

Can LLMs Design Computational Machines? Pushdown Automaton Synthesis as a Test of Structured Computational Reasoning

Syed Mumtahir Mahmud^{*†}, Nazira Jesmin Lina^{*}, Shahriyar Zaman Ridoy,
S. M. Muhtasimul Hasan, Md Mosaddek Khan

Cognitive Agents and Interaction Lab, Department of Computer Science and Engineering,
University of Dhaka

syedmumtahir-2019617833@cs.du.ac.bd , nazirajesmin-2019117838@cs.du.ac.bd , shahriyar.zaman01@gmail.com
muhtasimhasan904@gmail.com , mosaddek@du.ac.bd

Abstract

Artificial intelligence has long been evaluated by its ability to perform computation and structured reasoning. While recent large language models (LLMs) demonstrate strong capabilities in language generation, code synthesis, and structured problem solving, it remains unclear whether they can construct formal computational mechanisms from high-level specifications. In this work, we investigate whether modern LLMs can synthesize pushdown automata (PDAs) from language descriptions expressed in natural language or mathematical form. Because PDA construction requires explicit reasoning about states, transitions, and stack-based memory, it provides a controlled test of structured symbolic reasoning. We introduce the task of pushdown automaton synthesis and develop an automated evaluation framework that verifies generated PDAs through structural buildability checks and execution-based validation. Using a benchmark of context-free languages with labeled test cases, we evaluate contemporary LLMs and analyze their ability to construct and execute formal computational machines. Our results show that current state-of-the-art LLMs struggle to reliably construct executable PDAs, indicating that synthesizing computational mechanisms requiring coordinated reasoning over state transitions and stack-based memory manipulation remains challenging for present-day models. All code and data are available at: <https://github.com/Dip-to/Pushdown-Automaton-Synthesis-Dataset.git>

1 Introduction

Artificial intelligence has long sought reliable ways to determine whether machines can perform genuine computation and reasoning rather than merely

imitate intelligent behavior. Turing’s seminal work introduced the Turing machine as a formal model capable of performing any computable procedure, establishing a mathematical foundation for understanding the limits of algorithmic computation (Turing, 1936). Later, Turing proposed the Turing Test as a behavioral evaluation framework, suggesting that a machine could be considered intelligent if its responses in natural language conversation are indistinguishable from those of a human (Turing, 1950). Together, these ideas highlight two complementary perspectives on machine intelligence: the ability to construct computational mechanisms capable of solving problems and the ability to demonstrate such intelligence through language-based interaction.

Subsequent developments in formal language theory further refined the relationship between computational models and the classes of languages they can process. The Chomsky hierarchy classifies computational models into four levels of increasing expressive power—finite automata, pushdown automata, linear bounded automata, and Turing machines—ranging from simple pattern recognition to fully general computation (Chomsky, 1956; Hopcroft et al., 2001; Sipser, 1996; Delétang et al., 2023). Within this hierarchy, finite automata represent the simplest machines capable of recognizing regular patterns, while Turing machines represent the most powerful model capable of solving any computable problem. This hierarchy therefore provides a principled framework for understanding the computational capabilities required to process increasingly complex language structures.

Recent advances in artificial intelligence, particularly through large language models (LLMs), have significantly expanded the ability of machines to perform complex reasoning and language-driven tasks (Zhang et al., 2026; Lamia et al., 2025; Sheilat et al., 2026). Built upon Transformer architectures (Vaswani et al., 2017) and trained on massive

^{*} Equal Contribution

[†] Corresponding Author

corpora with large-scale computational resources, modern LLMs can generate coherent text, write executable code, and solve a wide variety of structured problems (Brown et al., 2020; OpenAI et al., 2023; Grattafiori et al., 2024; Chen et al., 2025; Wei et al., 2022). However, constructing formal computational machines—such as pushdown automata or Turing machines—from high-level descriptions requires explicit algorithmic design, including the ability to define states, transitions, and memory mechanisms that correctly realize the intended computation. Prior work has examined the formal-language processing abilities of neural sequence models and Transformers, showing that these models can learn certain patterns associated with formal languages under controlled settings (Bhattamishra et al., 2020; Merrill et al., 2022; Strobl et al., 2024; Butoi et al., 2024). Yet these studies primarily focus on language recognition or sequence prediction rather than the explicit construction of computational mechanisms. Related work has also explored explicit grammatical representations in LLM-based generation. Wang et al. (Wang et al., 2023) use LLMs to predict specialized BNF grammars as intermediate representations for domain-specific generation, while Albinhassan et al. (Albinhassan et al., 2025) learn context-sensitive constraints from LLM interactions and enforce them during generation. However, these approaches focus on grammar-guided or constrained generation rather than the direct synthesis of executable computational machines. Consequently, whether modern LLMs can synthesize formal machines from natural language or mathematical specifications remains largely unexplored, offering a principled test of structured symbolic reasoning beyond surface-level pattern generation.

Pushdown automata (PDAs) extend finite automata by introducing an unbounded stack, enabling the recognition of context-free languages. This stack-based memory allows PDAs to capture nested and recursive structures that frequently arise in computing practice, including arithmetic expressions, balanced delimiters, and block-structured program syntax. Context-free languages also admit dual characterizations: they can be described generatively through context-free grammars or recognized computationally by pushdown automata (Hopcroft et al., 2001; Sipser, 1996). Because PDAs require explicit manipulation of stack memory in addition to state transitions, constructing them from language descriptions demands struc-

tured reasoning about both control flow and memory behavior. Importantly, PDAs occupy a principled middle ground in the Chomsky hierarchy between memoryless finite automata and fully general Turing machines, making them a suitable level of abstraction for evaluating structured computational reasoning. Motivated by this perspective, we investigate whether modern AI models can synthesize correct pushdown automata from language specifications, using PDA construction as a controlled and interpretable test of state-dependent and memory-based reasoning. The main contributions of this work are as follows:

- We formulate the task of pushdown automaton synthesis from language specifications, where a model must construct a PDA from either natural language descriptions or formal mathematical expressions of a language. To the best of our knowledge, this is the first systematic study evaluating whether large language models can synthesize executable pushdown automata from such specifications.
- We introduce an automated evaluation framework that verifies generated PDAs through structural buildability checks and execution-based validation on labeled test cases, thereby distinguishing failures of automaton construction from failures of semantic correctness.
- We construct a benchmark comprising context-free languages with 50 labeled test cases per language, covering diverse stack-based phenomena, including counting dependencies, nesting, and multi-block constraints.
- We conduct a systematic empirical evaluation of several state-of-the-art LLMs, providing quantitative comparisons and detailed analysis of structural and semantic failure modes in pushdown automaton synthesis.

