In-Context Learning for Esoteric Programming Languages: Evaluating and Enhancing LLM Reasoning Without Fine-Tuning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large Language Models (LLMs) have revolutionized mainstream software development, yet their ability to generalize to esoteric languages — who may have small or no representation in the training corpus —remains poor. Programming in esoteric languages tests a model's capacity to infer novel grammar and leverage nontrivial reasoning capabilities in utilizing the documentation. To quantify these effects, we evaluate both open and closed-source LLMs on code generation and language identification tasks across four esoteric languages—Minipy, Pyth, Rhokell, and 0815—and compare traditional prompt-based methods to agentic coding IDEs. Our findings reveal that LLMs can now generate some correct code in these languages when provided with documentation and sparse examples; however, performance remains far below that of similar models in common programming languages.
Furthermore, we introduce a novel in-context augmentation strategy in which LLMs first generate solutions, which are then manually verified and re-inserted as examples into subsequent prompts. Our results indicate that strategically embedding just a few analogous problems can yield large accuracy improvements without any model retraining. Our findings show that this "self-scaffolding" approach can boost performance on coding benchmarks: inserting Deepseek's verified EsoEval solutions raised EsoEval accuracy on Pyth from 16.67% to 30.82 %, while HumanEval accuracy on Minipy jumped from 51% to 65%. We offer this as a flexible alternative to costly fine-tuning, paving the way for rapid adaptation of LLMs to highly specialized, emerging, or other low data domains.

1 Introduction

Large Language Models (LLMs) pretrained on massive amounts of text and code data have demonstrated promising performance across various code generation tasks. As these models become increasingly prevalent, a key application area is the generation of code in specialized domains. New languages are constantly being developed to better address things like performance, security, or ease of writing specific types of programs. Given the high cost of fine-tuning LLMs, in-context learning—leveraging instructions and examples provided within prompts—has emerged as the preferred method for adapting these models to tasks and domains that were not encountered during training.

Previous studies have employed in-context demonstrations to prompt LLMs to generate code that interfaces with external, task-specific library functions [Gupta and Kembhavi, 2023]. Additionally, [Patel et al., 2024] observed that LLMs exhibit a strong capability to understand and utilize novel

*Equal contribution

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

code libraries based solely on in-context information and showed that LLMs could learn an entirely unfamiliar programming language known as Isabelle despite minimal available data.

Unlike mainstream programming languages, which benefit from extensive online documentation and training data, esoteric programming languages present a challenging test bed where models have had far less data to learn from but can rely on full documentation provided at run-time. Building on the preliminary observation that LLMs can learn new programming languages using only in-context demonstrations, we aim to explore several key questions. Specifically, we investigate: which esoteric languages can LLMs effectively handle? How far can these languages deviate from conventional programming paradigms while still being learned effectively? Do smaller, open-source models exhibit similar capabilities?

Whereas previous studies focus on a single API or language, we evaluate LLMs across a diverse range of esoteric languages (Minipy, Pyth, Rhokell, 0815) that vary in syntax and online footprint, and under two benchmarks (HumanEval, EsoEval) of differing complexity. We introduce a self-scaffolding procedure: model-generated solutions are manually verified and then re-inserted as in-context examples, to boost performance without retraining, providing a lightweight adaptation method that complements these prior techniques.

2 Methodology

2.1 Model Familiarity

We conducted an assessment of our model's familiarity with our chosen esolangs by asking the models to describe the language and to identify code examples. We evaluated the model's descriptions of the programming languages by hand, as summarized in Figure 1. For the code identification task none of the models were able to correctly identify the programming language from the code examples.

2.2 Measuring Esolang Obscurity

We first quantify how obscure these esoteric languages are by gathering data on two key indicators: (1) the number of search engine results containing references to each language and (2) the number of publicly available GitHub repositories referencing this language.

