ESOLANG-BENCH: EVALUATING GENUINE REASONING IN LARGE LANGUAGE MODELS VIA ESOTERIC PROGRAMMING LANGUAGES

Aman Sharma **Paras Chopra**
Lossfunk
{aman.sharma, paras}@lossfunk.com

ABSTRACT

Large language models achieve near-ceiling performance on code generation benchmarks, yet these results increasingly reflect memorization rather than genuine reasoning. We introduce **EsoLang-Bench**, a benchmark using five esoteric programming languages—Brainfuck, Befunge-98, Whitespace, Unlambda, and Shakespeare—that have 1,000–100,000× fewer public repositories than Python (based on GitHub search counts), making them resistant to data contamination and benchmark gaming. These languages require the same computational primitives as mainstream programming but in radically different syntactic forms. We evaluate five frontier models across five prompting strategies and find a dramatic capability gap: models achieving 85–95% on standard benchmarks score only **0–11%** on equivalent esoteric tasks, with **0% accuracy beyond the Easy tier**. Few-shot learning provides negligible benefit ($p = 0.505$), suggesting these techniques exploit training priors rather than enabling genuine in-context learning. EsoLang-Bench provides a contamination-resistant probe for transferable reasoning that is resistant to gaming incentives.

1 INTRODUCTION

Large language models have achieved impressive performance on code generation benchmarks, with state-of-the-art systems reaching 85–95% on HumanEval and MBPP (Chen et al., 2021; Austin et al., 2021). However, these benchmarks increasingly suffer from data contamination (Zhang et al., 2024; Sainz et al., 2023) and design flaws that allow high accuracy through pattern matching rather than genuine reasoning (Gupta et al., 2024). Evaluating models on truly out-of-distribution (OOD) tasks is essential for measuring genuine reasoning. Esoteric programming languages offer a principled solution: minimal training corpus representation makes them economically irrational to include in pretraining, while they still require the same fundamental computational reasoning (loops, conditionals, state management) as mainstream languages. This reflects Goodhart's Law (Goodhart, 1984): when a measure becomes a target, it ceases to be a good measure.

Our contributions: (1) **EsoLang-Bench**, 80 programming problems across four difficulty tiers in five esoteric languages; (2) comprehensive evaluation of five frontier LLMs across five prompting strategies, showing ICL provides no significant benefit for OOD tasks; (3) evaluation of agentic systems (Codex, Claude Code) showing interpreter-in-the-loop feedback provides the largest gains; (4) error analysis revealing a syntax/semantics split between partially- and completely-unseen languages.

2 ESOLANG-BENCH DATASET

2.1 DATASET DESIGN

EsoLang-Bench consists of 80 programming problems organized into four difficulty tiers with 20 problems each. Each problem includes a natural language description and 6 input-output test cases. Problems test fundamental algorithmic reasoning (loops, conditionals, state management) rather than domain-specific knowledge, and are language-agnostic: the same challenge must be solved in

