

From Language Specifications to Executable Turing Machines: Evaluating LLMs as Computational Machine Designers

Shahriyar Zaman Ridoy^{1†}, S. M. Muhtasimul Hasan¹, Azmine Toughshik Wasi²,
Kinga Tshering³, Koushik Ahamed Tonmoy³, Md Mosaddek Khan¹

¹Cognitive Agents and Interaction Lab (CAIL), University of Dhaka

²Computational Intelligence and Operations Laboratory

³Department of Electrical and Computer Engineering, North South University

shahriyar.zaman01@gmail.com, muhtasimhasan904@gmail.com, azminetoughshik.wasi@gmail.com
kinga.tshering@northsouth.edu, koushik.tonmoy@northsouth.edu, mosaddek@du.ac.bd

Abstract

Recognizing a formal language does not imply the ability to construct the computation that decides it. We operationalize this distinction as an executable benchmark for language-model reasoning: given a formal-language specification, an LLM must synthesize a deterministic or nondeterministic Turing machine that decides the language. We introduce TMEVAL, a benchmark of 253 language specifications across 7 structural categories spanning context-free and non-context-free phenomena. Each language includes 600 oracle-labeled membership tests, and every output is evaluated as an executable artifact rather than a textual response. Our pipeline enforces a strict output contract, validates machine construction, executes each buildable machine, and assigns one of four outcomes: STATIC_FAIL, BUILD_FAIL, SEMANTIC_MISMATCH, or EXACT_PASS. Across eight LLMs, 2,024 model-language candidates, and up to 1,214,400 behavioral evaluations, only 11.12% exactly pass all tests. More than half fail before execution, over one-third implement the wrong language, and 75 languages remain unsolved by all systems. Failure depends less on the context-free/non-context-free boundary than on the operational structure required to realize the specification. TMEVAL reveals a substantial gap between formal-language recognition and machine-checked symbolic construction. TMEVAL is available at <https://turing-machineeval.github.io/>.

1 Introduction

Large language models (LLMs) are increasingly evaluated as general-purpose reasoners: they write executable code, solve mathematical problems, and produce formal-looking derivations (Brown et al., 2020; OpenAI et al., 2024; Touvron et al., 2023; Grattafiori et al., 2024; Chen et al., 2021; Hüttel,

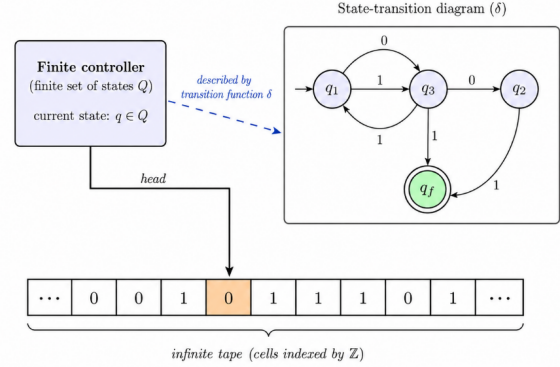


Figure 1: Schematic view of a Turing machine. A finite controller maintains the current state and follows the transition function δ . The tape head reads and writes symbols on an unbounded tape and moves left, right, or stays in place. Reaching an accepting state corresponds to acceptance; in our evaluation, undefined transitions from non-final states cause rejection.

2024; Hendrycks et al., 2021a,b; Cobbe et al., 2021; Wei et al., 2022). Yet many evaluations still test *answer production*: given a problem, produce a final answer. A stricter test is *constructive execution*: given a specification, produce an explicit mechanism whose behavior can be mechanically checked. The distinction is substantial. A model may output the right answer to a membership question or describe the right high-level algorithm without being able to construct the underlying computational object that implements it.

Formal language theory provides a principled setting for this form of evaluation. The Chomsky hierarchy relates classes of languages to computational mechanisms of increasing expressive power, including finite automata, pushdown automata, and Turing machines (Chomsky, 1956; Hopcroft et al., 2001; Sipser, 1996). The Turing machine, introduced by Turing (1937), is the canonical formal model of effective computation: every decidable language can be decided by some Turing machine. Figure 1 illustrates the machine model adopted throughout this work. Prior work has stud-

[†]Corresponding author

ied whether neural sequence models and language models can *recognize* formal languages at different levels of this hierarchy, including theoretical analyses of recurrent and transformer architectures (Weiss et al., 2018; Merrill et al., 2020; Hahn, 2020; Weiss et al., 2021; Merrill and Sabharwal, 2023; Strobl et al., 2024) as well as empirical evaluations on Dyck languages, counting, and length generalization (Bhattamishra et al., 2020; Delétang et al., 2023; Ruoss et al., 2023). However, recognizing whether a string belongs to a language is not the same as constructing the machine that decides the language. The latter requires a model to operationalize a symbolic specification into states, tape symbols, head movements, and transitions.

This distinction is directly relevant to NLP. Many practical language understanding tasks require mapping a textual specification to an executable symbolic artifact: a question to SQL (Zhong et al., 2017; Yu et al., 2018; Li et al., 2023), an instruction to an API call, a description to a regular expression, or a programming problem to code (Chen et al., 2021; Hüttel, 2024; Hendrycks et al., 2021a). Turing-machine synthesis is a controlled and demanding abstraction of this setting. The input is a formal language specification expressed in mathematical notation or precise prose; the output is a finite transition system; and correctness is evaluated by executing the generated object against oracle-labeled inputs. Unlike ordinary code generation, where solutions may rely on high-level library calls, helper predicates, or direct membership tests, a valid Turing-machine construction must expose the computation through the formal machinery of states and transitions. This makes the task a clean diagnostic for grounded, machine-checked symbolic construction.

We study executable Turing-machine synthesis from formal language specifications (§2). Given a language definition $L \subseteq \Sigma^*$, an LLM must generate complete Python code defining exactly one deterministic or nondeterministic Turing machine that decides the language. The generated artifact is parsed, validated, instantiated, and executed on frozen oracle-labeled membership tests, allowing us to separate failures of syntax, machine construction, and semantic correctness. Unlike prior formal-language evaluations that focus on string classification (Delétang et al., 2023; Ruoss et al., 2023), our task requires models to construct the recognizer itself. We instantiate this setting in TMEVAL, a benchmark of 253 formal language

specifications across 7 structural categories, containing 181 context-free and 72 non-context-free languages, each paired with 600 frozen oracle-labeled tests. Across eight evaluated LLMs, this yields 2,024 model–language candidates and up to 1,214,400 behavioral evaluations. All models are tested under identical zero-shot prompting and a strict output contract that requires a single machine object named `tm` and forbids high-level recognizers, helper predicates, or alternative artifacts. Our results show that current LLMs remain unreliable compilers from formal specifications to exact symbolic recognizers. The main contributions of this work are as follows:

1. We formulate executable Turing-machine synthesis from formal language specifications as a benchmark for language-model reasoning.
2. We introduce TMEVAL, a benchmark of 253 formal language specifications across 7 structural categories with 600 frozen oracle-labeled tests per language.
3. We develop an automated evaluation pipeline that validates machine construction, executes generated machines, and assigns each candidate to one of four terminal outcomes: `STATIC_FAIL`, `BUILD_FAIL`, `SEMANTIC_MISMATCH`, or `EXACT_PASS`.
4. We evaluate eight contemporary LLMs and show that exact TM synthesis remains very difficult. We also provide a diagnostic analysis showing that empirical difficulty is driven more by operational structure than by the classical context-free/non-context-free boundary.

2 Background

In this section, we briefly introduce the formal-language and Turing-machine concepts underlying our evaluation, define the machine model used in TMEVAL, and motivate executable TM synthesis as a benchmark for symbolic construction in large language models.

Chomsky Hierarchy. The Chomsky hierarchy organizes language classes according to the computational resources required to recognize them (Chomsky, 1956; Hopcroft et al., 2001; Sipser, 1996). Regular languages are recognized by finite automata, which operate with finitely many states and no auxiliary memory. Context-free languages extend regular languages by allowing recursive and nested structures and are recognized by pushdown automata, which augment a

finite-state controller with a single last-in–first-out stack. Context-sensitive languages are recognized by linear-bounded automata, while recursively enumerable languages are recognized by Turing machines. In this work, we focus on Turing machines used as deciders under a bounded empirical evaluation protocol: generated machines are expected to halt and return an accept/reject behavior on the benchmark inputs.

Formal Definition. Formally, let Σ be a finite set of symbols called an *alphabet*. The set Σ^* denotes all finite strings over Σ , including the empty string ϵ . A *language* is any subset $L \subseteq \Sigma^*$. A language is *decidable* if there exists a Turing machine that halts on every input $w \in \Sigma^*$ and correctly determines whether $w \in L$.

A Turing machine is a finite-state controller coupled to an unbounded tape. We write a TM as the tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F),$$

where Q is a finite set of states, Σ is the input alphabet, $\Gamma \supseteq \Sigma$ is the tape alphabet, $\square \in \Gamma \setminus \Sigma$ is the designated blank symbol, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and δ is a partial transition function. For a deterministic Turing machine (DTM), the transition function has type

$$\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\},$$

where L , R , and N denote a left move, a right move, and no movement of the tape head, respectively. For a nondeterministic Turing machine (NTM), the transition function has type

$$\delta : (Q \setminus F) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\}),$$

allowing multiple possible successors for a given state and read symbol.

Executable TM Semantics. A computation begins with the input string written on the tape, the head positioned at the leftmost input cell, and the controller in state q_0 . At each step, the machine reads the current symbol, writes a symbol, moves the head, and transitions to a new state. Reaching a state in F corresponds to acceptance. In our implementation, an undefined transition from a non-final state causes rejection. For nondeterministic machines, an input is accepted if at least one computation path reaches an accepting state.