To estimate the prevalence of each esolang in search engine results, we queried the phrase ``X'' ```programming language`'' (where `X` is the language name) To assess the extent to which each esolang is actively used in coding projects, we performed a GitHub search using the same query format (``X'' ```programming language`'') to identify repositories mentioning the language. Figure 2

	Minipy	0815	Rhokell	Pyth
gpt-4o-mini	/	×	×	✓
gpt-4o	/	×	×	✓
Deepseek-V3	✓	×	✓	✓
LLAMA-3.3-70B	✓	×	×	✓
Agentic 4o	×	×	×	✓
Agentic Claude	×	×	×	✓

Figure 1: Level of familiarity each tested model has with our Esolang dataset.
/ = Attempted definition of the language, but vague—could likely be guessed just from context of the name of esolang
× = No familiarity with the language
✓ = Clear understanding of the language

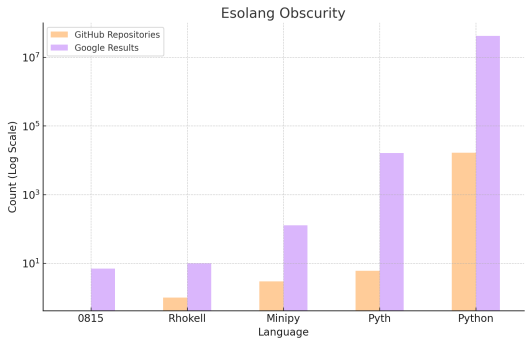

Figure 2: Obscurity of our esoteric programming languages, along with Python as a reference. We see the selected languages being orders of magnitude less common than Python.

shows the stark contrast between mainstream programming languages and esoteric ones in terms of online presence. As a reference point, Python has approximately 42 million adjusted Google search results and 16,500 GitHub repositories, whereas Pyth, our best known language, has only 16,200 Google results and 6 repositories (1000-10000x fewer examples), underscoring how unlikely LLMs are to encounter these esolangs in their training data.

2.3 Evaluation Setup

We form a benchmark that is language agnostic, Esoeval. While numerous benchmarks exist to evaluate the general code generation abilities of language models, we do not use popular benchmarks like SWE-bench Verified or MBPP, which are not language agnostic, instead, focusing on HumanEval and our novel EsoEval benchmark.

HumanEval is a hand-written evaluation set consists of 164 programming problems, each including a function signature, docstring, function body, and several unit tests (an average of 7.7 tests per problem). The HumanEval dataset was quite challenging for LLMs to code in, so we generated an additional simpler baseline for comparison. We present EsoEval—a simplified set of 100 problems. EsoEval includes tasks ranging from basic output statements (e.g., printing "Hello world") to more complex logic problems (e.g., computing factorials, evaluating prime numbers, and performing string manipulations). Despite the complexity variations, these tasks remain relatively simple, OpenAI's gpt-4o-mini achieved a 100.0% accuracy rate.

We experimented with a range of open and closed source models, including GPT-4o-mini, GPT-4o OpenAI [2024],LLAMA-3.3-70B-Instruct-Turbo Grattafiori et al. [2024], Deepseek V3 Liu et al. [2024], and agentic IDEs, e.g. Codeium's Windsurf Codeium [2025].

We standardized the prompt format as follows: Write a function in *[esoteric language]*, an esoteric programming language. The function should perform the following: *[prompt]*. The documentation for [esoteric language] is provided here: [documentation] The standardized prompt and documentation are sent to the model to generate candidate code, which is extracted, saved, and executed using the esoteric language's interpreter. Using HumanEval test cases, the code's outputs are compared to expected results (with minor formatting tolerance), providing a robust framework for evaluating models' ability to generate and run esoteric programs.

For MiniPy, which is fully compatable with Python code the prompt included instructions that Minipy specific code must be used and the displayed results are for correct programs that use at least one Minipy specific function or syntax.

3 Results

We find that providing code examples and documentation allows at least some improvement in all languages, even in some cases where no correct programs were generated without the context. Figures 3 and 4 show accuracies with and without this additional information. Although some gains were present, the near zero success on HumanEval shows that current LLMs have limited ability to utilize documentation to program in esoteric languages.

Our findings show that augmenting model prompts with in-context examples generated by the LLMs themselves can improve subsequent performance on difficult code-generation benchmarks, see Figure 5. Gains plateau after a few examples, suggesting that a few well-chosen examples suffices to saturate the model's context-driven learning capacity. For example, when tackling complex or specialized problems, strategically embedding a few similar examples within the context window can potentially lead to enhanced accuracy without the need for extensive retraining, offering a lightweight, resource-efficient alternative to fine-tuning for adapting models to specialized tasks.