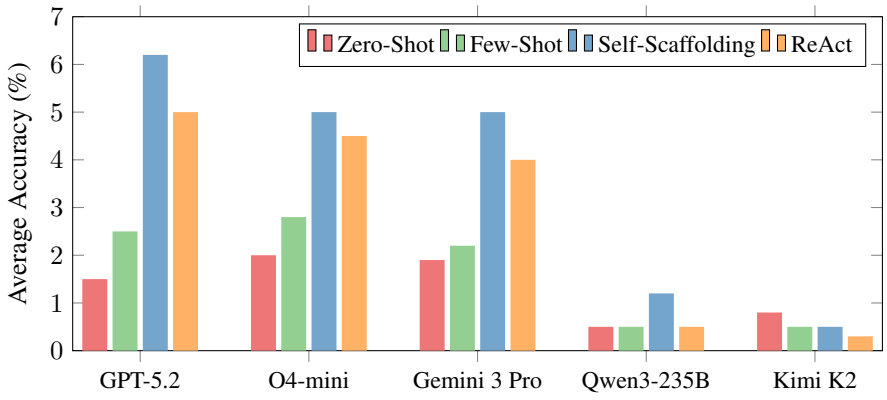

Figure 1: Average accuracy (%) across all five esoteric languages (Brainfuck, Befunge-98, Whitespace, Unlambda, Shakespeare) by model and prompting strategy. Self-Scaffolding consistently achieves the highest accuracy; GPT-5.2 reaches 6.2%. All models score below 7% even with advanced prompting. Agentic system results (Codex, Claude Code) are reported separately in Table 3 and are not included here.

any of the five target languages. Difficulty tiers are defined by algorithmic complexity of a Python reference solution, independent of observed model performance: **Easy** requires single-loop or basic I/O (e.g., sum two integers); **Medium** requires multi-loop control flow or basic recursion (e.g., Fibonacci, factorial); **Hard** requires nested data structures or non-trivial string algorithms (e.g., balanced parentheses, prime counting); **Extra-Hard** requires classical algorithms with complex state management (e.g., longest increasing subsequence, Josephus problem). Calibration was performed by expert assessment prior to any model evaluation. The complete list of all 80 problems is provided in Appendix B.

2.2 Target Languages

We select five esoteric languages representing diverse paradigms (see Appendix A for full descriptions):

Brainfuck: Memory-tape language with only 8 commands; requires pointer arithmetic without variables or functions ($\approx$5,000 GitHub repos).

Befunge-98: 2D stack-based language where the instruction pointer travels in four cardinal directions; requires spatial reasoning ($\approx$2,000 repos).

Whitespace: Only space, tab, and newline have semantic meaning; stack-based with whitespace-encoded commands ($\approx$200 repos).

Unlambda: Pure combinatory logic with no variables; computation via combinators `s`, `k`, `i` ($\approx$100 repos).

Shakespeare: Theatrical-play syntax where dialogue performs computation; alien semantics despite natural-language appearance ($\approx$150 repos).

All five are Turing-complete. Public repositories are 1,000–100,000$\times$ scarcer than Python ($>$10M repos), making contamination economically irrational while still allowing human experts to learn the languages from documentation. These languages were created as recreational programming challenges and lack formal academic publications; their reference specifications are maintained on the Esolang Wiki (`https://esolangs.org`), which we use as the documentation source in all prompts.

	Brainfuck		Befunge-98		Whitespace		Unlambda		Shakespeare	
Model	0-S	3-S	0-S	3-S	0-S	3-S	0-S	3-S	0-S	3-S
GPT-5.2	2.5	2.5	2.5	**8.8**	0	0	0	0	**2.5**	1.2
O4-mini	2.5	**3.8**	**6.2**	7.5	0	0	0	0	1.2	1.2
Gemini 3 Pro	2.5	3.8	5.0	3.8	0	0	0	0	1.2	1.2
Qwen3-235B	2.5	1.2	0	0	0	0	0	**1.2**	0	0
Kimi K2	0	0	2.5	1.2	0	0	0	0	1.2	1.2

Table 1: Zero-shot and few-shot accuracy (%) across all models and languages. Each language has 80 problems (20 per difficulty tier). Only Easy-tier problems were solved; Medium/Hard/Extra-Hard = 0% across all configurations. Best per language in **bold**. 0-S = Zero-Shot, 3-S = 3-Shot Few-Shot.

3 Experimental Setup

3.1 Models and Prompting Strategies

We evaluate five frontier LLMs: **GPT-5.2**, **O4-mini-high**, **Gemini 3 Pro**, **Qwen3-235B**, and **Kimi K2 Thinking**. We assess five prompting strategies:

Zero-Shot: Language documentation + problem + 6 test cases. No examples.

Few-Shot: Zero-shot extended with 3 solved example programs in the target language.