2 Background

This section summarizes the theoretical foundations underlying our evaluation framework, situating pushdown automata (PDAs) within the classical theory of computation and the formal language hierarchy. Formal language theory provides a principled framework for understanding the computational mechanisms required to recognize different classes of languages and therefore offers a natural

basis for evaluating whether machines can construct executable computational models.

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 (RLs) represent the simplest class and are recognized by finite automata (FAs), which operate with finitely many states and no auxiliary memory. While efficient, FAs cannot enforce unbounded counting or nested dependencies across arbitrarily long inputs.

Context-free languages (CFLs) extend RLs by allowing recursive and hierarchical structures such as balanced delimiters, recursive patterns, and matched counts. This additional expressive power arises from augmenting a finite automaton with stack memory, yielding a *pushdown automaton* (PDA), which recognizes exactly the class of CFLs (Hopcroft et al., 2001; Sipser, 1996). More expressive language classes include context-sensitive languages and recursively enumerable languages, the latter recognized by Turing machines with unrestricted read-write memory that characterize general algorithmic computation.

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 $L \subseteq \Sigma^*$. Context-free languages admit both generative and computational characterizations. A *context-free grammar* (CFG) is a tuple

$$G = (V, \Sigma, R, S),$$

where V is a finite set of nonterminals, Σ is a finite set of terminals ($V \cap \Sigma = \emptyset$), R is a finite set of productions $A \rightarrow \beta$ with $A \in V$ and $\beta \in (V \cup \Sigma)^*$, and S is the start symbol. The language generated by G is

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

A language is context-free if and only if it is generated by some CFG, equivalently, if and only if it is recognized by some PDA (Hopcroft et al., 2001; Sipser, 1996). This equivalence provides two complementary perspectives on CFLs: a generative description through grammatical productions and a computational realization through stack-based state transitions.

A *pushdown automaton* (PDA) is a finite-state machine augmented with a stack. Formally, a PDA

is a tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

where Q is a finite set of states, Σ is the input alphabet, Γ is the stack alphabet, $q_0 \in Q$ is the start state, $Z_0 \in \Gamma$ is the initial stack symbol, $F \subseteq Q$ is the set of accepting states, and

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$$

is the transition relation. If $(p, \alpha) \in \delta(q, a, X)$, then in state q , reading $a \in \Sigma \cup \epsilon$ with top-of-stack X , the machine may move to p and replace X by $\alpha \in \Gamma^*$.

The stack provides unbounded last-in–first-out memory. This enables PDAs to track long-range dependencies by pushing symbols during one phase of computation and later matching or removing them. Figure 1 illustrates a PDA recognizing

$$L = \{a^{2m}c^{4n}d^nb^m \mid m, n \geq 0\},$$

where the stack enforces separated counting constraints across multiple symbol blocks. Such dependencies are beyond finite automata but do not require the full computational power of Turing machines.

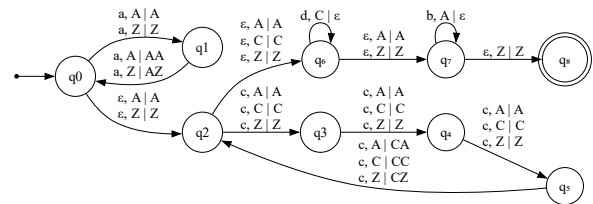


Figure 1: Pushdown automaton (PDA) recognizing $L = \{a^{2m}c^{4n}d^nb^m \mid m, n \geq 0\}$, using the stack to enforce the dependencies between the numbers of a 's and b 's and between c 's and d 's, with q_8 as the accepting state.

Because PDAs lie between finite automata and Turing machines in expressive power, they provide a controlled setting for assessing structured reasoning. Unlike regular-language recognition, PDA construction requires explicit memory manipulation; unlike unrestricted Turing computation, it remains formally constrained and mechanically verifiable. These properties make PDA synthesis a suitable substrate for evaluating whether AI systems can construct executable computational mechanisms from language specifications.

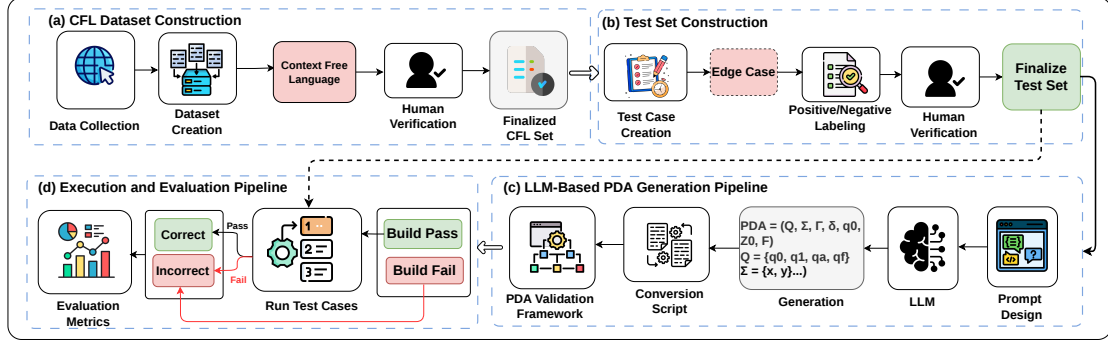


Figure 2: End-to-end workflow of the evaluation framework. The process consists of four stages: (a) CFL dataset construction with human verification, (b) test-set generation with edge cases and labeled positive/negative examples, (c) LLM-based PDA generation and conversion into an executable automaton, and (d) execution and evaluation. Generated PDAs are first validated for successful construction (BUILDPASS or BUILDFAIL); buildable automata are then executed on the test suite and labeled CORRECT or INCORRECT based on test-case outcomes.