The tape provides unbounded two-way read–write memory, distinguishing Turing machines from

weaker automata. Unlike pushdown automata, TMs can revisit earlier cells, compare distant regions of the input, and verify global relations between separated blocks. This makes them suitable for languages requiring coordinated reasoning over multiple parts of the string, including counting, copying-style dependencies, and arithmetic constraints. Turing machines therefore provide a demanding but mechanically evaluable setting for constructive reasoning. Generated transition systems are finite and can be validated structurally before execution, while behavior can be tested deterministically on frozen benchmark inputs. These properties make TM synthesis a suitable substrate for evaluating whether LLMs can construct executable computational mechanisms from formal language specifications.

3 Methodology

We describe the evaluation protocol used to test whether LLMs can synthesize executable Turing machines from formal language specifications; Figure 2 summarizes the full TMEVAL workflow. Model outputs are evaluated as executable artifacts rather than natural-language explanations or high-level programs: the generated machine must build successfully and match oracle labels on frozen membership tests.

Task Definition. We study Turing-machine synthesis from formal language specifications. Given a language description defining a target language $L \subseteq \Sigma^*$ over a finite alphabet Σ , a model must generate an executable deterministic or nondeterministic Turing machine M that decides L . The task is therefore a specification-to-machine translation problem: the model must convert a high-level symbolic constraint into an explicit transition system whose behavior can be evaluated by execution.

Formally, we view the task as a mapping

$$f_\theta : \text{LangSpec}(L, \Sigma) \rightarrow \text{TM}(M), \quad (1)$$

where $\text{LangSpec}(L, \Sigma)$ denotes the input language specification and $\text{TM}(M)$ denotes an executable Turing-machine artifact. Ideally, the generated machine satisfies

$$\forall w \in \Sigma^*, \quad M(w) = 1 \iff w \in L, \quad (2)$$

where $M(w) = 1$ denotes acceptance.

Full semantic equivalence between arbitrary machines and language specifications is not decidable

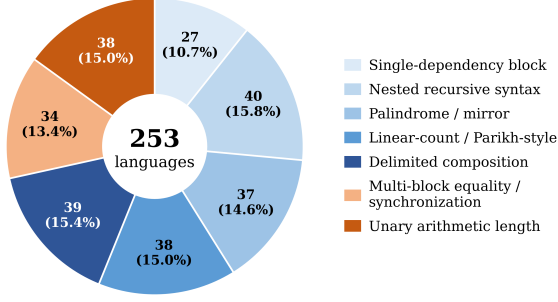


Figure 3: Distribution of the reported evaluation scope across the seven structural language categories in TMEVAL. Each slice reports the number of languages and its percentage share within the full 253-language benchmark, while the center denotes the total number of languages.

feedback, test strings, or conversational carryover are provided.

The generated object must use `automata-lib` and be instantiated as either `tm = DTM(...)` or `tm = NTM(...)`. The permitted constructor fields are `states`, `input_symbols`, `tape_symbols`, `transitions`, `initial_state`, `blank_symbol`, and `final_states`. These fields require the model to specify the finite control, input alphabet, tape alphabet, transition relation, initial state, blank symbol, and accepting states explicitly.

The output contract prohibits natural-language explanations, markdown, comments, print statements, tests, helper functions, custom classes, custom membership predicates, parser libraries, regular expressions, and alternative formal artifacts such as PDA, CFG, DFA, or NFA definitions. These restrictions prevent models from bypassing Turing-machine synthesis with high-level recognizers. Under our protocol, acceptance behavior must arise only from the transition table of the generated DTM or NTM. §A.7 provides a complete example of a valid `automata-lib` artifact following this output contract.

Frozen Test Suites and Oracles. Each language is paired with two frozen oracle-labeled test suites, $\mathcal{S}_i^{(1)}$ and $\mathcal{S}_i^{(2)}$, each containing 300 strings. Each suite contains 150 positive and 150 negative examples, giving 600 test cases per language. The same 600 cases are reused identically across all models. For the reported 253-language scope, this gives $253 \times 600 = 151,800$ expected case evaluations per model, and $8 \times 253 \times 600 = 1,214,400$ maximum case evaluations across all models.

Positive examples are strings that satisfy the target language definition and include minimal valid strings, boundary-valid cases, longer constructions,

Table 1: Illustrative frozen test cases for one representative language. The full benchmark uses 600 oracle-labeled tests per language, balanced across two suites.

String	Oracle	Rationale
Language: $L = \{0^n 1^n \mid n > 0\}$		
01	Accept	Minimal valid string with one matched pair.
000111	Accept	Valid string with three matched pairs.
001	Reject	Off-by-one count mismatch.
0101	Reject	Invalid block order.
00011	Reject	Near-boundary case with one missing 1.
00111	Reject	Near-boundary case with one extra 1.

deeper nesting, mirror structures, and arithmetic length relations where applicable. Negative examples include both simple violations and near-boundary perturbations: simple negatives break explicit constraints such as alphabet membership, block order, delimiter placement, or required surface form, while near-boundary negatives are derived by minimally modifying valid strings, including off-by-one counting errors, broken mirror relations, malformed nesting, incorrect copy relations, invalid arithmetic constraints, and violations of strict versus non-strict inequalities. Table 1 shows this construction for a representative language, and the same principle is applied across all categories, with additional examples provided in §A.5.

Each test string w receives a binary oracle label:

$$y = \begin{cases} 1, & \text{if } w \in L_i, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

All test suites are fixed before model evaluation. Model outputs are never used to generate, filter, rebalance, or modify test cases, and models do not receive the test strings in the prompt.

Automated Evaluation Pipeline and Metrics. We evaluate every generated candidate using a deterministic pipeline that separates artifact validity from semantic correctness. Model outputs are normalized, parsed, and checked against a strict contract requiring a single Turing-machine object named `tm`. Static-valid programs are executed in a controlled environment and validated using the underlying `automata` library. Each buildable

machine is then evaluated on 600 frozen oracle-labeled membership tests. Full procedural details are provided in §A.8.

Every candidate terminates in exactly one of four mutually exclusive outcomes: `STATIC_FAIL` if the output violates the code contract; `BUILD_FAIL` if it parses but cannot instantiate a valid TM; `SEMANTIC_MISMATCH` if the machine builds but fails at least one oracle test; and `EXACT_PASS` if it matches all 600 oracle labels. Concrete examples are provided in §A.11.

We report metrics at both candidate and case level. Let N_{cand} denote the number of model-language candidates, N_{sem} and N_{exact} the numbers of semantic-mismatch and exact-pass candidates, and $N_{\text{run}} = N_{\text{sem}} + N_{\text{exact}}$ the number of runnable candidates. Let C_{run} denote the number of correctly classified test cases among runnable candidates. We compute

$$\text{EP} = \frac{N_{\text{exact}}}{N_{\text{cand}}}, \quad (4)$$

$$\text{Runnable} = \frac{N_{\text{run}}}{N_{\text{cand}}}, \quad (5)$$

$$\text{E2E} = \frac{C_{\text{run}}}{600 N_{\text{cand}}}, \quad (6)$$

$$\text{Acc}_c = \frac{C_{\text{run}}}{600 N_{\text{run}}}. \quad (7)$$

Exact pass rate (EP) is the primary candidate-level metric: a candidate receives credit only if it matches all 600 oracle labels. E2E is the primary case-level metric and assigns zero case credit to candidates that fail before behavioural execution. Runnable rate and conditional accuracy are diagnostic metrics. In particular, conditional accuracy measures behavior only on the model-dependent subset of runnable candidates and is therefore not used for primary model ranking.

We additionally report the rates of all four terminal outcomes and aggregate EP, runnable rate, E2E, and terminal-outcome rates by branch and structural category. This analysis tests whether difficulty is explained by the CFL/non-CFL distinction or by finer-grained operational structure. Additional denominator and statistical-reporting details are provided in §A.10.

4 Results and Analysis

We evaluate 8 models on 253 formal languages, yielding 2,024 model-language candidates and a maximum of 1,214,400 case evaluations. Since

1,085 candidates fail before behavioural execution, 563,400 case records are observed. The central result is that executable Turing-machine synthesis remains far from solved. Only 225 of 2,024 candidates pass all 600 tests, yielding an exact-pass rate of 11.12%. More importantly, failures are not confined to final behavioural errors. Models fail at three distinct levels: they emit invalid artifacts, construct invalid machines, and produce executable machines that recognize the wrong language.

The main paper reports six compact tables that establish the core empirical claims: the global benchmark outcome (Table 2), the failure-stage funnel (Table 3), the metric interpretation table (Table 4), the model leaderboard (Table 5), category-level results (Table 6), and semantic error direction (Table 7). Detailed diagnostic tables are deferred to §C, and statistical analysis is deferred to §B.

4.1 Core Findings

Overall Outcome. Table 2 summarizes the benchmark-level outcome. Exact pass is deliberately strict at the candidate level: a candidate receives exact-pass credit only if it produces the correct accept/reject decision on all 600 frozen tests. End-to-end case score (E2E) is case-level, but uses a fixed denominator by assigning zero credit to candidates that fail before behavioural testing. The most important observation is not simply that exact pass is low. More than half of all candidates never reach behavioural evaluation. The benchmark therefore does not merely test whether a model can choose the correct accept/reject label. It tests whether a model can compile a formal-language specification into an executable symbolic object that is syntactically valid, structurally well-formed, and semantically faithful.