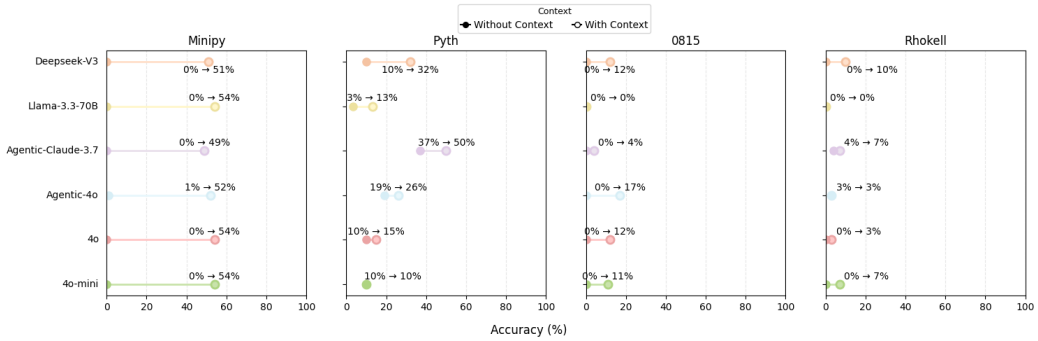

Figure 3: EsoEval Accuracy With & Without Contextual Examples/Documentation

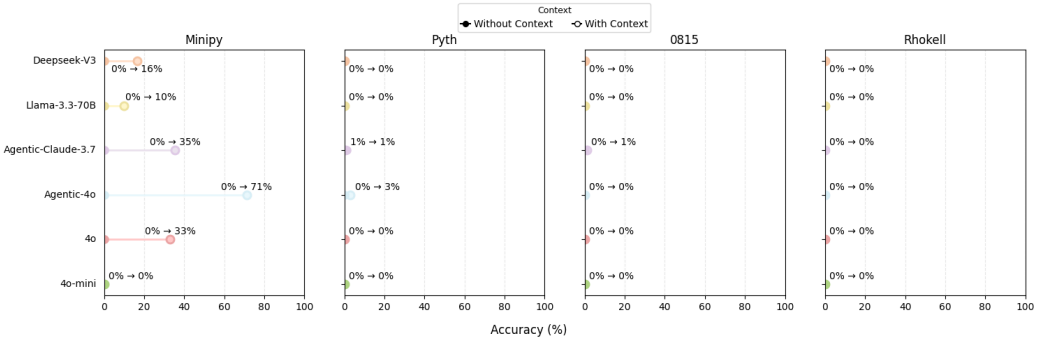

Figure 4: HumanEval Accuracy With & Without Contextual Examples/Documentation

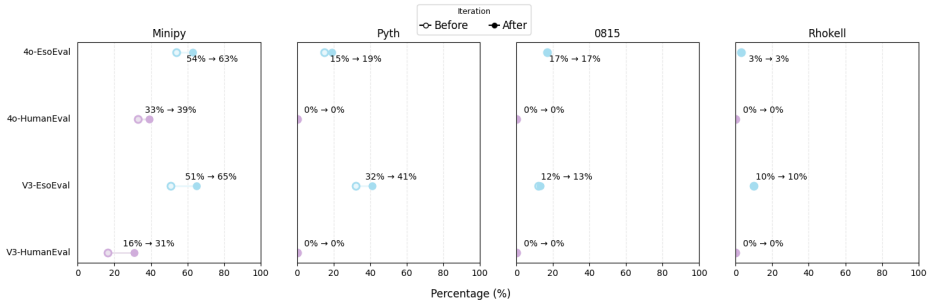

Figure 5: Accuracy after iteratively adding correct model generated examples to the context.

Experiments show that syntactic similarity to Python strongly influences performance: languages like Minipy and Pyth achieve higher accuracy, while more divergent ones such as 0815 elicit near-zero accuracy on complex tasks. Incorporating documentation into prompts further improves validity, highlighting the value of context in guiding code generation.