Self-Scaffolding (S-S): Iterative: model generates code, receives interpreter feedback (actual vs. expected output, error messages), and refines for up to 5 iterations (1 LLM call/iteration).

Textual Self-Scaffolding (TSS): Two-agent iteration: a *coder* generates code and a separate *critic* analyzes failures and provides natural-language guidance (2 LLM calls/iteration).

ReAct: Three-stage pipeline: *planner* generates pseudocode, *code editor* translates to target language, *critic* analyzes failures (3 LLM calls/iteration).

3.2 Agentic Systems

We additionally evaluate **GPT-5.2 Codex** and **Claude Code (Opus 4.5)** on Brainfuck and Befunge-98 only; Whitespace, Unlambda, and Shakespeare were excluded from agentic evaluation as all non-agentic models scored 0% on those languages in preliminary runs, making additional compute expenditure uninformative.

3.3 Evaluation Protocol

We evaluate all models on the complete 80-problem dataset across all five languages (400 combinations per strategy). Three independent runs (seeds 0, 1, 2), temperature $\tau = 0.7$. We report mean accuracy with 95% confidence intervals (bootstrap, $n = 1000$). A problem is considered solved if and only if all six test cases produce **exact-match interpreter output** (i.e., the interpreter's stdout matches the expected output exactly, character-for-character). Statistical comparisons use paired Wilcoxon signed-rank tests with Bonferroni correction ($\alpha = 0.05$).

4 Results

Tables 1 and 2 present our main results. **All models achieve 0% on Medium, Hard, and Extra-Hard problems across every configuration**—success is entirely limited to the Easy tier.

4.1 Static Prompting

Performance correlates with training data availability: Befunge-98 achieves the highest accuracy, followed by Brainfuck and Shakespeare; all models achieve 0% on Whitespace and near-zero on Unlambda. Within the Easy tier, GPT-5.2 self-scaffolding solves 9/20 Befunge-98 problems (45%) and 5/20 Brainfuck problems (25%); Codex solves 11/20 Brainfuck Easy problems (55%). This

	Brainfuck			Befunge-98			Whitespace			Unlambda			Shakespeare		
Model	S-S	TSS	Re	S-S	TSS	Re	S-S	TSS	Re	S-S	TSS	Re	S-S	TSS	Re
GPT-5.2	**6.2**	**3.8**	**5.0**	**11.2**	**10.0**	**8.8**	0	0	0	**1.2**	0	0	**2.5**	**2.5**	**1.2**
O4-mini	5.0	2.5	3.8	10.0	6.2	7.5	0	0	0	0	0	0	1.2	1.2	0
Gemini 3 Pro	5.0	**3.8**	3.8	7.5	6.2	7.5	0	0	0	0	0	0	1.2	0	0
Qwen3-235B	2.5	1.2	2.5	0	0	0	0	0	0	**1.2**	0	0	1.2	0	0
Kimi K2	0	0	0	0	0	0	0	0	0	0	0	0	1.2	0	0

Table 2: Scaffolding strategy accuracy (%) across all models and languages. S-S = Self-Scaffolding (direct interpreter feedback, 1 LLM call/iter); TSS = Textual Self-Scaffolding (coder-critic pair, 2 LLM calls/iter); Re = ReAct pipeline (3 LLM calls/iter). Best per column in **bold**.

System	Brainfuck	Befunge-98	Avg.
Codex (Agentic)	**13.8%**	**8.8%**	**11.2%**
Claude Code (Opus 4.5)	12.5%	8.8%	10.6%

Table 3: Agentic system accuracy on Brainfuck and Befunge-98 only (see §3.2 for exclusion rationale). Both systems have direct interpreter access with iterative feedback.

reveals a sharp performance cliff: models show substantial partial success at Easy but drop to exactly 0% at the boundary between single-loop tasks and multi-step algorithmic reasoning.