Failure-Stage Decomposition. Each candidate terminates in exactly one of four mutually exclusive pipeline stages: static failure, build failure, semantic mismatch, or exact pass. Table 3 gives the full decomposition. This decomposition is the benchmark’s central diagnostic value. Among the 1,799 non-passing candidates, 1,085 fail before behavioural execution. Thus, the dominant coarse failure mode is not incorrect recognition, but failure to emit a valid recognizer at all. At the same time, 714 candidates build successfully but still recognize the wrong language. A benchmark that only evaluates final answers, or only checks whether generated code runs, would miss both sides of this failure structure.

Table 2: Global benchmark outcome. Exact pass is candidate-level; E2E is case-level with invalid artifacts counted as zero.

Metric	Value
Languages	253
Models	8
Model-language candidates	2,024
Tests per candidate	600
Maximum case evaluations	1,214,400
Runnable case records observed	563,400
Exact-pass candidates	225/2,024
Exact-pass rate	11.12%
End-to-end case score	31.00%
Runnable candidates	939/2,024
Runnable rate	46.39%
Invalid-artifact candidates	1,085/2,024
Invalid-artifact rate	53.61%
Languages solved by at least one model	178/253
Languages unsolved by every model	75/253

Table 3: Failure-stage funnel. Static and build failures occur before behavioural testing. Semantic mismatch means the machine builds successfully but fails at least one oracle test.

Terminal stage	Candidates	Share
STATIC_FAIL	522	25.79%
BUILD_FAIL	563	27.82%
SEMANTIC_MISMATCH	714	35.28%
EXACT_PASS	225	11.12%
Invalid artifact	1,085	53.61%
Runnable artifact	939	46.39%

Why Exact Pass and E2E Are Both Necessary.

A tempting alternative is conditional case accuracy (Acc_c): accuracy over only the runnable candidates of each model. We do not use it as a primary metric because its denominator varies sharply across models. For example, gemini-2.5-pro has only 14 runnable candidates, while claude-sonnet-4.5 has 238. Conditional accuracy therefore measures performance on a model-specific surviving subset, not on the benchmark as a whole. This distinction changes the interpretation of model quality. On conditional accuracy alone, gemini-2.5-pro appears stronger than claude-sonnet-4.5 by 23.8 percentage points. On E2E, gemini-2.5-pro trails claude-sonnet-4.5 by 52.4 percentage points because almost all of its candidates fail before behavioural testing. We therefore use exact pass as the primary candidate-level metric and E2E as the primary case-level metric.

Oracle Coverage and Saturation. Seventy-five

Table 4: Conditional case accuracy is not directly comparable across models because each model has a different runnable subset. E2E restores comparability by assigning zero credit to invalid artifacts.

Model	Run.	Acc_c	E2E	Gap
gpt-5.2	189	96.1	71.8	24.3
gemini-2.5-pro	14	84.5	4.7	79.8
qwen3-235B	172	62.1	42.2	19.9
claude-sonnet-4.5	238	60.7	57.1	3.6
gemma-3-27B	4	59.3	0.9	58.4
llama-3.3-70B	105	59.5	24.7	34.8
deepseek-v3.2	41	58.7	9.5	49.2
mistral-large	176	53.3	37.0	16.2

languages are unsolved by all models. Oracle coverage is dominated by gpt-5.2, which solves 168 of the 178 solvable languages and is the unique solver for 137. A best-per-language oracle therefore improves over gpt-5.2 by only 10 additional languages, indicating limited complementary coverage across models for exact TM synthesis. More details in §C.2.

Within-Test-Set Reproducibility. Each candidate is evaluated on two parallel 300-case test suites from the same generator. Aggregate accuracy is nearly identical across suites: 66.77% vs. 66.85% (difference 0.08 pp), indicating stable within-test-set behavior. Per-model results are in §C.6. This is a reproducibility check within the frozen test construction, not a robustness claim to new or adversarial inputs.

4.2 Model-Level Results

Table 5 reports per-model performance. We report exact pass with Wilson 95% confidence intervals at $N = 253$, along with runnable rate, E2E, and terminal failure counts.

One model substantially leads. gpt-5.2 is the only model that exact-passes a majority of languages: 168/253. Its exact-pass confidence interval [60.4, 71.9] is disjoint from every other model’s interval. The gap to the next model is 59.3 percentage points (66.4% vs. 7.1%), showing that exact TM synthesis is currently concentrated in one model rather than broadly distributed across model families. We suspect two possible explanations: (i) prior exposure via training or fine-tuning on related experimental or code-generation setups, and (ii) genuinely stronger capacity to represent and manipulate structured computational constraints underlying the task. Either way, future work should

Table 5: Model leaderboard. $N = 253$ candidates per model. EP denotes exact-pass rate. Runnable rate is the fraction of candidates that pass static and build validation. Failure-stage columns sum to 253.

Model	Exact	EP%	CI95	Run%	E2E%	STATIC	BUILD	SEMANTIC
gpt-5.2	168	66.4	[60.4, 71.9]	74.7	71.8	61	3	21
claude-sonnet-4.5	18	7.1	[4.6, 11.0]	94.1	57.1	2	13	220
qwen3-235B	17	6.7	[4.2, 10.5]	68.0	42.2	5	76	155
gemini-2.5-pro	9	3.6	[1.9, 6.6]	5.5	4.7	239	0	5
llama-3.3-70B	7	2.8	[1.4, 5.6]	41.5	24.7	20	128	98
mistral-large	4	1.6	[0.6, 4.0]	69.6	37.0	3	74	172
deepseek-v3.2	2	0.8	[0.2, 2.8]	16.2	9.5	189	23	39
gemma-3-27B	0	0.0	[0.0, 1.5]	1.6	0.9	3	246	4

include probing studies, mechanistic analyses, and latent-space investigations to verify whether the observed gains reflect true algorithmic generalization in formal machine synthesis.

Runnable rate and exact correctness diverge. claude-sonnet-4.5 has the highest runnable rate (94.1%), but only 7.1% exact pass. This is one of the clearest findings of the benchmark: valid executable artifacts are not sufficient evidence of formal correctness. A model can produce machines that parse, build, and run, yet still implement the wrong language.

Models fail in different regimes. Although exact-pass rates are low below gpt-5.2, the failure profiles are not identical. claude-sonnet-4.5, qwen3-235B, and mistral-large are semantic-bound: they frequently emit runnable machines, but these machines implement the wrong language. In contrast, gemini-2.5-pro, deepseek-v3.2, and gemma-3-27B are artifact-bound: most of their candidates fail static or build validation. llama-3.3-70B lies between these regimes, with a build-heavy profile. These distinctions matter because the remedies differ. Syntax repair may help static-bound models, schema validation may help build-bound models, but neither solves semantic mismatch.

Statistical Reliability of Model Comparisons. Because every model is evaluated on the same 253 languages, we conduct inference at the language level rather than treating the 600 cases within a language as independent. Paired exact McNemar tests with Holm correction over all 28 model pairs confirm that gpt-5.2 outperforms every other model ($p_{\text{Holm}} \leq 1.79 \times 10^{-37}$). Across these comparisons, gpt-5.2 alone solves 153–168 discordant languages, whereas the reverse counts are only 0–6. Differences among non-leading models are

less stable: claude-sonnet-4.5 and qwen3-235B differ by 0.4 percentage points and are not distinguishable after correction ($p_{\text{Holm}} = 1.00$). A language-level bootstrap for E2E yields positive differences between gpt-5.2 and every alternative; the smallest gap is against claude-sonnet-4.5 at 14.7 points (95% CI [8.9, 20.4]). These intervals quantify variation over languages, not decoding seeds or prompt variants. Full analyses appear in §B.

4.3 Category-Level Results

We also analyze whether difficulty follows the formal distinction between context-free and non-context-free languages. Table 6 shows that it does not: branch-level exact-pass rates are nearly identical for CFL and non-CFL languages (11.12% vs. 11.11%), with within-branch variation far exceeding between-branch differences. The hardest categories are both context-free (CFL-2 and CFL-5), while non-CFL categories are not uniformly harder (NCFL-1: 12.5%, NCFL-4: 9.9%). Oracle coverage further highlights this pattern: CFL-1 is solved on 96.3% of its languages by at least one model, whereas CFL-2 and CFL-5 drop to 42.5% and 41.0%, together accounting for 46 of the 75 universally unsolved languages. Overall, difficulty is driven more by operational structure than by formal language-family membership, as the benchmark evaluates machine construction rather than recognition.

4.4 Error Analysis

We now analyze the non-passing candidates in more detail. This subsection remains in the main results because the error types explain why exact synthesis fails and what kinds of future improvements would be required.

Table 6: Per-category aggregates over all 8 models, sorted by exact-pass rate. Oracle%: % of languages solved by ≥ 1 model.

Branch	Cat.	#Langs	N	EP%	STATIC%	BUILD%	SEMANTIC%	E2E%	Oracle%
CFL	CFL-5	39	312	6.1	28.9	28.2	36.9	26.7	41.0
CFL	CFL-2	40	320	7.2	25.6	33.4	33.8	25.3	42.5
Non-CFL	NCFL-4	38	304	9.9	28.6	28.6	32.9	29.0	68.4
CFL	CFL-3	37	296	12.2	26.4	22.6	38.9	32.5	89.2
Non-CFL	NCFL-1	34	272	12.5	26.8	26.1	34.6	32.6	82.4
CFL	CFL-4	38	304	15.5	20.7	29.3	34.5	35.5	84.2
CFL	CFL-1	27	216	16.7	22.7	25.0	35.7	38.2	96.3

Table 7: First-failure decomposition in semantic mismatches.