4 Future Work

We evaluated four esolangs—Minipy, Pyth, 0815, and Rhokell—but hundreds remain. Future work should extend our framework to more languages and broaden model comparisons beyond GPT-4o/mini and a few open-source baselines, testing scaling laws across sizes and architectures. In addition to better understanding scaling behavior both with model size and context; a large sample of esoteric languages could help disentangle what makes a programming language easier or harder for LLMs to use successfully and how large of a factor obscurity plays.

References

Codeium. Windsurf editor. https://codeium.com/windsurf, 2025. Accessed: 2025-05-01.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. URL https://arxiv.org/abs/2211.11559.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023. URL https://arxiv.org/abs/2308.00675.

Jia Li, Ge Li, Chongyang Tao, Huangzhao Zhang, Fang Liu, and Zhi Jin. Large language model-aware in-context learning for code generation. *arXiv preprint arXiv:2310.09748*, 2023. URL https://arxiv.org/abs/2310.09748.

Jia Li, Yunfei Zhao, Yongmin Li, Ge Li, and Zhi Jin. AceCoder: An effective prompting technique specialized in code generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 33(8), 2024. doi: 10.1145/3675395. URL https://arxiv.org/abs/2303.17780.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Federico Mora, Justin Wong, Haley Lepe, Sahil Bhatia, Karim Elmaaroufi, George Varghese, Joseph E. Gonzalez, Elizabeth Polgreen, and Sanjit A. Seshia. Synthetic programming elicitation for text-to-code in very low-resource programming and formal languages. In *Advances in Neural Information Processing Systems (NeurIPS) 2024*, 2024. URL https://openreview.net/forum?id=kQPzFiwVIu.

OpenAI. Gpt-4o system card. https://arxiv.org/abs/2410.21276, 2024. Accessed: 2025-05-01.

Arkil Patel, Siva Reddy, Dzmitry Bahdanau, and Pradeep Dasigi. Evaluating in-context learning of libraries for code generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2024. URL https://arxiv.org/abs/2311.09635.

Hongjin Su, Shuyang Jiang, Yuhang Lai, Haoyuan Wu, Boao Shi, Che Liu, Qian Liu, and Tao Yu. EVOR: Evolving retrieval for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024. URL https://arxiv.org/abs/2402.12317.

Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. DocPrompting: Generating code by retrieving the docs. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2207.05987.

A Related Work

Recent studies have made significant progress in enabling large language models (LLMs) to generate code from in-context prompts, even when using unfamiliar libraries or syntaxes.

One prominent direction is retrieval-augmented code generation, where external documentation or code is provided as part of the prompt. For example, *DocPrompting* by Zhou et al. [2023] retrieves relevant API documentation and adds it to the model's prompt, helping LLMs adapt to unseen libraries without retraining. Similarly,Hsieh et al. [2023] show that supplying tool documentation can enable zero-shot tool use, matching or exceeding few-shot performance without requiring explicit demonstrations.

Another important line of work focuses on optimizing which examples to include in few-shot prompts. Li et al. [2023] propose *Large Language Model-Aware In-Context Learning*, a technique that selects in-context examples based on how much they boost the model's likelihood of solving the task. This leads to substantial gains over traditional retrieval strategies. Complementary to this, Li et al. [2024] introduce *AceCoder*, a staged prompting approach where LLMs are asked to first generate a high-level problem analysis before writing code, further improving code generation accuracy across multiple benchmarks.

In addition to static prompts, dynamic retrieval strategies have been explored. Su et al. [2024] propose *EVOR*, an evolving retrieval framework where the model iteratively refines its retrievals based on generated partial code and execution feedback. EVOR demonstrates significant gains on tasks involving frequently updated libraries and obscure programming languages compared to traditional static retrieval methods.

The question of whether LLMs can learn novel libraries and programming languages purely from in-context information has been explicitly studied by Patel et al. [2024]. Their evaluation shows that LLMs can effectively understand and use previously unseen APIs when provided with either usage examples or plain text descriptions. However, they focus on domain specific tasks testing vision recognition libraries and the language Isabella for automated theorem proving. We are interested in broad programming abilities and examine multiple programming languages with differing properties. Similarly, Gupta and Kembhavi [2023] demonstrate that LLMs can generate compositional programs by observing a new vision-language API without any task-specific fine-tuning.