3 Methodology

This section describes the methodological framework used to evaluate large language models (LLMs) on the task of synthesizing pushdown automata (PDAs) from language specifications expressed either in natural language or in formal mathematical form. We first formalize the PDA synthesis task and then describe the benchmark construction, prompt design, and automated evaluation pipeline used in our experiments.

3.1 Task Definition

We study the problem of synthesizing a pushdown automaton from a language specification expressed in textual or mathematical form. Given a language description L , the model is required to generate a PDA M that recognizes the language defined by the specification. The task therefore requires translating a high-level language description into an explicit computational mechanism that implements the corresponding recognition procedure.

Formally, the task is defined as a mapping

$$f_{\theta} : \text{CFLSpec}(L) \rightarrow \text{PDA}(M),$$

where $\text{CFLSpec}(L)$ denotes a standardized textual or mathematical specification of a context-free language and $\text{PDA}(M)$ denotes a formally defined pushdown automaton.

The generated PDA must follow the standard 7-tuple representation

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

explicitly defining its states Q , input alphabet Σ , stack alphabet Γ , transition relation δ (including

optional ϵ -moves), initial state q_0 , initial stack symbol Z_0 , and accepting states F . The output must therefore constitute a complete and executable formal artifact rather than an informal construction or proof sketch. This task specifically evaluates a model’s ability to generate executable control-flow logic (state transitions) and stack-manipulation logic (push/pop operations) directly, requiring joint reasoning about when to transition states and how to manipulate memory. Figure 2 illustrates the end-to-end workflow from language specification to executable PDA verification.

3.2 Benchmark Construction

We construct a benchmark consisting of 41 context-free languages. The languages are collected from standard textbooks, instructional materials, and publicly available resources on formal language theory (Hopcroft et al., 2001; Sipser, 1996). In several cases, the original formulations are lightly modified or rephrased to diversify surface descriptions while preserving their formal properties.

Each candidate language is examined to ensure that it is context-free and that a recognizing PDA can be constructed. Languages that are ambiguous in definition or not context-free are excluded. The resulting benchmark covers a range of stack-based phenomena commonly studied in formal language theory, including counting dependencies, multi-block arithmetic constraints, palindromic structures, inequality relations, and balanced or nested delimiters. While many specifications are expressed in mathematical form (e.g., set-builder notation), a subset is written as textual descriptions to reflect realistic language specification styles.

For each language, we construct a verification suite consisting of 50 labeled strings (accepted or rejected). Candidate strings are initially generated using a deterministic script with fixed length bounds and random seeds to ensure reproducibility. The script samples strings within predefined limits and filters them according to the formal constraints of the language. Automatically generated strings are then supplemented with manually designed boundary examples, including minimal valid strings, edge-count configurations, and near-miss negative cases.

Finally, all test cases associated with each language are manually reviewed to verify that their acceptance labels are correct and consistent with the intended language definition. Only test cases that pass this human verification step are retained in the final benchmark.

3.3 Prompt Design and Output Constraints

To isolate reasoning performance from formatting variability, we design a fixed instruction template that specifies the exact schema required for PDA construction. The template instructs the model to produce a PDA using the standard 7-tuple representation and requires explicit declaration of each component, including the state set, input alphabet, stack alphabet, transition relation, start state, initial stack symbol, and accepting states.

For each evaluation instance, the instruction template is concatenated with a language specification expressed in textual or mathematical form. All models are evaluated under identical zero-shot conditions using the same prompt to ensure comparability across systems (Kojima et al., 2022). The template also prohibits explanatory text or commentary outside the required schema, ensuring that the output corresponds to a formal machine specification rather than a descriptive explanation. The complete instruction template used in the experiments is provided in Appendix C.

3.4 Automated Evaluation Pipeline

To evaluate the generated automata, we employ an automated verification pipeline that checks both structural correctness and behavioral validity (Cobbe et al., 2021). The pipeline first performs a structural validation step that parses the model output and verifies whether it conforms to the required PDA schema. Outputs that fail to define a valid 7-tuple or contain inconsistent states, alphabets, or transition definitions are categorized as structural

failures and are excluded from execution.

If the generated PDA passes structural validation, it is instantiated within a PDA simulator and executed on the verification suite associated with the corresponding language. Each automaton is evaluated on the 50 labeled strings constructed during benchmark generation. For every test string, the simulator determines whether the PDA accepts or rejects the input according to the defined transition relation and stack operations.

All executable automata are instantiated using the `automata-lib` Python library,¹ which provides implementations of deterministic and nondeterministic pushdown automata (DPDA and NPDA). Generated PDA specifications are first parsed by a custom conversion script that maps the model output into the library’s PDA object representation. The simulator supports standard PDA execution semantics, including nondeterministic transitions and ϵ -moves. Acceptance is evaluated using the library’s criteria, which include acceptance by final state, acceptance by empty stack, or both conditions simultaneously, depending on the PDA specification. The execution engine handles stack manipulation, transition expansion, and nondeterministic branching internally according to the library’s implementation.

The predicted results are then compared with the ground-truth labels to determine correctness. This execution-based evaluation allows us to distinguish between failures caused by malformed machine constructions and those resulting from incorrect computational behavior. By separating structural and semantic errors, the pipeline provides a precise assessment of whether a model successfully constructs an executable automaton that implements the intended language constraints.

To account for the stochastic nature of LLM generation, each model is queried independently three times for every language specification, producing three generations without conversational carryover. The generated PDAs are evaluated separately and aggregated during analysis to measure per-generation accuracy, pooled accuracy across generations, retry-based reliability, and structural robustness through build-failure rates. All models are evaluated under identical prompting conditions using the same instruction template, without fine-tuning or external tool augmentation, ensuring a controlled and reproducible evaluation setup.

¹<https://pypi.org/project/automata-lib/>

4 Empirical Evaluation

We empirically evaluate the ability of large language models (LLMs) to synthesize executable pushdown automata (PDAs) from language specifications expressed either as natural language descriptions or formal mathematical expressions. The evaluation examines whether generated PDAs are both structurally well-formed and semantically correct with respect to the intended context-free language. Using the benchmark introduced in Section 3.2, model outputs are verified through an automated framework that combines structural buildability checks with execution-based validation on labeled test strings. Our experimental design focuses on evaluating structured symbolic synthesis under strict formal constraints while enabling controlled comparison across models.