First-failure type	N	Share
Gold accept, model rejects	583	81.7%
Gold reject, model accepts	131	18.3%
REJECT_HALTED	557	78.0%
REJECT_ALL_BRANCHES_HALTED	26	3.6%
ACCEPT	131	18.4%

Static failures. There are 522 static failures. The dominant root cause is unterminated string literals in the emitted artifact, accounting for 363 failures, or 69.5% of all static failures. These failures are heavily concentrated in two models: gemini-2.5-pro contributes 239 static failures, and deepseek-v3.2 contributes 189. We provide the full static-error decomposition in §C.3.

Build failures. There are 563 build failures: artifacts that parse but do not define a valid TM. The dominant build error is `FinalStateError`, which accounts for 381 failures, or 67.7% of all build failures. More analysis is provided in §C.4.

Semantic mismatches. The most important errors are semantic mismatches. These are the 714 candidates that pass static and build validation but fail at least one of the 600 oracle tests. They represent the strongest evidence that runnable code is not equivalent to correct formal synthesis.

The direction of semantic error is strongly asymmetric. On the first observed failing case, 81.7% of semantic mismatches reject a string that belongs to the target language. At case level, false negatives outnumber false positives by 6.6:1. The semantic-bound models therefore tend to undergenerate the target language: their errors are dominated by false negatives, suggesting recognizers that are too restrictive rather than too permissive.

Per-model confusion confirms the same pattern. `claude-sonnet-4.5`, `qwen3-235B`, and `mistral-large` have high specificity but low re-

call on accepted strings. Their machines are conservative recognizers: they frequently decline to accept valid members of the language. In contrast, `gpt-5.2` maintains both high recall and high specificity. This balance between recall and specificity is more informative than raw conditional accuracy alone, because exact synthesis requires agreement on both members and non-members of the language. The full confusion matrix and near-miss analysis are provided in §C.5.

5 Concluding Remarks

We introduced TMEVAL, an executable benchmark for evaluating whether large language models can synthesize Turing machines from formal language specifications. Unlike recognition-based formal-language benchmarks, TMEVAL requires models to construct an explicit deterministic or nondeterministic machine whose behavior is verified against frozen oracle-labeled tests. Overall, our findings across 253 languages, 8 models, and 2,024 model-language candidates support six conclusions. First, exact executable TM synthesis remains difficult, with only 11.12% exact-pass. Second, failures are multi-stage: 53.61% fail before execution and 35.28% build but implement the wrong language. Third, `gpt-5.2` substantially leads, but success is not broadly distributed across model families, warranting mechanistic analysis. Fourth, failure modes are heterogeneous, splitting into artifact- and semantic-level errors. Fifth, the CFL/non-CFL boundary is not the main difficulty axis; nested recursion, synchronization, and composition better explain performance. Sixth, the benchmark remains unsaturated, with 75 languages unsolved by all models. Overall, TMEVAL highlights a persistent gap between recognizing formal patterns and constructing exact executable recognizers, suggesting that reliable symbolic construction remains an open challenge for current LLMs.

Limitations

TMEVAL evaluates executable Turing-machine synthesis under a controlled but necessarily bounded protocol. First, EXACT_PASS requires agreement with all 600 frozen oracle-labeled tests per language and does not imply full equivalence over Σ^* . Although the test suites include positives, simple negatives, and near-boundary perturbations, finite testing cannot cover all longer or adversarial inputs. This is an inherent limitation of any empirical evaluation of undecidable equivalence; future work should extend oracle construction or adopt formal verification where possible to tighten guarantees beyond finite behavioral agreement.

Second, the output contract is intentionally restrictive. Models must emit a single Turing-machine object and cannot use helper functions, regular expressions, parser libraries, or direct membership predicates. This isolates machine synthesis but does not reflect best-case performance under more flexible or tool-augmented code generation. Rather than a weakness, this design choice enforces a clean separation between symbolic construction and auxiliary computation; future work can study how tool use or relaxed interfaces trade off against faithful machine construction.

Third, all experiments use a zero-shot, single-generation protocol. We do not evaluate few-shot prompting, self-correction, execution feedback, verifier-guided decoding, or iterative repair, any of which may improve performance. This choice reflects a minimal baseline for measuring raw synthesis ability; future work should explore interactive and feedback-driven settings to quantify how much improvement comes from scaffolding versus intrinsic reasoning.

Fourth, the benchmark spans 253 languages across 7 structural categories but is not exhaustive. Other decidable families, larger alphabets, deeper arithmetic constraints, or alternative machine formalisms may yield different failure modes. This scope is sufficient to expose structural trends but not to characterize the full landscape of formal synthesis; future extensions should broaden coverage to test generalization across richer language classes and representations.

Finally, results depend on specific model versions and API-accessible systems. Provider-side updates may affect future generations, and some formal-language patterns or automata-lib conventions may appear in public training data. We

therefore release generated artifacts and execution records to support reproducibility. While this may introduce potential contamination and version sensitivity, it also reflects realistic deployment conditions; future work should perform controlled decontamination studies and longitudinal evaluations across model updates to isolate true capability shifts.

Ethical Considerations and Potential Risks

Although TMEVAL is a diagnostic benchmark and does not deploy models in real-world decision settings, it raises ethical considerations related to interpretation and downstream use. In particular, strong performance on executable Turing-machine synthesis may be misinterpreted as general-purpose reliability in automated system construction, despite the controlled and synthetic nature of the task. This can lead to overconfidence in deploying language models for formal system generation without sufficient verification.

A second consideration is the potential dual-use nature of advanced code synthesis capabilities, where improved ability to generate structured computational artifacts could be misapplied in unsafe or unintended contexts. However, TMEVAL itself does not increase this capability; it only measures it under constrained conditions. We therefore position the benchmark as an auditing tool for understanding model limitations, emphasizing the need for human verification and interpretability in any downstream use of generated formal systems.

Reproducibility Statement

To support reproducibility, we publicly release the code, data, evaluation pipeline, and generated outputs required to reproduce the results reported in this paper. The release includes the complete TMEVAL benchmark, all 253 language specifications, frozen oracle-labeled test suites, model prompts, inference configurations, and execution scripts. We additionally provide the generated model outputs, stage-wise evaluation traces (STATIC_FAIL, BUILD_FAIL, SEMANTIC_MISMATCH, and EXACT_PASS), and scripts for reproducing the reported metrics and analyses. TMEVAL is available at <https://turing-machinereval.github.io/>.

AI Usage Disclosure

Large language models, including ChatGPT, Gemini, Claude, and other models were used as part of the experimental setup for generating model outputs in the evaluation of TMEVAL. In addition, AI-assisted tools were used for non-experimental support tasks, including grammar checking, editing, figure ideation, and visualization assistance. All AI-assisted content was manually reviewed and verified by the authors. The authors retain full responsibility for the integrity, correctness, and interpretation of all results and reported analyses.

References

- Anthropic. 2025. [Introducing claude sonnet 4.5](#). Accessed: 2026-05-26.
- Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. [On the Ability and Limitations of Transformers to Recognize Formal Languages](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online. Association for Computational Linguistics.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language models are few-shot learners](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *Preprint*, arXiv:2107.03374.
- N. Chomsky. 1956. [Three models for the description of language](#). *IRE Transactions on Information Theory*, 2(3):113–124.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *Preprint*, arXiv:2110.14168.
- Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Naveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsen, Idan Szpektor, Nan-Jiang Jiang, and 3416 others. 2025. [Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities](#). *Preprint*, arXiv:2507.06261.
- DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. [Deepseek-v3.2: Pushing the frontier of open large language models](#). *Preprint*, arXiv:2512.02556.
- Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. 2023. [Neural networks and the Chomsky hierarchy](#). In *Proceedings of the 11th International Conference on Learning Representations*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Michael Hahn. 2020. [Theoretical limitations of self-attention in neural sequence models](#). *Transactions of the Association for Computational Linguistics*, 8:156–171.
- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021a. [Measuring coding challenge competence with APPS](#). In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. [Measuring mathematical problem solving with the MATH dataset](#). In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.
- John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*, 2nd edition. Addison-Wesley.
- Hans Hüttel. 2024. [On program synthesis and large language models](#). *Communications of the ACM*, 68(1):33–35.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as

- a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.
- William Merrill and Ashish Sabharwal. 2023. [The parallelism tradeoff: Limitations of log-precision transformers](#). *Transactions of the Association for Computational Linguistics*, 11:531–545.
- William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. 2020. [A formal hierarchy of RNN architectures](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459. Association for Computational Linguistics.
- Mistral AI. 2024. [Mistral large](#). Accessed: 2026-05-26.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bannani, Shane Legg, and Joel Veness. 2023. [Randomized positional encodings boost length generalization of transformers](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1889–1903. Association for Computational Linguistics.
- Michael Sipser. 1996. *Introduction to the Theory of Computation*, volume 27. Association for Computing Machinery, New York, NY, USA.
- Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. 2024. [What formal languages can transformers express? A survey](#). *Transactions of the Association for Computational Linguistics*, 12:543–561.
- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. [Gemma 3 technical report](#). *Preprint*, arXiv:2503.19786.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. [Llama: Open and efficient foundation language models](#). *Preprint*, arXiv:2302.13971.
- A. M. Turing. 1937. [On computable numbers, with an application to the entscheidungsproblem](#). *Proceedings of the London Mathematical Society*, s2-42(1):230–265.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.
- Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. [On the practical computational power of finite precision RNNs for language recognition](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745. Association for Computational Linguistics.
- Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. [Thinking like transformers](#). In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11080–11090. PMLR.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. [Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics.
- Victor Zhong, Caiming Xiong, and Richard Socher. 2017. [Seq2sql: Generating structured queries from natural language using reinforcement learning](#). *Preprint*, arXiv:1709.00103.