Finally, Mora et al. [2024] explore a different setting: enabling LLMs to handle very low-resource and formal languages through synthetic intermediate representations. Their method, *SPEAC*, improves LLM performance by constraining generation to a repairable pseudo-language that can later be compiled into the target formalism.

B Additional Language Obscurity Information and Analysis

B.1 Model Familiarity Discussion

Here we provide more detail on the responses summarized in Figure 1. ChatGPT-4o and its mini variant both correctly identified Pyth as a Python-inspired golfing language but showed no genuine familiarity with Rhokell or 0815 and only minimal awareness of Minipy. Deepseek V3 properly classified Pyth and Rhokell while offering only generic or erroneous descriptions for 0815 and Minipy. LLAMA-3.3-70B accurately labeled Pyth and Minipy but failed to provide substantive information on Rhokell or 0815. When presented with five representative code snippets for each language, all models misclassified every example. We observed some common mistakes were interpreting Minipy as buggy Python, labeling Rhokell as Unlambda or vague "functional logic," and giving only superficial labels for Pyth and 0815—thereby demonstrating a marked inability to recognize these esoteric languages from source code alone.

C In Context Examples

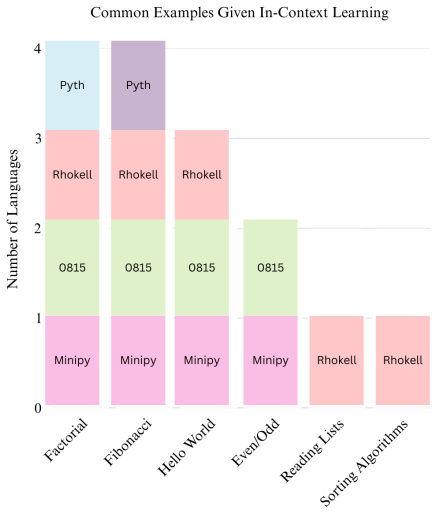

Figure 6: Distribution of common examples provided during in-context learning across the different esoteric programming languages.

0815

A total of 13 sample programs were provided alongside the 0815 documentation to reinforce model understanding. These included basic outputs like "Hello World!" and Cat, computational tasks such as odd/even checks, binary representation, factorial sequences, arithmetic mean, Fibonacci, and summing squares. More complex problems included "99 Bottles of Beer", prime numbers, the Hailstone sequence, a simple randomizer, and truth machines (numeric and ASCII). There is no overlap between the examples provided and HumanEvaltest set but there is minor overlap between the examples given and those in EsoEval. There is overlap between the in-context examples and EsoEval for the following 4 examples: printing "Hello World!", even/odd number function, factorial, and Fibonacci.

Pyth

For Pyth, a total of 7 examples were provided, with a strong focus on factorial computation. This included three factorial-related sub-examples: Factorial 3.1.1, Factorial 3.1.3 (The Iterative Factorial), and the Recursive Factorial. Additionally, there were examples showcasing memoization (subsets function), functional programming with reduce, Fibonacci sequence generation, and solving the Collatz sequence. There is no overlap between the examples provided and HumanEvaltest set but there is minor overlap between the examples given and those in EsoEval. There is overlap between the in-context examples and EsoEval for the following 2 examples: factorial and fibonacci.

Rhokell

For Rhokell, a total of 11 examples were provided, covering a range of algorithmic and computational topics. Several examples focus on mathematical sequences, such as computing factorials, Fibonacci numbers, primes, and the Kolakoski sequence. Sorting and list manipulation are also demonstrated, with a quicksort implementation and a general lists example. Additional examples explore syntax and functional programming concepts, including Peano arithmetic, binary arithmetic, and a quine program. There is no overlap between the examples provided and HumanEvaltest set but there is minor overlap between the examples given and those in EsoEval. There is overlap between the in-context examples and EsoEval for the following 3 examples: printing "Hello World!", factorial, and Fibonacci.

Minipy

For Minipy, no examples were provided for EsoEval. However, among the code generated by Deepseek V3 for EsoEval—which compiled correctly only in the Minipy interpreter and not in the standard Python interpreter—the resulting examples were collected and subsequently used for testing on HumanEval. There is no overlap between the examples provided and those in HumanEval.