4.1 Models

We evaluate six contemporary LLMs representing different positions in the current model landscape: three proprietary frontier systems (Gemini 3 Pro, GPT-5.2, Claude Sonnet 4.5) and three large open-weight models (Qwen3-235B-A22B, Deepseek v3.2, Mistral Large 3).

Gemini 3 Pro, GPT-5.2, and Claude Sonnet 4.5 represent the highest-capability publicly available variants of their respective model families at the time of evaluation and are therefore used to estimate the upper limits of proprietary LLM performance on PDA synthesis. Among open-weight systems, Qwen3-235B-A22B provides a high-capacity alternative to closed models, Deepseek v3.2 enables examination of reasoning-oriented training for automata-theoretic construction, and Mistral Large 3 serves as a general-purpose baseline for structured generation robustness. This selection allows us to examine whether PDA synthesis capability correlates with reasoning specialization, model scale, or training ecosystem.

All models are evaluated under identical zero-shot conditions using the same fixed PDA instruction template to ensure controlled comparison. To ensure a controlled and consistent comparison across models, we used identical inference configurations for all evaluated systems. Specifically, the temperature was fixed at **0.0**, corresponding to greedy decoding, while top-*p* was set to **1.0**, effectively disabling nucleus-based truncation. The maximum generation length was set to **4096 tokens** for all models, which was sufficient for produc-

ing complete PDA specifications; no generation reached this limit in practice. All models were evaluated using their **default non-thinking configuration**, with no extended reasoning or additional reasoning-budget features enabled.

4.2 Evaluation Metrics

PDA construction is a formally constrained task requiring both syntactic validity and semantic correctness. We therefore evaluate generated automata using a two-stage verification process: *build* and *test execution*.

Stage 1: Build We first parse the model output according to the required schema and attempt to instantiate an executable PDA within our evaluation framework. If any issue occurs—including schema violations, missing or invalid symbols, malformed transitions, library instantiation failures, or runtime exceptions—the output is labeled **BUILDFAIL**; otherwise, it is labeled **BUILDPASS**.

Stage 2: Test execution For each **BUILDPASS** PDA, we execute the automaton on the 50 labeled test cases associated with the target language. A PDA is labeled **CORRECT** if it matches all test-case labels (0 mismatches), and **INCORRECT** if at least one test case is mismatched (≥ 1 mismatch). Outputs labeled **BUILDFAIL** are also counted as **INCORRECT**. Importantly, passing the curated test suite demonstrates behavioral correctness on the evaluated benchmark cases, rather than formal language equivalence between the synthesized PDA and the target language.

4.3 Experimental Results

This section presents the empirical evaluation of PDA synthesis from CFL specifications across six state-of-the-art LLMs. We analyze overall accuracy, failure modes, robustness across independent attempts, and the statistical reliability of observed performance differences.

Comparative Accuracy Across Models. Table 1 and Figures 3 and 4 summarize how reliably each model generates correct PDAs on the same set of 41 CFL instances when sampled three times independently (Turns 1–3). Table 1 reports, for each turn, the numbers of correct and incorrect generations together with accuracy, and also provides a pooled “Total” over all three turns. Figure 3 visualizes the per-turn and pooled accuracies

Model	Turn 1			Turn 2			Turn 3			Total		
	Corr	Incorr	Acc.%	Corr	Incorr	Acc.%	Corr	Incorr	Acc.%	Corr	Incorr	Acc.%
Gemini 3 Pro	31	10	75.61	30	11	73.17	32	9	78.05	93	30	75.61
GPT-5.2	23	18	56.10	26	15	63.41	25	16	60.98	74	49	60.16
Qwen3-235B-A22B	22	19	53.66	19	22	46.34	22	19	53.66	63	60	51.22
Deepseek v3.2	14	27	34.15	14	27	34.15	13	28	31.71	41	82	33.33
Claude Sonnet 4.5	12	29	29.27	12	29	29.27	13	28	31.71	37	86	30.08
Mistral Large 3	5	36	12.20	6	35	14.63	5	36	12.20	16	107	13.01

Table 1: Accuracy Statistics on 41 CFLs. Each model is queried three times independently (Turns 1–3). Corr and Incorr are the numbers of correct and incorrect PDAs, respectively; Acc.% is the corresponding accuracy.

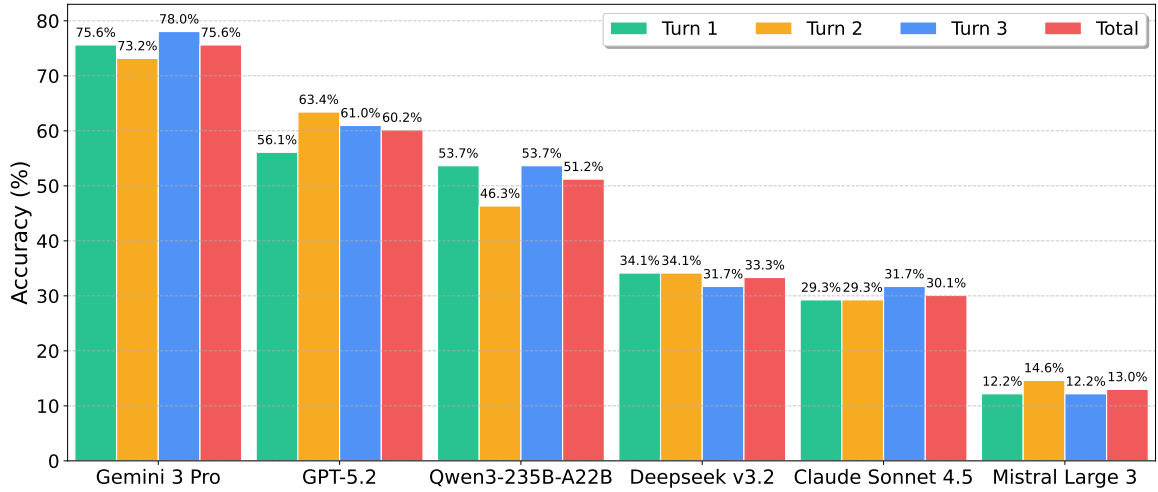


Figure 3: Accuracy across independent generations on the 41 CFL instances. For each model, Turn 1–3 bars report per-turn accuracy, and the *Total* bar reports pooled accuracy across all three turns.