In-context learning provides negligible benefit. Few-shot prompting shows no statistically significant improvement over zero-shot (Wilcoxon $p = 0.505$, average $+0.8$ pp). This aligns with Min et al. (2022): ICL activates pre-trained knowledge rather than teaching genuinely new skills. In this ultra-low-resource setting, demonstration examples cannot compensate for absent foundational knowledge.

4.2 Scaffolding Strategies

Self-scaffolding achieves the best overall result: GPT-5.2 reaches 11.2% on Befunge-98 ($p < 0.01$ vs. zero-shot). No significant difference between S-S and TSS ($p = 0.803$), yet S-S uses half the compute (1 vs. 2 LLM calls/iter). When all components lack domain knowledge, multi-agent intermediaries introduce noise rather than signal—the critic cannot provide useful feedback about syntax it has never encountered.

4.3 Agentic Systems

Agentic systems achieve 2–3$\times$ improvement over non-agentic approaches, with Codex reaching 13.8% on Brainfuck—the highest single-language result. Gains come from efficient context management: structured logging with selective retrieval of relevant prior attempts (mitigating the "lost-in-the-middle" effect (Liu et al., 2024)) and dynamic question-specific example retrieval rather than static demonstrations. Qualitative analysis reveals a consistent sub-pattern: decimal number I/O problems (E04, E05, E10–E13) account for most Easy-tier failures across all agentic and non-agentic systems alike, as parsing ASCII digit sequences and performing multi-cell arithmetic have extremely scarce training data even within Brainfuck's limited corpus.

5 Discussion

Error profiles reveal syntax vs. semantics gap. Figure 2 shows error distributions. Languages with more online presence (Brainfuck, Befunge-98; 500–2,000 GitHub repos) show low compile error rates (15–20%) but high logic error rates (35–60%): models acquire surface syntax but fail on algorithmic reasoning. Ultra-low-resource languages (Whitespace, Unlambda, Shakespeare; $<$200 repos) exhibit near-total compilation failure (70–100%). This binary pattern—syntax acquisition with semantic failure vs. complete syntactic failure—is consistent with a boundary between partial and

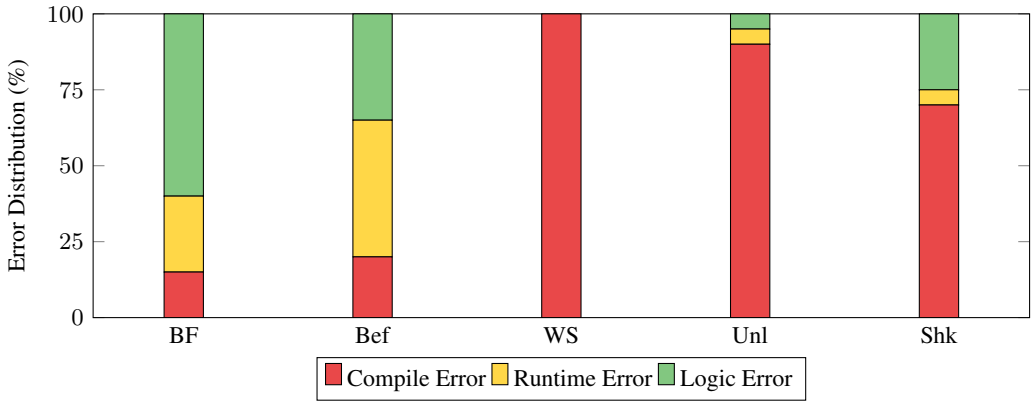

Figure 2: Error distribution by language (GPT-5.2 zero-shot). BF=Brainfuck, Bef=Befunge-98, WS=Whitespace, Unl=Unlambda, Shk=Shakespeare. High-resource languages (BF, Bef) fail on logic; ultra-low-resource languages fail primarily at compile time.

absent pre-training coverage, though additional confounds (e.g., tokenizer behaviour for Whitespace, where BPE tokenizers may merge or strip the sole meaningful characters) may contribute.