A Methodological Details

This appendix provides additional details about benchmark construction, category rationale, representative language examples, test-case generation, output validation, metric computation, and reproducibility. The main paper reports the stable 253-language evaluation scope; the appendix documents how the scope was obtained and how the evaluation protocol is implemented.

A.1 Benchmark Construction and Filtering

The benchmark was developed through a multi-stage expert-guided construction process. An initial pool of candidate languages was collected from standard theory-of-computation textbooks, formal-language exercises, instructional materials, and expert-generated variants. The goal was to cover a broad range of computational structures rather than only canonical textbook examples.

Each candidate language was recorded with an explicit alphabet, a formal or textual language definition, an intended branch label, and a preliminary structural category. Candidates were retained only if their membership condition was precise enough to support oracle construction and executable evaluation. Ambiguous descriptions were either rewritten into precise specifications or removed.

The candidate pool was then filtered for duplicates, near-duplicates, underspecified definitions, inconsistent alphabets, unclear empty-string behavior, ambiguous inequality conditions, and cases whose membership condition could not be reliably converted into oracle-labeled test suites. The final reported evaluation uses the 253-language scope. Earlier nonlinear arithmetic instances are excluded from the reported main analysis because they introduce oracle-budget ambiguities that would confound the interpretation of model behavior.

A.2 Category Rationale

The categories are based on the operational structure required to decide the language.

CFL-1: Single-dependency block languages. These languages test whether a model can encode one equality, inequality, or proportional relation between separated regions of a string.

CFL-2: Nested recursive syntax. These languages require hierarchical reasoning over balanced delimiters, expression-like structures, programming-language fragments, JSON-like

forms, XML-like fragments, or related recursive patterns.

CFL-3: Palindrome and mirror languages. These languages require matching substrings against their reverse, often around a delimiter or central region.

CFL-4: Linear-count and Parikh-style languages. These languages require enforcing linear equalities, inequalities, modular constraints, or symbol-count relations.

CFL-5: Delimited composition. These languages require reasoning over separators, markers, and fixed composition patterns that determine how substrings must relate.

NCFL-1: Multi-block equality and synchronization. These languages require coordinating three or more symbol counts or separated string regions, often through repeated marking and comparison.

NCFL-4: Unary arithmetic length. These languages include arithmetic conditions over unary block lengths, requiring arithmetic tests to be expressed through tape operations rather than high-level predicates.

This category design is important because the results show that empirical difficulty is not explained by the CFL/non-CFL boundary alone. Instead, difficulty depends on the operational structure that the model must compile into a transition system.

A.3 Representative Language Examples by Category

Table 8 provides representative examples from each structural category. These examples clarify the operational distinction between categories that may look similar at the surface level but require different machine-construction strategies.

A.4 Test-Case Generation Details

Each language has two frozen suites of 300 examples. Each suite contains 150 positive and 150 negative examples, yielding 600 tests per language. The two-suite design supports the within-test-set reproducibility analysis in Appendix C.6.

Positive examples are generated to cover the intended valid behavior of the language. Depending on the category, this includes minimal strings, boundary cases, long strings, nested structures, mirror pairs, delimiter-based compositions,

Table 8: Representative examples from each structural category in TMEVAL.

Branch	Category	Type	Example language
CFL	CFL-1	Single dependency	$L = \{0^i 1^j 2^k \mid i, j, k \geq 0 \text{ and } i = k\}$
CFL	CFL-1	Single dependency	$L = \{a^n b^m c^m \mid m, n \in \mathbb{N}\}$
CFL	CFL-1	Disjunctive dependency	$L = \{0^m 1^n 2^p \mid m = n \text{ or } m = p, m, n, p \geq 0\}$
CFL	CFL-2	Balanced delimiters	$L = \{w \in \{(\,,)\}^* \mid w \text{ is a well-balanced parenthesis string}\}$
CFL	CFL-2	Multi-bracket nesting	BalDelim = $\{w \in \Sigma^* \mid w \text{ is properly balanced over } (\,,), [], \{\,,\}\}$
CFL	CFL-2	Programming-language syntax	Well-formed postfix expressions over num and binary operators $\{+, -, *, /\}$
CFL	CFL-3	Mirror form	$L = \{x \# x^R \mid x \in \{a, b\}^*\}$
CFL	CFL-3	Palindrome	$L = \{w \in \{a, b\}^* \mid \text{mir}(w) = w\}$
CFL	CFL-3	Delimited reverse	$L = \{x \# x^R \mid x \in \{a, b\}^*, x \text{ is odd}\}$
CFL	CFL-4	Linear inequality	$L = \{a^n b^m \mid m > n, m, n > 0\}$
CFL	CFL-4	Parikh equality	$L = \{w \in \{a, b\}^* \mid \#a(w) = \#b(w)\}$
CFL	CFL-4	Linear relation	$L = \{a^n b^m \mid n \leq m \leq 2n\}$
CFL	CFL-5	Delimited equal-length composition	$L = \{xay \mid x, y \in \{a, b\}^* \text{ and } x = y \}$
CFL	CFL-5	Boundary-sensitive composition	$L = \{w \in \{0, 1\}^* \mid w \text{ starts and ends with the same symbol}\}$
CFL	CFL-5	Delimiter-centered composition	$L = \{ab^i c^j a^k b^\ell c \mid i, j, k, \ell \geq 0\}$
Non-CFL	NCFL-1	Three-symbol equality	$L = \{w \in \{a, b, c\}^* \mid w _a = w _b = w _c\}$
Non-CFL	NCFL-1	Four-symbol equality	$L = \{w \in \{0, 1, 2, 3\}^* \mid \#0(w) = \#1(w) = \#2(w) = \#3(w)\}$
Non-CFL	NCFL-1	Cross-block synchronization	$L = \{a^n b^{2n} d^n \mid n \geq 1\}$
Non-CFL	NCFL-4	Unary arithmetic with primality	$L = \{a^m b^n \mid m > n, \text{ or } m \text{ is prime and } n \geq m\}$
Non-CFL	NCFL-4	Conditional primality	$L = \{a^i b^j \mid i, j \geq 0 \text{ and if } i = 1 \text{ then } j \text{ is prime}\}$
Non-CFL	NCFL-4	Unary arithmetic length	Unary block-length constraints requiring arithmetic tests over the number of repeated symbols.

multi-block synchronizations, and unary arithmetic lengths.

Negative examples are generated using two strategies. Simple negatives violate visible constraints such as alphabet membership, block order, delimiter placement, or required surface pattern. Near-boundary negatives are obtained by minimally perturbing valid examples. These perturbations include off-by-one count errors, broken mirror correspondence, malformed nesting, invalid delimiter placement, incorrect block synchronization, and arithmetic-condition violations.

All test cases are generated before any model is evaluated. The same test cases are used for every model, ensuring that model comparisons are not affected by test-set variation.

A.5 Expanded Test-Case Examples

Table 9 gives representative test-case patterns across categories. These are illustrative examples of how positive, simple-negative, and near-boundary-negative strings are used to test behavioral agreement.

A.6 Output Extraction and Validation Rules

The evaluator applies deterministic extraction and validation rules. A response is invalid if it is empty, contains only markdown or prose, contains placeholder code, defines no recoverable tm object, or defines multiple competing artifacts.

The static validator rejects outputs that con-

tain helper functions, custom classes, direct membership predicates, regular expressions, parser libraries, test code, print statements, or alternative formal artifacts. The purpose of these restrictions is to ensure that the model solves the requested task: synthesis of a Turing machine represented through the automata-lib DTM or NTM interface.

After static validation, the candidate is executed in a controlled environment. The resulting tm object must pass library-level validation, including checks on state declarations, alphabet membership, transition consistency, blank symbol validity, initial state validity, movement directions, and accepting-state declarations.

A.7 Example Valid Turing-Machine Artifact

Figure 4 shows a compact example of a valid automata-lib artifact following the output contract. This toy machine is included only to illustrate the required artifact form; the benchmark evaluation uses model-generated artifacts for the target language specifications.

A.8 Automated Evaluation Pipeline

Every generated candidate is evaluated by a deterministic automated pipeline. The pipeline separates artifact validity from semantic correctness, allowing us to identify where each candidate fails.

Output extraction and static validation. The raw model response is first normalized and inspected. Empty responses, incomplete programs,

Table 9: Representative test-case construction across categories. Examples illustrate positive cases, simple negatives, and near-boundary negatives.