Model	Incorrect	BuildFail	Built-but-wrong	BuildFail (%)
Gemini 3 Pro	30	2	28	6.67
GPT-5.2	49	9	40	18.37
Qwen3-235B-A22B	60	17	43	28.33
Deepseek v3.2	82	20	62	24.39
Claude Sonnet 4.5	86	39	47	45.35
Mistral Large 3	107	54	53	50.47

Table 2: Distribution of errors across models, showing how incorrect cases are split between build failures and built-but-wrong outputs, along with the corresponding build-failure rate.

for cross-model comparison. Gemini 3 Pro consistently achieves the highest performance across turns (about 73–78% per turn; 75.6% pooled), followed by GPT-5.2 (about 56–63% per turn; 60.2% pooled) and Qwen3-235B-A22B (about 46–54% per turn; 51.2% pooled). Deepseek v3.2, Claude Sonnet 4.5, and Mistral Large 3 perform substantially worse, with pooled accuracies of 33.3%, 30.1%, and 13.0%, respectively.

Figure 4 further shows pooled correct and incorrect counts (out of 123 total trials per model). Gemini 3 Pro and GPT-5.2 accumulate far more

Model	$\Pr[\geq 1]$	$\Pr[\geq 2]$	$\Pr[3/3]$	$\mathbb{E}[\text{Att.} \text{Succ.}]$	$\Pr[\text{Fail}]$
Gemini 3 Pro	78.0	75.6	73.2	1.06	22.0
GPT-5.2	70.7	61.0	48.8	1.24	29.3
Qwen3-235B-A22B	65.9	53.7	34.1	1.22	34.1
Deepseek v3.2	58.5	26.8	14.6	1.67	41.5
Claude Sonnet 4.5	36.6	29.3	24.4	1.33	63.4
Mistral Large 3	22.0	9.8	7.3	1.56	78.0

Table 3: Per-instance retry performance over three independent generations. We report best-of-3 success ($\Pr[\geq 1]$), cross-retry consistency ($\Pr[\geq 2]$ and $\Pr[3/3]$), expected attempt of first success ($\mathbb{E}[\text{attempt}|\text{success}]$), and complete failure rate ($\Pr[\text{Fail}]$).

correct trials and fewer failures, whereas Deepseek v3.2, Claude Sonnet 4.5, and Mistral Large 3 are dominated by incorrect generations. These results reveal a clear performance gap: while Gemini 3 Pro and GPT-5.2 can often construct correct PDAs, reliability remains limited across the contemporary systems.

Failure Mode Breakdown. Table 2 decomposes incorrect trials into two categories: *BuildFail* and *Built-but-wrong*. We quantify the contribution of

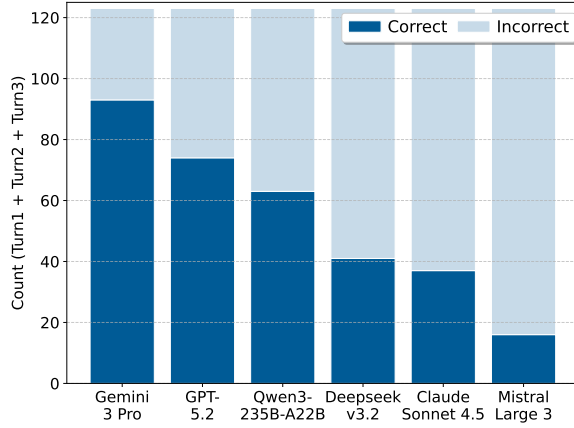


Figure 4: Visualization of correct vs. incorrect generations aggregated over three independent turns (123 total trials per model).

construction errors using the *BuildFail rate*, defined as $\text{BuildFail}/\text{Incorrect} \times 100$, which measures the proportion of errors caused by executability failures. A clear pattern emerges across models. Gemini 3 Pro exhibits relatively low BuildFail rates and primarily fails at the semantic level, producing syntactically valid PDAs that incorrectly capture the language constraints. In contrast, Claude Sonnet 4.5 and Mistral Large 3 show substantially higher BuildFail rates, indicating that many outputs cannot be parsed or executed. These results highlight that successful PDA synthesis requires both structural correctness and semantic correctness: models must generate automata that are syntactically valid and functionally accurate. Separating construction failures from functional errors, therefore, provides important diagnostic insight into model behavior.

Error Analysis To understand where LLMs struggle in PDA synthesis, we examine three factors: task difficulty, model behavior, and the format used to specify the target language. Detailed results are provided in Appendix D; here, we summarize the main findings.

As CFL difficulty increases, PDA synthesis becomes less reliable. The overall error rate rises from 33.3% on Easy languages to 58.5% on Medium and 67.0% on Hard languages. Logical errors account for most failures, meaning that models often produce a valid PDA whose transition or stack behavior does not correctly implement the target language. Construction failures are less common but also increase with difficulty, from 6.8% on Easy tasks to 16.0% on Medium and 30.0% on Hard tasks. Thus, increasing language complexity

makes it harder not only to construct a valid PDA, but especially to implement its intended computation correctly.

Models also show different failure patterns. Gemini 3 Pro primarily makes logical errors and has relatively few construction failures, whereas Claude Sonnet 4.5 and Mistral Large 3 experience substantially more construction failures. Other models, including GPT-5.2, Qwen3-235B-A22B, and Deepseek v3.2, show increasing logical errors as difficulty grows. These differences indicate that models can struggle at different stages of synthesis, from producing an executable automaton to implementing the correct computational behavior.