The Easy–Medium cliff reflects compositional failure. This boundary aligns with Dziri et al. (2023): transformers fail when tasks require novel *combinations* of known primitives—precisely what Medium problems demand. Capability also diverges substantially within the frontier tier: Qwen3-235B and Kimi K2 score near-zero even on Easy Befunge-98, while GPT-5.2 reaches 11.2%.

Key Takeaways for Practitioners. (1) Skip few-shot curation for OOD tasks. ICL cannot activate priors that do not exist. Zero-shot with documentation matches few-shot while saving prompt engineering effort and context budget. **(2) Prefer direct feedback over multi-agent reasoning.** When domain knowledge is absent, each additional LLM layer can amplify errors. Direct ground-truth interpreter signals outperformed multi-agent critique because the interpreter is objective, whereas the critic also lacked domain knowledge. **(3) Invest in context management.** Selective retrieval of prior attempts and dynamic question-specific examples outperforms both static few-shot demonstrations and full conversation history approaches.

Limitations and future directions. The complete failure beyond Easy limits fine-grained differentiation among models at harder tiers. For Whitespace specifically, near-total compilation failure may partly reflect tokenizer constraints: BPE tokenizers commonly normalize or strip whitespace characters during encoding and decoding, yet Whitespace programs consist *entirely* of space, tab, and newline as their sole meaningful tokens—a mechanical incompatibility that is distinct from reasoning limitations. Disentangling these effects would require byte-level models or tokenizer-aware prompting, which we leave to future work. Two further extensions: (i) a *Python-first-translate* approach—write a Python solution then translate to the esoteric language—would reveal whether the bottleneck is algorithmic reasoning or syntactic knowledge; (ii) an extra-easy sub-tier would enable finer-grained probing of the Easy-to-Medium capability cliff.

6 Conclusion

EsoLang-Bench reveals that frontier models attain only 0–11% accuracy on esoteric-language equivalents of problems they solve at 85–95% in Python, with complete failure beyond the Easy tier regardless of prompting strategy. The negligible benefit of few-shot learning ($p = 0.505$) indicates that ICL primarily activates existing training priors rather than enabling authentic adaptation to novel domains. By targeting languages with 1,000–100,000$\times$ fewer public repositories than Python, EsoLang-Bench provides a principled, contamination-resistant probe for transferable computational reasoning. Our results suggest that improving OOD performance may require architectural innovations—such as more flexible in-context adaptation or test-time learning mechanisms—rather than further scaling of training data alone.

Reproducibility Statement

Upon publication we will open-source: the complete dataset (80 problems, 6 test cases each); Python interpreters for all five languages; evaluation pipelines for all prompting strategies; and raw experimental outputs. The dataset is available at `https://huggingface.co/datasets/lossfunk/esolang-bench` under CC BY 4.0.

Ethics Statement

This work evaluates publicly available models on programming tasks. Results should not be extrapolated to claims about general AI capabilities; our objective is calibrated assessment of capability boundaries. Interpreters are sandboxed. AI assistants were used to aid literature survey and writing refinement; all content was reviewed and verified by the authors.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023.

Charles A. E. Goodhart. Problems of monetary management: The UK experience. In *Monetary Theory and Practice: The UK Experience*, pp. 91–121. Macmillan, London, 1984.

Vipul Gupta, David Pantoja, Candace Ross, Adina Williams, and Megan Ung. Changing answer order can decrease MMLU accuracy. *arXiv preprint arXiv:2406.19470*, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11048–11064, 2022.

Oscar Sainz, Jon Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10776–10787, 2023.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Charlotte Zhuang, Dylan Slack, Qin Lyu, Sean Hendryx, Russell Kaplan, Michele Lunati, and Summer Yue. A careful examination of large language model performance on grade school arithmetic. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024.

A Esoteric Language Details

Brainfuck (1993): Created by Urban Müller. Only 8 commands (`>`, `<`, `+`, `-`, `[`, `]`, `.`, `,`) operating on a 30,000-cell memory tape. Solving problems requires pointer arithmetic, loop invariants, and memory layout reasoning without named variables or functions.