Category	Language	Example strings	Purpose
CFL-1	$L = \{a^n b^m c^n \mid n, m \geq 1\}$	abc, aabbcc; abbc, aabcc	Tests a separated equality between the first and final block.
CFL-2	Balanced parentheses	$()$, $(())$; $(((, ())$	Tests recursive nesting and malformed near-boundary strings.
CFL-3	$L = \{x \# x^R \mid x \in \{a, b\}^*\}$	ab#ba; ab#ab, aba#ba	Tests mirror matching around a delimiter.
CFL-4	$L = \{a^n b^m \mid n \leq m \leq 2n\}$	ab, aabb, aabbbb; aaab, aabbbbb	Tests lower and upper linear count bounds.
CFL-5	$L = \{xay \mid x, y \in \{a, b\}^*, x = y \}$	a, bab; ba, bbab	Tests delimiter-centered equal-length composition.
NCFL-1	$L = \{w \mid w _a = w _b = w _c\}$	abc, aabbcc; aabcc, abbc	Tests multi-symbol count synchronization.
NCFL-4	$L = \{a^m b^n \mid m > n \text{ or } m \text{ prime and } n \geq m\}$	aaab, aaabbbb; aaaabbbb, abb	Tests unary arithmetic and primality-conditioned membership.

markdown-only outputs, placeholder code, and responses containing unrecoverable prose outside the program are marked invalid. Recovered code is parsed using Python’s abstract syntax tree. The static validator enforces the output contract: exactly one top-level assignment named `tm`, no auxiliary functions or classes, no additional executable logic, no disallowed constructors, and no disallowed keyword arguments. Outputs that violate this stage are assigned `STATIC_FAIL`.

Machine instantiation and build validation.

Static-valid programs are executed in a controlled environment to instantiate the `tm` object. The machine is then validated by the underlying automata library. This step checks consistency of states, input symbols, tape symbols, blank symbol, initial state, final states, transition definitions, and movement directions. Programs that cannot be imported, instantiated, or validated are assigned `BUILD_FAIL`.

Behavioral execution. Each buildable machine is executed on the 600 frozen test strings associated with the target language. For every input string, the evaluator records whether the machine accepts or rejects under a fixed execution policy.

Oracle comparison and terminal outcome. The observed behavior is compared with the oracle label for each test case. A candidate is assigned `EXACT_PASS` if all 600 cases are correct. If the machine builds but fails at least one oracle comparison, it is assigned `SEMANTIC_MISMATCH`. Thus, every candidate terminates in exactly one of four stages: $\mathcal{T} = \{\text{STATIC_FAIL}, \text{BUILD_FAIL},$

`SEMANTIC_MISMATCH`, `EXACT_PASS\}`. §A.11 gives examples of each terminal outcome.

A.9 Models Evaluated

We evaluate eight contemporary large language models: gpt-5.2 (OpenAI et al., 2024), gemini-2.5-pro (Comanici et al., 2025), qwen3-235B (Yang et al., 2025), claude-sonnet-4.5 (Anthropic, 2025), gemma-3-27B (Team et al., 2025), llama-3.3-70B (Grattafiori et al., 2024), deepseek-v3.2 (DeepSeek-AI et al., 2025), and mistral-large (Mistral AI, 2024).

A.10 Metric Computation Details

Equations 4–7 define all reported metrics. For each model, the exact-pass denominator is 253 model–language candidates. Across all eight models, the denominator is 2,024 candidates.

E2E retains 600 expected cases for every candidate, including candidates that fail before behavioral execution. Therefore, the full E2E denominator is

$$2,024 \times 600 = 1,214,400. \quad (8)$$

Exact pass is binary at the candidate level: a machine either passes all 600 tests or it does not. Case-level metrics are used to examine partial behavior and semantic-error patterns.

We report Wilson 95% confidence intervals for each model’s exact-pass rate over its 253 model–language candidates. We avoid fine-grained model rankings when the corresponding confidence intervals overlap.

```

from automata.tm.dtm import DTM

tm = DTM(
    states={'q_even', 'q_odd', 'q_accept', 'q_reject'},
    input_symbols={'0', '1'},
    tape_symbols={'0', '1', ' '},
    transitions={
        'q_even': {
            '0': ('q_odd', '0', 'R'),
            '1': ('q_odd', '1', 'R'),
            ' ': ('q_reject', ' ', 'N')
        },
        'q_odd': {
            '0': ('q_even', '0', 'R'),
            '1': ('q_even', '1', 'R'),
            ' ': ('q_accept', ' ', 'N')
        },
        'q_reject': {
            '0': ('q_reject', '0', 'N'),
            '1': ('q_reject', '1', 'N'),
            ' ': ('q_reject', ' ', 'N')
        }
    },
    initial_state='q_even',
    blank_symbol=' ',
    final_states={'q_accept'}
)

```

Figure 4: Example of a valid automata-lib Turing-machine artifact following the output contract. This toy DTM accepts binary strings of odd positive length. Full model-generated artifacts are released with the benchmark.

A.11 Examples of Terminal Evaluation Outcomes

Table 10 illustrates the four terminal outcomes used by the evaluation pipeline.

A.12 Metric Computation

For each model, the denominator for exact pass is 253 candidates. For the full benchmark, the denominator is 2,024 candidates. The denominator for E2E is fixed at 600 expected cases per candidate, including candidates that fail before behavioral execution. Thus, the full E2E denominator is

$$2,024 \times 600 = 1,214,400. \quad (9)$$

Candidate-level exact pass is binary. A machine either passes all 600 tests or it does not. Case-level scores are used only to analyze partial behavior, failure direction, and semantic mismatch patterns.

We report Wilson 95% confidence intervals for per-model exact-pass rates because each model is

evaluated on 253 independent model-language candidates. Confidence intervals are used for model-level comparison, but we avoid fine-grained ranking when intervals overlap.

A.13 Reproducibility and Release

The benchmark release includes the language specifications, category labels, frozen test suites, oracle labels, prompts, generated model artifacts, extracted candidates, terminal-stage labels, per-case execution records, and scripts for recomputing exact pass, E2E, conditional accuracy, failure-stage rates, semantic error direction, and suite-level reproducibility.

The reported results can therefore be reproduced from the released artifacts without rerunning model generation.

B Statistical Analysis

This section reports statistical analyses for model comparison. A candidate exact-passes if all 600

Table 10: Illustrative terminal outcomes in the evaluation pipeline.

Outcome	Meaning	Example failure mode
STATIC_FAIL	The response violates the output contract before machine construction.	The response contains prose, helper functions, multiple artifacts, or unparseable Python code.
BUILD_FAIL	The artifact parses but does not instantiate a valid TM.	A transition refers to an undeclared state, or final states are inconsistent with the machine definition.
SEMANTIC_MISMATCH	The machine builds and runs but fails at least one oracle test.	For $L = \{0^n 1^n \mid n > 0\}$, the machine accepts 001 or rejects 0011 .
EXACT_PASS	The machine matches all 600 oracle-labeled tests.	The generated TM agrees with the oracle on every frozen test case.

frozen tests are correct. The exact-pass counts, E2E scores, and pairwise comparisons reported below are computed using this scoring rule.

B.1 Statistical Protocol

All model comparisons are performed at the language level. This is important because every model is evaluated on the same 253 languages, and each language contributes 600 test cases. Treating individual test cases as independent would overstate significance because test cases are clustered within languages. For exact pass, each model is represented by a binary vector of length 253, where each entry indicates whether the model exact-passed the corresponding language.

We report Wilson 95% confidence intervals for per-model exact-pass rates. For pairwise model comparisons, we use the exact McNemar test on the paired exact-pass vectors. If model A exact-passes a language and model B does not, the language contributes to A -only; if model B exact-passes and model A does not, it contributes to B -only. Languages solved by both models or by neither model do not contribute to the McNemar statistic. We apply Holm correction over all 28 pairwise model comparisons.

For E2E, we use a language-level nonparametric bootstrap. Each bootstrap replicate samples 253 languages with replacement and recomputes the mean E2E score. This preserves language-level clustering and avoids treating individual strings as independent samples. The resulting intervals quantify uncertainty over the evaluated language set; they should not be interpreted as uncertainty over decoding seeds, prompt variants, or future model versions.

Table 11: Exact-pass rates with Wilson 95% confidence intervals. Each model is evaluated on the same 253 languages.

Model	Exact	EP%	Wilson CI95
gpt-5.2	168	66.4	[60.4, 71.9]
claude-sonnet-4.5	18	7.1	[4.5, 11.0]
qwen3-235B	17	6.7	[4.2, 10.5]
gemini-2.5-pro	9	3.6	[1.9, 6.6]
llama-3.3-70B	7	2.8	[1.3, 5.6]
mistral-large	4	1.6	[0.6, 4.0]
deepseek-v3.2	2	0.8	[0.2, 2.8]
gemma-3-27B	0	0.0	[0.0, 1.5]

B.2 Exact-Pass Confidence Intervals

Table 11 reports exact-pass rates with Wilson 95% confidence intervals. These values are directly aligned with the main leaderboard. Globally, the benchmark contains 2,024 model-language candidates, of which 225 exact-pass, yielding the reported exact-pass rate of 11.12%.

The confidence intervals confirm the main result: gpt-5.2 is separated from all other models by a large margin. The lower-ranked models have substantially overlapping confidence intervals, so we avoid overinterpreting fine-grained rank differences below the leading model.

B.3 Paired Exact-Pass Tests

Table 12 reports paired exact-pass comparisons between gpt-5.2 and every other model. The paired design is important: all models are evaluated on the same language set, so the test compares which model solves each language under the same scoring rule.

The paired tests show that gpt-5.2 significantly outperforms every other model after Holm correc-

Table 12: Paired exact-pass comparisons between gpt-5.2 and each other model. GPT-only is the number of languages exact-passed by gpt-5.2 but not by the comparison model; Other-only is the reverse. p_{Holm} is the Holm-corrected exact McNemar p value over all 28 model pairs.