A particularly important finding is that *how the CFL is specified also affects synthesis performance*. Figure 5 compares mathematical notation with natural-language descriptions. Mathematical specifications achieve higher accuracy for most models and have a higher overall median accuracy. For example, Gemini 3 Pro achieves 79.4% accuracy with mathematical notation compared with 71.7% with textual descriptions, while Claude Sonnet 4.5 improves from 25.0% to 34.9%. This trend is not universal, as GPT-5.2 and Mistral Large 3 perform better with textual specifications. Nevertheless, the overall result suggests that mathematical notation provides a more precise representation of the language constraints, making them easier to translate into PDA transitions. In contrast, natural-language descriptions require additional interpretation before the model can construct the corresponding computation. This highlights that PDA synthesis depends not only on model capability, but also on how the target language is specified.

Robustness Across Independent Generations.

Table 3 analyzes performance under three independent attempts per CFL instance. For each model, we report the probability that an instance is solved at least once (best-of-3), in at least two turns, or in all three turns, capturing both peak success and cross-turn consistency. We also report $\mathbb{E}[\text{attempt} \mid \text{success}]$, the expected attempt index of first success (lower values indicate faster success), and $\text{Pr}[\text{no success}]$, the probability that all three attempts fail.

For example, Gemini 3 Pro shows high best-of-3 accuracy (78.0%) and strong consistency (73.2% solved in all three turns), whereas Deepseek v3.2 shows a large gap between occasional success (58.5%) and full consistency (14.6%). Overall,

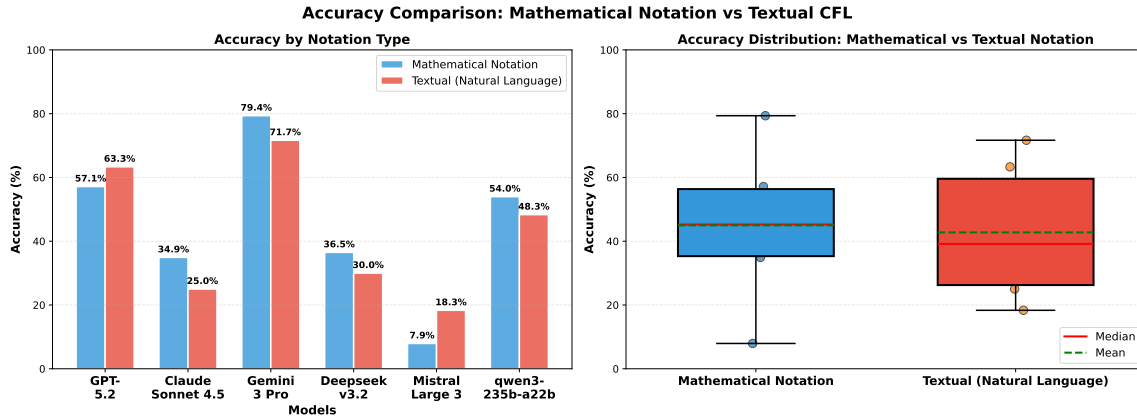


Figure 5: Comparison of model accuracy when CFLs are specified using mathematical notation versus natural-language descriptions. Mathematical specifications generally yield higher accuracy across models. The boxplot summarizes the overall distribution, showing a higher median accuracy for mathematical representations than for textual descriptions.

retries increase the probability of success, but reliability varies substantially across models. Performance remains relatively stable across attempts, suggesting that most errors reflect model capability limitations rather than stochastic decoding. Even with retries, Claude Sonnet 4.5 and Mistral Large 3 rarely achieve consistent success.

Paired Significance Tests. Each model produces a binary outcome (Correct/Incorrect) on the same 123 evaluation items. Because results are paired on identical CFL instances, we use McNemar’s exact test to assess whether performance differences between models are statistically significant. Complete methodological details are provided in Appendix E.

The results establish a clear performance hierarchy. Gemini 3 Pro significantly outperforms all other models. GPT-5.2 forms the next tier, outperforming lower-ranked models but showing no significant difference from Qwen3-235B-A22B. Qwen3-235B-A22B occupies the middle tier, while Deepseek v3.2 and Claude Sonnet 4.5 are statistically similar but below the top systems. Mistral Large 3 performs the weakest overall.

Multiple-comparison correction. Without correction, the probability of false positives increases with multiple testing. We therefore apply the Benjamini–Hochberg procedure (Benjamini and Hochberg, 1995) to control the false discovery rate at $q = 0.05$. Complete methodological details are provided in Appendix F. After correction, the ranking remains unchanged: Gemini 3 Pro significantly outperforms all others; GPT-5.2 forms the second

tier, outperforming lower-ranked models but not differing significantly from Qwen3-235B-A22B; and Mistral Large 3 remains the weakest. The only statistically indistinguishable pairs under BH are GPT-5.2 vs. Qwen3-235B-A22B and Claude Sonnet 4.5 vs. Deepseek v3.2.

5 Conclusions

In this work, we conducted a systematic evaluation of LLMs on the task of synthesizing PDAs from CFL specifications. We introduced a benchmark of diverse CFL instances together with an automated evaluation framework that separates structural buildability from functional correctness, enabling precise diagnosis of construction and reasoning errors.

Our empirical results reveal substantial variation across models. While Gemini 3 Pro demonstrates comparatively strong performance in generating executable PDA structures, several other SOTA systems frequently fail either during automaton construction or during semantic verification. Statistical analysis using McNemar’s exact test with Benjamini–Hochberg correction confirms that many of these performance differences are significant.

Overall, the results suggest that although modern LLMs show progress in structured generation tasks, reliably synthesizing computational machines that require coordinated state transitions and stack-based memory operations remains challenging. Beyond the specific task studied here, PDA synthesis provides a principled and interpretable testbed for evaluating structured reasoning capabilities in LLMs.

Limitations

Despite providing a controlled evaluation of pushdown automaton synthesis, our study has several limitations. First, the evaluation uses 41 context-free languages (CFLs), each accompanied by 50 labeled test cases, yielding 2,050 total evaluation strings. The dataset was constructed with careful verification to ensure formal correctness and diversity. Each candidate language was collected from multiple educational and theoretical sources in formal language theory and manually verified to confirm that it is indeed context-free. Furthermore, all generated test strings were reviewed by human annotators to ensure that the Accept/Reject labels correctly reflect the target language definition. The dataset emphasizes correctness and theoretical validity rather than raw volume, which is particularly important for evaluating formal computational artifacts.

