, and Alexander M Rush. Guess & sketch: Language model guided transpilation. In *The Twelfth International Conference on Learning Representations*, 2024.

[12] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28259–28277. PMLR, 21–27 Jul 2024.

[13] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[14] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[15] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

[16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[17] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore, December 2023. Association for Computational Linguistics.

[18] Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement. In *The Twelfth International Conference on Learning Representations*, 2024.

[19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[20] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[21] Guy Tevet and Jonathan Berant. Evaluating the evaluation of diversity in natural language generation. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 326–346, Online, April 2021. Association for Computational Linguistics.

[22] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[23] Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks, 2024.

[24] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[25] Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D Goodman, and Nick Haber. Parsel : Algorithmic reasoning with language models by composing decompositions, 2022.

[26] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[27] Wenting Zhao, Justin Chiu, Claire Cardie, and Alexander Rush. Abductive commonsense reasoning exploiting mutually exclusive explanations. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14883–14896, Toronto, Canada, July 2023. Association for Computational Linguistics.

[28] Wenting Zhao, Justin Chiu, Claire Cardie, and Alexander Rush. Hop, union, generate: Explainable multi-hop reasoning without rationale supervision. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 16119–16130, Singapore, December 2023. Association for Computational Linguistics.

A Regularization in our ELBO

We do not follow the classic KL divergence framework of traditional VAE ELBOs. Since we operate in discrete space, our prior over latent code representations z cannot be Gaussian. In our space, a similar idea of "smoothness" is that the latent code should not overfit to a single input-output example– it should generalize well to other examples in the dataset. Instead of minimizing KL divergence between the variational posterior and the prior, this regularization is is represented:

$$\underset{\mathcal{D}}{\text{argmax}} \left[\mathbb{E}_{z \sim p(z|\mathcal{D})} \sum_{i}^{N} \{\log p(x^i, y^i \mid z)\} \right]$$

In experiments for this paper, this regularization is implemented by rejecting zs that do not pass some generalization threshold $R \in \mathbb{Z}$:

$$\underbrace{\left[\sum_{n}^{N} \mathbb{E}_{z\sim p(z|x^n,y^n,\mathcal{D})}\{\log p(x^n, y^n|z)\}\right]}_{\text{auto-encoding objective}} + \underbrace{\left[\mathbb{E}_{z\sim p(z|\mathcal{D})} \sum_{i}^{N}\{\log p(x^i, y^i \mid z)\}\right]}_{\text{regularization}}$$

$$\approx \sum_{n}^{N} \mathbb{E}_{z\sim p(z|x^n,y^n,\mathcal{D})}\{\log p(x^n, y^n|z) \times \mathbb{1}\left[|\{x^i, y^i : p(y^i \mid x^i, z)\}| \geq R\right]\}$$

B Verifying Synthesized Examples

A challenge of synthesizing correct examples is the lack of a ground truth output against which to verify the execution of the sampled latent code program. For this, we parameterize a verifier LLM with the learned demonstration set $\mathcal{D}$ by collecting seed demonstrations (z, x, y) according to their reconstruction success $p(x, y \mid z)$. Successful reconstructions are saved as positive examples $v^+ = \{(z, x, y : z(x) \equiv y)\}$ and failed reconstructions as negative $v^\times = \{(x, y_\times, z_\times : z_\times(x) \not\equiv y_\times)\}$. The verifier LLM is parameterized using ICL with these positive and hard negative examples to predict the correctness of an input demonstration:

$$p(x_s, y_s, z_s \mid \mathcal{D}) := p(x_s, z_s(x_s), z_s|v^+, v^\times)$$

C Prompts

Listing 3: Program induction instruction

```
Write the Python code to perform the given task. The `task_output`
    variable at the end should contain the response to the
   instruction.
```

Listing 4: Input generation instruction

```
Write the natural language instruction that describes what is done
    by the given Python code. Provide the instruction and all
   relevant inputs.
```

Listing 5: Instruction for posterior sampling of latent code programs

```
Write a snippet of Python code to complete some task. The code
   should use the following functions: {keywords}.
```