Comparison model	GPT-only	Other-only	Discordant	Diff. pp	p_{Holm}
claude-sonnet-4.5	156	6	162	59.3	1.79×10^{-37}
qwen3-235B	153	2	155	59.7	1.22×10^{-41}
gemini-2.5-pro	162	3	165	62.8	7.69×10^{-43}
llama-3.3-70B	163	2	165	63.6	1.46×10^{-44}
mistral-large	166	2	168	64.8	1.97×10^{-45}
deepseek-v3.2	166	0	166	65.6	5.77×10^{-49}
gemma-3-27B	168	0	168	66.4	1.50×10^{-49}

tion. The effect sizes are also large: gpt-5.2 solves between 153 and 168 languages that the comparison model does not, whereas the reverse discordant counts range from 0 to 6.

Table 13 reports the full pairwise matrix. These results should be interpreted with the exact-pass metric in mind: exact pass measures complete machine correctness on all 600 tests, not partial behavioral accuracy.

The full matrix supports two conclusions. First, the leading model is statistically separated from the rest under exact pass. Second, the ordering among non-leading models is much less stable. For example, claude-sonnet-4.5 and qwen3-235B differ by only 0.4 percentage points and are not distinguishable under paired exact-pass testing.

B.4 Bootstrap Confidence Intervals for E2E

Exact pass is a candidate-level all-or-nothing metric. To analyze partial behavior under a fixed denominator, we also report language-level bootstrap intervals for E2E. Table 14 reports the mean E2E score and bootstrap 95% confidence interval for each model.

E2E gives a more graded view than exact pass. claude-sonnet-4.5 has low exact pass, but its E2E score is much higher than other non-gpt-5.2 models because it produces runnable machines for most languages. This supports the distinction made in the main paper: exact pass measures complete machine correctness, while E2E measures partial behavioral coverage under a fixed denominator.

Table 15 reports bootstrap intervals for E2E differences between gpt-5.2 and every other model.

All E2E difference intervals are positive, indicating that gpt-5.2 has higher end-to-end case coverage than every other model under the language-

level bootstrap. The smallest E2E gap is against claude-sonnet-4.5, which is consistent with the main result that claude-sonnet-4.5 is highly runnable but frequently semantically incorrect.

B.5 Interpretation

The statistical analysis supports three conclusions. First, the leading model is not merely numerically ahead; it is statistically separated from all other systems under paired exact-pass testing. Second, exact-pass differences among the non-leading models are much less stable, especially between claude-sonnet-4.5 and qwen3-235B, whose paired comparison is not significant. Third, E2E reveals a graded view of partial behavioral competence: claude-sonnet-4.5 has substantially stronger end-to-end case coverage than the other non-gpt-5.2 models despite its low exact-pass rate. Thus, exact pass and E2E answer different questions: exact pass measures complete correctness of the synthesized machine, while E2E measures partial behavioral coverage under a fixed denominator.

C Detailed Results and Diagnostic Tables

The main paper reports only the tables required to establish the central claims: global benchmark outcome, failure-stage decomposition, metric interpretation, model-level performance, category-level difficulty, and semantic error direction. This appendix provides the full diagnostic tables used to support the analysis. The tables are ordered to mirror the main results: model-by-category performance, oracle coverage, artifact-level errors, semantic-level errors, reproducibility checks, and auxiliary audits.

Table 13: Full paired exact-pass comparisons. *A*-only and *B*-only are discordant language counts. p_{Holm} is Holm-corrected over all 28 model pairs.

Model A	Model B	Diff. pp	A-only	B-only	p	p_{Holm}
gpt-5.2	gemma-3-27B	66.4	168	0	5.35×10^{-51}	1.50×10^{-49}
gpt-5.2	deepseek-v3.2	65.6	166	0	2.14×10^{-50}	5.77×10^{-49}
gpt-5.2	mistral-large	64.8	166	2	7.59×10^{-47}	1.97×10^{-45}
gpt-5.2	llama-3.3-70B	63.6	163	2	5.86×10^{-46}	1.46×10^{-44}
gpt-5.2	gemini-2.5-pro	62.8	162	3	3.20×10^{-44}	7.69×10^{-43}
gpt-5.2	qwen3-235B	59.7	153	2	5.29×10^{-43}	1.22×10^{-41}
gpt-5.2	claude-sonnet-4.5	59.3	156	6	8.13×10^{-39}	1.79×10^{-37}
claude-sonnet-4.5	gemma-3-27B	7.1	18	0	7.63×10^{-6}	1.60×10^{-4}
qwen3-235B	gemma-3-27B	6.7	17	0	1.53×10^{-5}	3.05×10^{-4}
qwen3-235B	deepseek-v3.2	5.9	15	0	6.10×10^{-5}	1.16×10^{-3}
claude-sonnet-4.5	deepseek-v3.2	6.3	17	1	1.45×10^{-4}	2.61×10^{-3}
qwen3-235B	mistral-large	5.1	15	2	2.35×10^{-3}	3.99×10^{-2}
claude-sonnet-4.5	mistral-large	5.5	17	3	2.58×10^{-3}	4.12×10^{-2}
gemini-2.5-pro	gemma-3-27B	3.6	9	0	3.91×10^{-3}	5.86×10^{-2}
llama-3.3-70B	gemma-3-27B	2.8	7	0	1.56×10^{-2}	2.19×10^{-1}
claude-sonnet-4.5	llama-3.3-70B	4.3	16	5	2.66×10^{-2}	3.46×10^{-1}
gemini-2.5-pro	deepseek-v3.2	2.8	8	1	3.91×10^{-2}	4.69×10^{-1}
claude-sonnet-4.5	gemini-2.5-pro	3.6	13	4	4.90×10^{-2}	5.39×10^{-1}
qwen3-235B	llama-3.3-70B	4.0	17	7	6.39×10^{-2}	6.39×10^{-1}
qwen3-235B	gemini-2.5-pro	3.2	14	6	1.15×10^{-1}	1.00
mistral-large	gemma-3-27B	1.6	4	0	1.25×10^{-1}	1.00
llama-3.3-70B	deepseek-v3.2	2.0	7	2	1.80×10^{-1}	1.00
gemini-2.5-pro	mistral-large	2.0	9	4	2.67×10^{-1}	1.00
deepseek-v3.2	gemma-3-27B	0.8	2	0	5.00×10^{-1}	1.00
llama-3.3-70B	mistral-large	1.2	7	4	5.49×10^{-1}	1.00
mistral-large	deepseek-v3.2	0.8	4	2	6.88×10^{-1}	1.00
gemini-2.5-pro	llama-3.3-70B	0.8	8	6	7.91×10^{-1}	1.00
claude-sonnet-4.5	qwen3-235B	0.4	14	13	1.00	1.00

Table 14: Language-level bootstrap confidence intervals for E2E. Each bootstrap replicate samples 253 languages with replacement and recomputes model E2E.

Model	E2E%	Bootstrap CI95
gpt-5.2	71.8	[66.4, 76.9]
claude-sonnet-4.5	57.1	[54.2, 60.0]
qwen3-235B	42.2	[38.1, 46.3]
mistral-large	37.0	[33.6, 40.5]
llama-3.3-70B	24.7	[20.9, 28.5]
deepseek-v3.2	9.5	[6.8, 12.4]
gemini-2.5-pro	4.7	[2.4, 7.3]
gemma-3-27B	0.9	[0.2, 2.0]

Table 15: Language-level bootstrap confidence intervals for E2E differences between gpt-5.2 and each other model.

Comparison model	E2E diff. pp	Bootstrap CI95
claude-sonnet-4.5	14.7	[8.9, 20.4]
qwen3-235B	29.6	[23.0, 35.6]
mistral-large	34.8	[28.6, 40.6]
llama-3.3-70B	47.1	[40.3, 53.9]
deepseek-v3.2	62.3	[56.4, 68.1]
gemini-2.5-pro	67.1	[61.1, 72.8]
gemma-3-27B	70.9	[65.4, 76.1]

C.1 Model-by-Category Performance

Table 16 expands the category-level analysis from the main paper by reporting exact pass for each model–category pair. This table verifies that the

aggregate category trends are not driven by a single model. In particular, the weakness on CFL-2 and CFL-5 persists across models: even gpt-5.2 falls below 40% on both categories, and no other model exceeds 10% on either.

Table 16: Exact-pass rate by model and category. gemma-3-27B is omitted because it exact-passes no language in any category.

Model	CFL-1	CFL-2	CFL-3	CFL-4	CFL-5	NCFL-1	NCFL-4
gpt-5.2	96.3	37.5	89.2	78.9	38.5	76.5	60.5
claude-sonnet-4.5	11.1	2.5	2.7	13.2	0.0	11.8	10.5
qwen3-235B	7.4	10.0	5.4	13.2	5.1	2.9	2.6
gemini-2.5-pro	7.4	0.0	0.0	13.2	2.6	2.9	0.0
llama-3.3-70B	7.4	0.0	0.0	2.6	2.6	2.9	5.3
mistral-large	0.0	7.5	0.0	0.0	0.0	2.9	0.0
deepseek-v3.2	3.7	0.0	0.0	2.6	0.0	0.0	0.0

Table 17: Oracle coverage. Most languages are solved by zero or one model; no language is solved by more than five models.

Number of exact-passing models	Languages	Share
0	75	29.64%
1	144	56.92%
2	24	9.49%
3	8	3.16%
4	1	0.40%
5	1	0.40%
6+	0	0.00%
Solved by at least one model	178	70.36%
Unsolved by every model	75	29.64%

C.2 Oracle Coverage and Unique Solves

Table 17 reports how many models exact-pass each language. This measures benchmark saturation independently of any single model.

Of the 178 languages solved by at least one model, gpt-5.2 solves 168 and is the unique solver for 137. claude-sonnet-4.5 uniquely solves 3, llama-3.3-70B 2, and gemini-2.5-pro and mistral-large one each. qwen3-235B, deepseek-v3.2, and gemma-3-27B contribute no unique solves. Thus, a best-per-language oracle improves over gpt-5.2 alone by only 10 languages.