D Language Specific Observations

Minipy occupies a unique position among our esoteric languages: although it extends Python with concise shorthand constructs, most HumanEval tasks do not require those extensions, so a model can "cheat" by emitting plain Python and still pass the tests. To prevent this shortcut, we enforced a non-Python compilability requirement in our EsoEval metric: any submission that successfully ran under a standard Python interpreter were excluded, regardless of functional accuracy.

When evaluated on HumanEval, GPT-4o-mini invariably fell back on plain Python, yielding 0% of solutions that failed to compile under a standard Python interpreter. Llama-3.3-70B-Instruct-Turbo exhibited the same tendency, with only 10 % (HumanEval) and 7.7% (HumanEval subset) of its outputs producing non-compilable code. By contrast, on the simpler EsoEval benchmark—where true Minipy syntax is required—both models showed dramatic gains in non-Python compilability accuracy: GPT-4o-mini reached 54% non-compilable submissions, Llama-3.3-70B-Instruct-Turbo 54%, and DeepSeek V3 51%.

These results suggest that, when confronted with complex tasks, models prefer the safety of familiar Python constructs rather than leverage Minipy's shorthand features. However, on more straightforward problems, they are capable of nontrivially applying the documented Minipy extensions. By measuring non-compilability in Python, we ensure that high EsoEval accuracy truly reflects understanding of Minipy's specialized syntax rather than a fallback to Python.

Across the other three esoteric languages—0815, Pyth, and Rhokell—and two evaluation frameworks (HumanEval, 10 tasks; EsoEval, 30 tasks), we observed that the degree of syntactic divergence is correlated with LLM performance. For example, the hexadecimal-only, comment-filtering 0815 language, GPT-4o-mini scored 0% on HumanEval but 11% on the simpler EsoEval benchmark, whereas LLAMA-3.3-70B achieved 0% and Deepseek V3 12% on EsoEval. In Pyth—a Python-inspired golfing language—GPT-4o-mini again scored 0% on HumanEval but attained 10% on EsoEval, with LLAMA-3.3-70B and Deepseek V3 reaching 13% and 32%, respectively. Finally, for Rhokell, which fuses ρ calculus with Haskell-style syntax, GPT-4o-mini produced 0% accuracy on HumanEval but 3% on EsoEval, while LLAMA-3.3-70B remained at 0% and Deepseek V3 achieved 10%. These results suggest that moderate syntactic departures—such as Pyth's concise, Python-derived abbreviations—permit some transfer of existing knowledge, but more unusual syntax like those of 0815 and Rhokell inhibit code generation.

E Agentic AI Framework Evaluation

For the second part of our evaluation, we turned to agentic AI frameworks. Specifically, we evaluated tools such as Windsurf and Cursor by prompting their respective agents to write the code for both the HumanEval and EsoEval datasets.

E.1 Language Parsing

One additional challenge arose from the need to adapt our testing harness to the specifications of each language and the structure of the HumanEval benchmark. Because HumanEval's reference implementations use Python-style assert statements, any candidate solution needed a a Python-callable function. In practice, many esoteric-language programs required input-output wrappers to conform to the HumanEval harness, and some languages lacked any built-in notion of user-defined functions. We translated between string, list, or integer representations —to ensure that each candidate program could be tested uniformly by the test runner. At the same time, we strove to respect each language's native syntax and execution model, providing only the smallest necessary adaptation rather than rewriting the core logic. As a result, every testing harness is slightly different; in the remainder of this section, we describe those per-language adjustments in detail

E.1.1 MiniPy

MiniPy was the most straightforward language to work with due to its similarity with Python as a coding language. For this language, the outputed code was directly executable using a Python compiler with a list of shorthands appended to the beginning of each program.

E.1.2 Pyth

Architecture Overview We developed a systematic approach to testing Pyth code using Python's testing infrastructure, focusing on three key components. The first component was our Code Translation Layer, which implemented `get_pyth_translation` to capture Python translations from the Pyth interpreter's stderr output. This was important since our testing dataset contained our tests using Python assert statements. Therefore, parsing the translation in the stderr output was the simplest solution.