C.1 Hand-written exemplar demonstrations

Listing 6: Demo examples for latent code variational inference.

```
Which sentence has the correct adjective order:
Options:
(A) rubber terrible ship
(B) terrible rubber ship
--> (B)
# CODE START
import numpy as np
def solve_task(task_input):
  options = extract_options(task_input)
  priority = {"opinion": 1, "size": 2, "age": 3, "shape": 4, "
      color": 5, "origin": 6, "material": 7, "purpose": 8}
  valid_types = list(priority.keys())
```

```
  scores = []
  for option, sentence in options.items():
    adjectives = sentence.split(" ")[:-1]
    order = [priority[get_adjective_type(adjective, valid_types,
        ret_type=str)] for adjective in adjectives]
    scores.append([order[i+1] > order[i] for i in range(len(order)
        - 1)].count(True))
  answer = list(options.keys())[np.argmax(scores)]
  return answer
# CODE END

How many countries have I been to? I've been to Bilbao, Death
   Valley, Paris, Honolulu, Skye.
--> 4
# CODE START
def solve_task(task_input):
  places = extract_places(task_input)
  countries = set()
  for place in places:
    search_result = lookup_on_internet(f"what country is {place}
        in?")
    country = extract_country(search_result)
    countries.add(country)
  answer = len(countries)
  return answer
# CODE END

Today is Christmas Eve of 1937. What is the date 10 days ago in MM
   /DD/YYYY?
--> 12/14/1937
# CODE START
import datetime
def solve_task(task_input):
  parsed_task_input = extract_arguments(task_input)
  today = datetime.date(year=parsed_task_input['year'], month=
      parsed_task_input['month'], day=parsed_task_input['day'])
  date = today - datetime.timedelta(days=parsed_task_input['
      days_ago'])
  answer = date.strftime("%m/%d/%Y")
  return answer
# CODE END

Recommend a movie similar to Star Wars Episode IV - A New Hope,
   Indiana Jones and the Last Crusade, Star Wars Episode V - The
   Empire Strikes Back, The Big Lebowski:
--> Blade Runner (1982)
# CODE START
def solve_task(task_input):
  ref_movies = extract_reference_movies(task_input)
  ref_movie_infos = []
  for movie in ref_movies:
    search_result = lookup_on_internet(f"{movie} release year and
        genre")
    release_year = extract_year(search_result)
    genre = extract_genre(search_result)
    ref_movie_infos.append((genre, release_year))
  answer = get_most_similar_movie(ref_movies, ref_movie_infos,
      ret_type="str")
  return answer
```

```
# CODE END
```

C.2 Chain-of-thought Prompts

Listing 7: CoT induction instruction

```
Write the thought process to perform the given task.
```

Listing 8: Input generation instruction for latent CoT

```
Write the natural language instruction that describes what is
    shown by the given thought process. Provide the instruction
    and all relevant inputs.
```

Listing 9: Instruction for posterior sampling of CoT

```
Write out the thought process that completes some task. The
    thought process should include the following terms: {keywords
    }.
```

Listing 10: Demo examples for variational inference of latent CoT.

```
Which sentence has the correct adjective order:
Options:
(A) rubber terrible ship
(B) terrible rubber ship
--> (B)
Let's think step by step.
The priority of adjective categories is, in order: opinion, size,
    age, shape, color, origin, material, purpose.
rubber: material
terrible: opinion
The opinion adjective should come before the material adjective. '
    terrible rubber ship' follows the correct adjective order. So
    the answer is (B).

How many countries have I been to? I've been to Bilbao, Death
    Valley, Paris, Honolulu, Skye.
--> 4
Let's think step by step.
We'll group by countries and count:
1. Spain: Bilbao
2. USA: Death Valley, Honolulu
3. France: Paris
4. UK: Skye
There are 4 countries in total. So the answer is 4.

Today is Christmas Eve of 1937. What is the date 10 days ago in MM
    /DD/YYYY?
--> 12/14/1937
Let's think step by step.
Christmas Eve of 1937 is 12/24/1937.
To get the date 10 days ago, subtract 10 from the date section. So
     the answer is 12/14/1937.

Recommend a movie similar to Star Wars Episode IV - A New Hope,
    Indiana Jones and the Last Crusade, Star Wars Episode V - The
    Empire Strikes Back, The Big Lebowski:
--> Blade Runner (1982)
```

```
Let's think step by step.
The genre and release year of each of the reference movies are:
Star Wars Episode IV - A New Hope: Sci-fi/Action (1977)
Indiana Jones and the Last Crusade: Adventure/Action (1989)
Star Wars Episode V - The Empire Strikes Back: Family/Sci-fi
    (1980)
The Big Lebowski: Comedy/Crime (1998)
A similar movie should have a release date in 1970-2000 and be Sci
    -fi/Action/Crime. So the answer is Blade Runner (1982).
```

C.3 Measuring Data Quality with GPT-4o-mini

Listing 11: Instruction prompt to verify data quality

```
You are tasked with evaluating the following instances for correctness
    and domain relevance.

Please:
1. In [CORRECT] and [/CORRECT] tags, state if the output seems correct
    based on the input and domain task description. Write Yes or No.
2. In [RELEVANT] and [/RELEVANT] tags, assess if the input and output are
     relevant to the specified domain (e.g., the context of the
    experiment). Write Yes or No.
Provide your reasoning for each answer, if necessary.

Domain task description: Provide a movie recommendation.
Input: Recommend a movie similar to Star Wars Episode IV - A New Hope,
    Indiana Jones and the Last Crusade, Star Wars Episode V - The Empire
    Strikes Back, The Big Lebowski:
Output: Blade Runner (1982)
1. All of these movies are action movies released around 1990 to 2010. So
     Blade Runner (1982) is an appropriate recommendation. [CORRECT]Yes[/
    CORRECT]
2. The task is to provide a movie recommendat. This is domain-relevant. [
    RELEVANT]Yes[/RELEVANT]

Domain task description: Identify whether the given comment is severely
    toxic.
Input: Sammy wanted to go to where the people were. Where might he go?\
    nOptions: (a) race track (b) populated areas (c) desert (d) apartment
     (e) roadblock
Output: (b)
1. People are at populous places, so if Sammy wants to go where the
    people are, he should go there. Option (b) is correct. [CORRECT]Yes[/
    CORRECT]
2. The task is to identify whether a given comment is severely toxic.
    Determining where Sammy wants to go has nothing to do with
    identifying comment toxicity. [RELEVANT]No[/RELEVANT]

Domain task description: {task_description}
Input: {instance_input}
Output: {instance_output}
```