C.3 Static Failure Decomposition

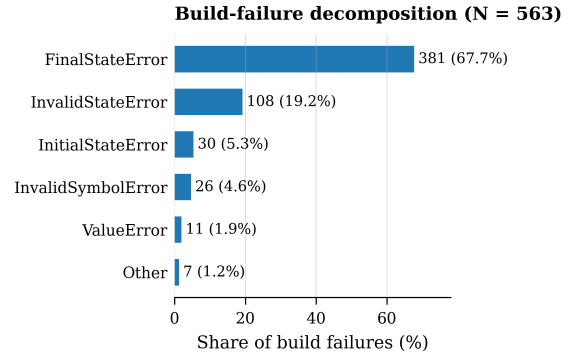
Table 18 decomposes static failures. The main paper reports only the dominant pattern: 363/522 static failures are unterminated string literals. The full table is included here to distinguish syntax-level failures from other static-validator failures.

C.4 Build Failure Decomposition

Figure 5 summarizes the build-failure decomposition. Unlike static failures, these candidates parse successfully but do not define valid Turing machines. The dominant error is FinalStateError,

Table 18: Static failure decomposition. Unterminated string literals are the dominant static error.

Status	Error group	Candidates
SYNTAX_ERROR	unterminated string literal	363
SYNTAX_ERROR	other static error	131
STATIC_FAIL	other static error	28

**Figure 5:** Build-failure decomposition over 563 build failures. The distribution is highly concentrated, with FinalStateError accounting for most invalid Turing-machine constructions.

which indicates inconsistent or invalid accepting-state declarations.

C.5 Semantic Error Diagnostics

The main paper reports the first-failure direction among semantic mismatches. Here we provide the full case-level confusion matrix and the near-miss distribution.

C.6 Within-Test-Set Reproducibility

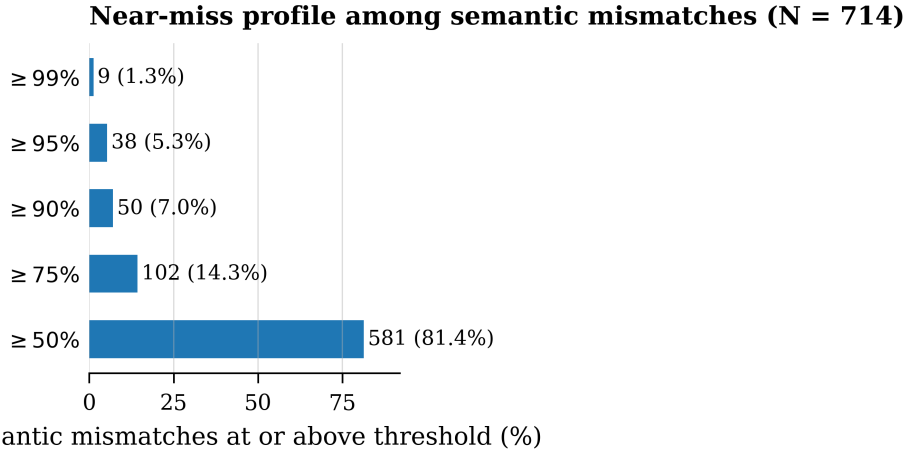
Table 20 reports suite-level stability. The two suites are drawn from the same generator and should therefore be read as a within-test-set reproducibility check, not as robustness to new prompts, new decoding seeds, or adversarial test generation.

C.7 Branch-Level Aggregate

Figure 7 reports the branch-level CFL vs. Non-CFL aggregate. This table supports the main-text

Table 19: Per-model case-level confusion. R = recall on accepted strings; Spec = specificity; BA = balanced accuracy.

Model	TP	FN	FP	TN	R%	Spec%	BA%
gpt-5.2	52,655	4,045	360	56,340	92.9	99.4	96.1
gemini-2.5-pro	3,301	899	404	3,796	78.6	90.4	84.5
qwen3-235B	15,170	36,430	2,691	48,909	29.4	94.8	62.1
claude-sonnet-4.5	19,929	51,471	4,675	66,725	27.9	93.5	60.7
llama-3.3-70B	10,587	20,913	4,586	26,914	33.6	85.4	59.5
gemma-3-27B	947	253	723	477	78.9	39.8	59.3
deepseek-v3.2	5,916	6,384	3,777	8,523	48.1	69.3	58.7
mistral-large	10,924	41,876	7,486	45,314	20.7	85.8	53.3

**Figure 6:** Near-miss profile among the 714 semantic mismatches. Bars show the fraction of executable-but-wrong candidates whose case accuracy meets or exceeds each threshold.**Table 20:** Suite-1 vs. Suite-2 reproducibility. $|\Delta|$ is absolute difference in percentage points.

Model	Suite 1	Suite 2	$ \Delta $
gpt-5.2	96.11	96.12	0.002
claude-sonnet-4.5	60.68	60.68	0.003
qwen3-235B	62.12	62.06	0.056
gemini-2.5-pro	84.50	84.48	0.024
llama-3.3-70B	59.33	59.72	0.384
mistral-large	53.13	53.38	0.242
deepseek-v3.2	58.83	58.56	0.268
gemma-3-27B	57.92	60.75	2.833
Aggregate	66.77	66.85	0.079

claim that branch-level exact-pass rates are nearly identical, but the richer category-level analysis in Table 6 is the primary evidence because it reveals much larger within-branch variation.

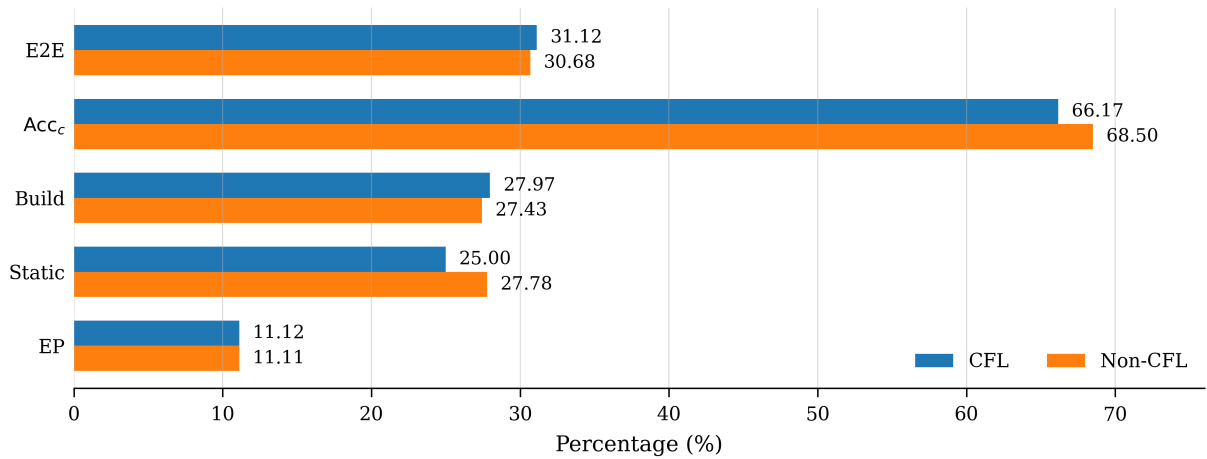
C.8 Length-Bucket Audit

Table 21 reports accuracy by input-length bucket. This table is included only as an audit. It should not be interpreted as a length-scaling law because the

mixture of language categories and models differs across length buckets.

D Inference Configuration

All model generations are performed under a zero-shot inference setting. For each benchmark instance, the prompt provides the language identifier, alphabet, and formal language definition. The model is required to output only executable Python code defining a single top-level Turing-machine object named `tm`, instantiated with `automata-lib` as either a DTM or NTM. We use temperature 0.2 and a maximum generation budget of 8000 tokens. Each instance is queried once per run, with retries used only for empty or failed responses.



CFL: 181 languages (N=1,448) | Non-CFL: 72 languages (N=576)

Figure 7: Branch-level comparison between CFL and Non-CFL languages. Bars report exact-pass rate (EP), static-failure rate, build-failure rate, conditional case accuracy (Acc_c), and end-to-end case score (E2E). The close agreement in exact pass and E2E indicates that the CFL/Non-CFL distinction is not the dominant empirical difficulty axis. Counts for the number of languages and total model–language candidates are shown as figure annotations.

Table 21: Per-case accuracy by input-length bucket over runnable cases. This is an audit, not evidence of a length-scaling law.

Input length	Cases	Accuracy
0	384	85.7%
1	838	82.1%
2	1,855	84.4%
3–5	34,685	87.9%
6–10	64,351	77.2%
11–20	116,739	68.9%
21–50	155,444	62.6%
51–100	53,671	57.5%
101–250	37,619	57.1%
251–500	27,777	58.7%
501–1,000	19,815	65.6%
1k–5k	41,783	65.7%
5k–10k	712	67.0%
10k–50k	2,827	83.2%
50k–100k	1,262	97.6%
100k–500k	469	78.9%
500k–1M	293	78.8%

Table 22: Inference configuration used for Turing-machine generation.

Setting	Value
Prompting strategy	Zero-shot
Input per instance	Language ID, alphabet, and definition
Output format	Python automata-lib code
Generated object	One top-level tm object
Machine type	DTM or NTM
Temperature	0.2
Maximum tokens	8000
Generations per instance	1
Retry handling	Retry on empty or failed responses