Second, our model evaluation covers only a subset of contemporary LLMs available at the time of the study. We evaluate six representative systems, including both proprietary frontier models and large open-weight models, in order to capture different points in the current model ecosystem. However, the LLM landscape evolves rapidly, with frequent updates to architectures, training data, and reasoning capabilities. As a result, the relative performance of models on PDA synthesis may change as newer generations of models are released. In addition, many proprietary systems do not disclose detailed architectural or training information, limiting deeper analysis of which model properties contribute to successful automaton synthesis. Our results should therefore be interpreted as a snapshot of model capabilities at the time of evaluation rather than a definitive ranking of all possible LLM systems.

Third, our evaluation requires models to synthesize PDAs directly. An alternative pipeline would have models first generate a context-free grammar (CFG) from the specification and then mechanically convert the CFG into a PDA using standard algorithmic transformations. Because CFG-to-PDA conversion is a well-defined procedure, this two-stage approach could separate language understanding from direct automaton construction and may yield higher success rates. We view this CFG-first synthesis strategy as an important baseline and a promising direction for future work.

Acknowledgments

The authors acknowledge the support of the Improving Computer and Software Engineering Tertiary Education Project (ICSETEP), Bangladesh, under Sub-project PIN A-77. We also acknowledge the Cognitive Agents and Interaction Lab (CAIL), University of Dhaka, for providing the necessary research facilities and computational resources.

References

- Hervé Abdi. 2007. The bonferonni and Šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics*, 3.
- Mohammad Albinhassan, Pranava Madhyastha, Mark Law, and Alessandra Russo. 2025. [Learning and enforcing context-sensitive control for LLMs](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 834–842, Vienna, Austria. Association for Computational Linguistics.
- Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300.
- 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 B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. 2024. Training neural networks as recognizers of formal languages. *arXiv preprint arXiv:2411.07107*.
- Junyi Chen, Shihao Bai, Zaijun Wang, Siyu Wu, Chuheng Du, Hailong Yang, Ruihao Gong, Shengzhong Liu, Fan Wu, and Guihai Chen. 2025. [Pre3: Enabling deterministic pushdown automata for faster structured LLM generation](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11253–11267. Association for Computational Linguistics.

- 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. *arXiv preprint arXiv:2110.14168*.
- 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](#). *arXiv preprint arXiv:2207.02098*.
- 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](#). *arXiv preprint arXiv:2407.21783*.
- John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*, 2 edition. Addison-Wesley.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213.
- Lubna Zahan Lamia, Mabsur Fatin Bin Hossain, and Md Mosaddek Khan. 2025. [Who holds the pen? caricature and perspective in LLM retellings of history](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 19691–19710, Suzhou, China. Association for Computational Linguistics.
- Quinn McNemar. 1947. [Note on the sampling error of the difference between correlated proportions or percentages](#). *Psychometrika*, 12(2):153–157.
- William Merrill, Ashish Sabharwal, and Noah A. Smith. 2022. [Saturated transformers are constant-depth threshold circuits](#). *Transactions of the Association for Computational Linguistics*, 10:843–856.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Shlok Shelat, Jay Raval, Souvik Roy, and Manas Gaur. 2026. [Beyond memorization: Testing LLM reasoning on unseen theory of computation tasks](#). In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 29904–29934, San Diego, California, United States. 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.
- Alan M. Turing. 1936. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265.
- Alan M. Turing. 1950. Computing machinery and intelligence. *Mind*, 59(236):433–460.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.
- Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. [Grammar prompting for domain-specific language generation with large language models](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 65030–65055. Curran Associates, Inc.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Xinglang Zhang, Yunyao Zhang, ZeLiang Chen, Junqing Yu, Wei Yang, and Zikai Song. 2026. [Logical phase transitions: Understanding collapse in LLM logical reasoning](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18836–18860, San Diego, California, United States. Association for Computational Linguistics.

A Performance Summaries for each LLM

This section summarizes per-model performance patterns across the three-turn setting (Figure 6 - 11). In this part, Turn 1, Turn 2, and Turn 3 are treated as independent trials: each turn is scored on its own, with no conversational carryover or shared context between turns.

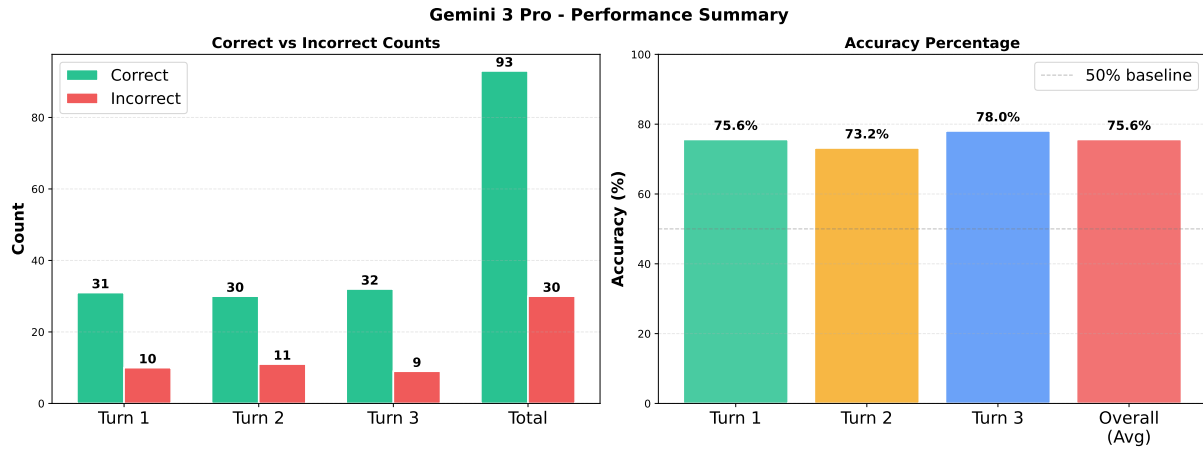


Figure 6: Gemini 3 Pro performance on the 41 CFL instances over three independent generations (Turns 1–3). The plot reports correct vs. incorrect counts per turn and the pooled total, together with per-turn accuracies and the pooled accuracy across all turns.