Befunge-98 (1993): Created by Chris Pressey. Two-dimensional stack-based language where the instruction pointer can travel in four cardinal directions (`>`, `v`, `<`, `^`); conditional commands `_` and `|` branch on stack values; `p`/`g` enable self-modifying code.

Whitespace (2003): Created by Edwin Brady and Chris Morris. Only space, tab, and newline characters have semantic meaning. Stack-based with commands encoded as whitespace sequences; other characters are comments.

Unlambda (1999): Created by David Madore. Minimal functional language based on combinatory logic with no variables; only function application via backtick. Core combinators: `s` (substitution), `k` (constant), `i` (identity). Simple arithmetic requires Church numerals through combinator compositions.

Shakespeare (2001): Created by Karl Wiberg and Jon Åslund. Programs are theatrical plays: variable declarations are character introductions, scenes/acts control flow, dialogue performs computation. Variable values are determined by adjectives (positive words contribute $+1$, negative words -1, iteratively).

B Dataset Specification

B.1 Problem Category Distribution

Table 4: Distribution of problems across programming categories.

Category	Count	Representative Problems
Basic I/O	5	Hello World, Echo Line, Concatenate Lines
Arithmetic	17	Sum, Multiply, Factorial, Mod Exponentiation
String Manipulation	26	Reverse, Palindrome, Caesar Cipher, RLE
Number Theory	8	GCD, Primes, Factorization, LCM
Base Conversion	4	Binary↔Decimal, Roman↔Integer
Sorting/Arrays	9	Sort, LIS, Inversions, Merge Sorted
Stack/Parsing	5	Balanced Parens, Postfix Eval
State Machines	4	Tape Walk, Josephus Problem
Bitwise Operations	2	Hamming Distance, Count Set Bits

B.2 Full Problem List

Easy (E01–E20): Hello World, Echo Line, Hello Name, Sum Two Integers, Multiply Two Integers, Even Or Odd, String Length, Reverse String, Count Vowels, Sum From 1 To N, Sum Of Digits, Minimum Of Two, Maximum Of Three, Repeat String N Times, Concatenate Two Lines, First And Last Character, Uppercase String, Count Spaces, Integer Average Of Two, Compare Two Integers.

Medium (M01–M20): Palindrome Check, Word Count, Run Length Encoding, Caesar Shift By 3, Simple Binary Expression, GCD, Factorial, Nth Fibonacci Number, Decimal To Binary, Binary To Decimal, Substring Occurrences, Remove Vowels, Sort Numbers, Second Largest Distinct Number, Anagram Test, Interleave Two Strings, Replace Spaces With Underscores, Sum Of List, Characters At Even Indices, Count Distinct Characters.

Hard (H01–H20): Balanced Parentheses, Evaluate Expression With Precedence, Count Primes Up To N, Nth Prime Number, Big Integer Addition, Longest Word, Longest Common Prefix, Digit Frequency, General Caesar Cipher, Remove Consecutive Duplicates, Run Length Decoding, ASCII

Sum, Polynomial Evaluation, List All Divisors, Tape Walk Final Position, Longest Run Length, Most Frequent Value, Divisible By 3, Plus Minus Reset Machine, Sort Strings Lexicographically.

Extra-Hard (X01–X20): Prime Factorization, Longest Increasing Subsequence Length, Matrix Multiplication Result Element, Evaluate Postfix Expression, Merge Two Sorted Arrays, Compute Power Modulo, Longest Palindromic Substring, Count Set Bits In Range, Bracket Depth Maximum, String Rotation Check, Count Inversions, Least Common Multiple, Valid Parentheses Types, Next Greater Element, Spiral Matrix Traversal, Hamming Distance, Roman To Integer, Integer To Roman, Permutation Check, Josephus Problem.