The second component, our Test Execution Environment, centered around the `test_pyth_function` which dynamically executed Pyth code with arbitrary inputs. Since the output of the Pyth program only existed withint the context of the interpreter, we set up an environment to manage variables through a global `environment` dictionary and handled return value propagation via `environment['K']`, ensuring consistent state management between Pyth and Python contexts.

The execution flow is shown in the following workflow:

```python
def workflow(pyth_code, input_value):
    translation = get_pyth_translation(pyth_code)
    python_func = create_python_function(translation)
    result = test_pyth_function(python_func, input_value)
    return result
```

E.1.3 0815

Architecture Overview For the 0815 esoteric language implementation, modified the testing framework to address the differences with working with a register-based hexadecimal language. The first component was our Register Management System, which handled the language's three 64-bit registers: X (write-only), Y (helper), and Z (read-only). This involved state tracking and hexadecimal conversions for all numeric operations.

For Test Case Integration, we implemented a system that bridged between decimal test inputs and 0815's hexadecimal requirements. This included automatic conversion of test inputs to hexadecimal format and proper interpretation of hexadecimal outputs back to decimal for test validation. This was especially important when figuring out representations for lists and other unique data structures. We also refactored the assert statements within the HumanEval test cases to generate text files with the test cases written out instead. They were then parsed and converted using the process described above to test each program.

E.1.4 Rhokell

Architecture Overview For the Rhokell language implementation, we developed a testing framework that integrated with Rust's cargo build system while providing a Python-based test harness. The first component was our Rust Integration Layer, which managed the compilation and execution of Rhokell code through cargo. This required careful handling of build processes and proper path management to ensure reliable interpreter access.

The second component was our Execution Environment, which utilized a robust subprocess management system to handle both compilation and runtime phases. This dual-phase approach was necessary due to Rhokell's compiled nature, distinguishing it from interpreted languages like Pyth and 0815. The environment tracked compilation success separately from execution results, providing detailed feedback for both phases.

For Test Case Management, we implemented a dataclass-based statistics tracking system that monitored multiple aspects of test execution. This included tracking total problems attempted, successful compilations, passed tests, and aggregate test counts, providing comprehensive metrics for evaluation.

Key Technical Challenges and Solutions The implementation presented several unique technical challenges. The primary challenge was managing the Rust-based interpreter's build process. Unlike the other esolangs we tested, Rhokell was not implemented with Python-based interpreters. We resolved this by implementing a pre-execution build check system that verified the interpreter's availability and triggered compilation when necessary.

This also involved delving into the process management, especially with testing. The solution involved implementing a timeout-aware execution system that properly handled both compilation and runtime errors while maintaining clean state. The test cases were treated similarly to previous esolangs, being written into a text file and then parsed into a form recognizable by the language.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims of testing in-context learning and in-context learning with verification of esoteric programming languages is mentioned in the abstract and intro and supported in the results section.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: Limitations of our study are discussed throughout the text and especially in the Future Work section.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: We do not have any theoretical claims.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We include details on models and prompts used as well as how we analyze the data. We believe this is sufficient to perform an equivalent experiment and we further intend to release code used in this project. However, the instability and incomplete documentation of non-open source LLMs may hamper the reproducibility for those cases.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We intend to make the code for our experiments and analysis publicly available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: No training was performed. Experiments are described in the paper and we plan to provide code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All regressions and correlations are reported with statistical significance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: Compute resources used were not uniform (different computers used to run them and different model providers) and not carefully tracked. Experiments were run primarily on personal laptops or with API's for closed source providers. We believe a few hundred dollars were allocated across subscription and compute purchases.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: I have read the Code of Ethics and believe our work complies completely.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: This is primarily discussed in the Introduction and Future Works sections.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: We do not foresee any significant risk of misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: No other assets were used in the paper. We will annotate all code and documentation as appropriate. The language documentation and code examples used in our testing were checked and we believe are all being used in accordance to their licenses.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We plan to release the paper under CC-BY 4.0 if accepted in accordance with this conference's standards and plan to release our code under a similar license.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: There were no human experiments or crowdsourcing of data.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: There were no human experiments or crowdsourcing of data.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Evaluation of LLM capabilities is the core question of this research. This type of evaluation is a standard line of research inquiry. We have also used LLMs for assistance in code generation, editing, and background research but none of this is original or of core importance to the research.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.