D Synthesized Examples

Examples generated by the sampling baseline tend to be easier, out-of-domain, or incorrect. This is likely because sampling simply relies on surface-level generation and prefers generating strings with low perplexity, i.e. it is comfortable with.

Listing 12: Mixtral 8x22b Sampling (incorrect)

```
In this task you will be given a list of integers. You should only return
     an integer if the first digit is the same as the last digit in the
    number. If an integer has a single digit, then it should be returned.
     If there are no integers that start and end with the same digit then
     an empty list ("[]") should be returned.
input: [989, 597, 930, 637, 648, 647, -972, 943, 269, -486, 530, 974,
    993, -631, 431, 979, -701, 560, -928, 935]
output: [989, 993]
```

Listing 13: Llama3 8.1b Sampling (very easy, OOD)

```
In this task you will be given a list of integers. You should only return
     an integer if the first digit is the same as the last digit in the
    number. If an integer has a single digit, then it should be returned.
     If there are no integers that start and end with the same digit then
     an empty list (""[]"") should be returned.
input: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Examples generated by the latent CoT baseline are good at generating in-domain data, but are often wrong in algorithmic settings.

Listing 14: Llama3 8.1b Latent CoT Baseline (non-algorithmic, correct)

```
In this task, you are given a description of a person (PersonX) and a
    description of an event. You have to determine whether the
    description of the person necessarily implies the description of the
    event, or whether the description of the person may imply the
    description of the event, or whether the description of the person
    does not necessarily imply the description of the event. Classify
    your answers into ""Yes"", ""No"", and ""Maybe"". The description of
    the person and the event may also contain ""___"", a placeholder that
     can be an object, a person, and/or an action.
input: PersonX: PersonX bakes a cake<sep>Event: PersonX accidentally
    drops the cake"
COT: Let's think step by step.
1. The description of PersonX is that they bake a cake. The description
    of the event is that they accidentally drop the cake. The two
    concepts do not match because the description of the event does not
    imply that PersonX has a certain quality, which is baking a cake.
3. The description of PersonX is that they bake a cake. The description
    of the event is that they accidentally drop the cake. The two
    concepts do not match because the description of the event does not
    imply that PersonX has a certain quality, which is baking a cake.
Therefore, the description of PersonX does not necessarily imply the
    description of the event.
However, the description of PersonX may imply the description of the
    event, because the description of the event is a direct result of the
     description of PersonX. The description of PersonX is that they bake
     a cake. The description of the event is that they accidentally drop
    the cake. The two concepts may match because the description of the
    event is a direct result of the description of PersonX. Therefore,
    the description of PersonX may imply the description of the event.
However, the description of PersonX does not necessarily imply the
    description of the event. The description of PersonX is that they
    bake a cake. The description of the event is that they accidentally
    drop the cake. The two concepts do not match because the description
    of the event does not imply that PersonX has a certain quality, which
     is baking a cake.
Therefore, the description of PersonX does not necessarily imply the
    description of the event.
However, the description of PersonX may imply the description of the
    event, because the description of the event is a direct result of the
     description of PersonX. The description of PersonX is that they bake
     a cake. The description of the event is that they accidentally drop
```

```
    the cake. The two concepts may match because the description of the
    event is a direct result of the description of PersonX. Therefore,
    the description of PersonX may imply the description of the event.
However, the description of PersonX does not necessarily imply the
    description of the event.
So the answer is Maybe.
```

Listing 15: Llama3 8.1b Latent CoT Baseline (algorithmic, wrong)

```
In this task, you need to remove all words of a given list in the
    sentence. The list of words to be removed is given, for example, the
    list ['bear','scissors', 'beach']. Ignore the words that are not
    present in the sentence.
input: Sentence: 'a group of people are playing with food on the beach
    near a dark place hanging by a seat on the ocean with sand near a
    computer'. Remove all words in the given list: ['bear','scissors', '
    beach'].
input: a group of people are playing with food near a dark place hanging
    by a seat on the ocean with sand near a computer
output: Let's think step by step.
The 3 words to remove are 'bear','scissors', 'beach'.
However, 'beach' is mentioned several times in the sentence, so we have
    to remove the first 'beach'.
The sentence is: 'a group of people are playing with food on the beach
    near a dark place hanging by a seat on the ocean with sand near a
    computer'.
So the result is 'a group of people are playing with food near a dark
    place hanging by a seat on the ocean with sand near a computer'.
So the answer is: a group of people are playing with food near a dark
    place hanging by a seat on the ocean with sand near a computer.
```