B.3 Sample Problems

B.3.1 Easy: E04 — Sum Two Integers

```
Read two integers a and b separated by whitespace.
Output their sum a + b as a plain integer.

1. Input: "5 7"      -> Output: "12"
2. Input: "-3 10"    -> Output: "7"
3. Input: "0 0"      -> Output: "0"
4. Input: "100 200"  -> Output: "300"
5. Input: "-50 -25"  -> Output: "-75"
6. Input: "999 1"    -> Output: "1000"
```

B.3.2 Extra-Hard: X20 — Josephus Problem

```
Read N and K. N people in circle 1..N. Starting from
person 1, count K clockwise and eliminate. Repeat
until one remains. Output survivor position.

1. Input: "5 2"  -> Output: "3"
2. Input: "7 3"  -> Output: "4"
3. Input: "1 1"  -> Output: "1"
4. Input: "6 1"  -> Output: "6"
5. Input: "10 2" -> Output: "5"
6. Input: "4 2"  -> Output: "1"
```

C Prompting Templates

C.1 Zero-Shot Prompt

System:

```
You are an expert {language_name} programmer. Given
a problem and sample tests, output ONLY valid code.
No explanations, no comments, no markdown. Programs
must read stdin exactly as specified and write
deterministic stdout matching expected output.

Reference documentation:
{documentation_text}
```

User:

```
Problem ID: {problem_id}
Title: {problem_title}
Description: {problem_description}

Tests (stdin => stdout):
```

```
1. Input: {test_1_input}
   Output: {test_1_output}
[... 6 test cases total ...]

Return only the program.
```

C.2 Self-Scaffolding Feedback Prompt

User (after each failed attempt):

```
[Problem specification as above]

Previous attempt and interpreter feedback:
=== Program ===
{previous_code}

=== Interpreter Feedback ===
Test 1:
Input: {test_input}
Expected: {expected_output}
Actual: {actual_output}
Error Type: {error_type}
Stderr: {stderr}
[... up to 6 test results ...]

Return only the updated program.
```

D Extended Results

D.1 Language-Specific Breakdown

Table 5: Brainfuck: Easy problems solved / 20 by model and strategy.

Model	0-S	3-S	S-S	TSS	Re
GPT-5.2	2	2	**5**	3	4
O4-mini	2	3	**4**	2	3
Gemini 3 Pro	2	3	**4**	3	3
Qwen3-235B	2	1	**2**	1	2
Kimi K2	0	0	0	0	0

Table 6: Befunge-98: Easy problems solved / 20 by model and strategy.

Model	0-S	3-S	S-S	TSS	Re
GPT-5.2	2	7	**9**	8	7
O4-mini	5	6	**8**	5	6
Gemini 3 Pro	4	3	**6**	5	6
Qwen3-235B	0	0	0	0	0
Kimi K2	2	1	0	0	0

D.2 Agentic Detailed Results

Codex – Brainfuck (11/20): E01, E02, E03, E06, E07, E08, E09, E15, E16, E17, E18.

Claude Code – Brainfuck (10/20): E01, E02, E03, E08, E09, E14, E15, E16, E17, E18.

Table 7: Whitespace, Unlambda, Shakespeare: Best Easy solved / 20 (best across all strategies).

Model	Whitespace	Unlambda	Shakespeare
GPT-5.2	0	1	2
O4-mini	0	0	1
Gemini 3 Pro	0	0	1
Qwen3-235B	0	1	1
Kimi K2	0	0	1
Best	0	1	2

Table 8: Agentic system performance: Easy problems solved / 20 per language.

System	Brainfuck	Befunge-98	Total (Acc)
Codex (OpenAI)	**11**	7	**18 (11.2%)**
Claude Code (Opus 4.5)	10	7	17 (10.6%)

Key failure pattern: Decimal I/O problems (E04, E05, E10–E13, E19, E20) fail consistently: parsing ASCII digits and multi-cell arithmetic have extremely scarce training data even in Brainfuck.