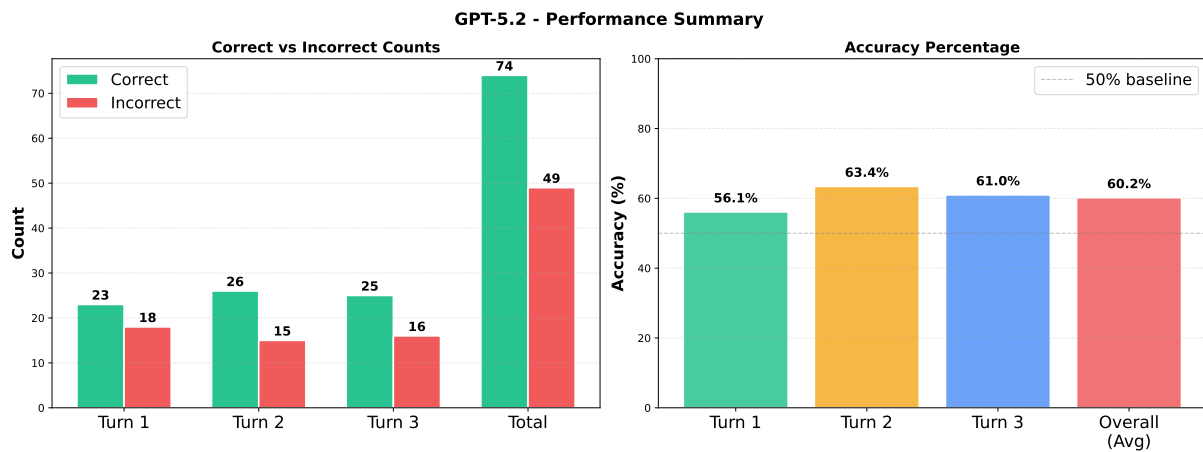


Figure 7: GPT-5.2 performance on the 41 CFL instances over three independent generations (Turns 1–3). The plot shows correct vs. incorrect counts per turn and in total, along with per-turn accuracies and the pooled accuracy across all turns.

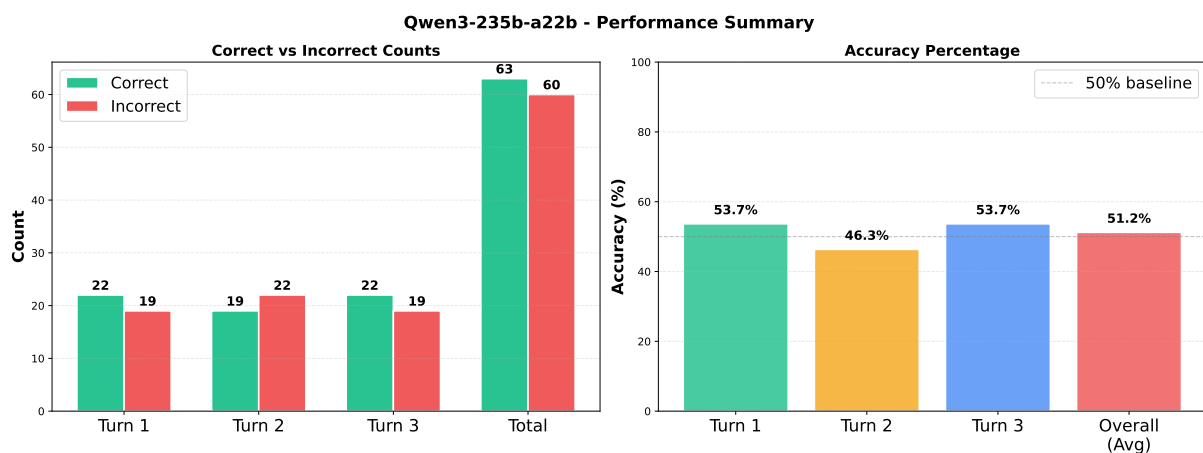


Figure 8: Qwen3-235B-A22B performance on the 41 CFL instances over three independent generations (Turns 1–3). The plot reports turn-wise correct/incorrect counts and the pooled total, and summarizes accuracy per turn and overall.

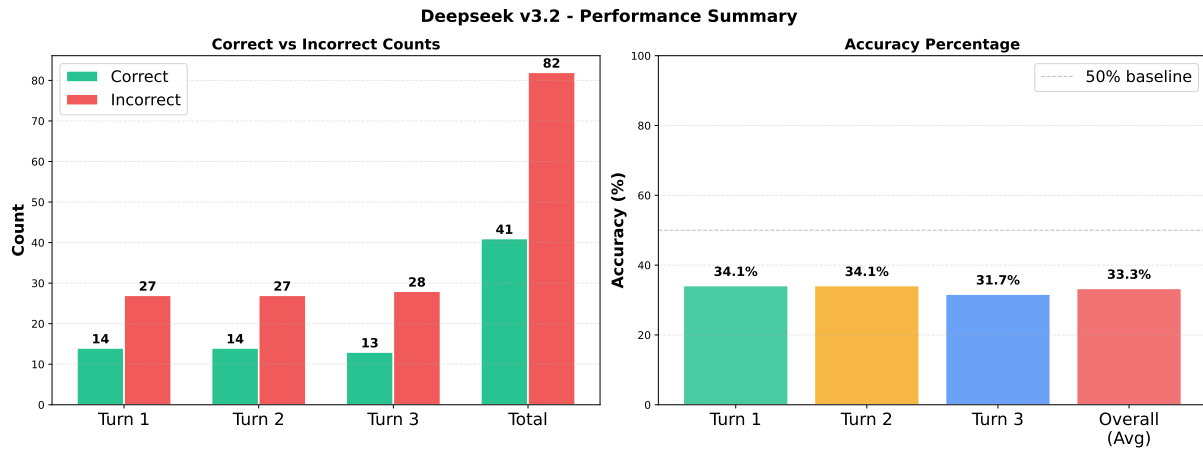


Figure 9: Deepseek v3.2 performance on the 41 CFL instances over three independent generations (Turns 1–3). The plot reports correct/incorrect counts per turn and pooled, and the corresponding per-turn and overall accuracies.

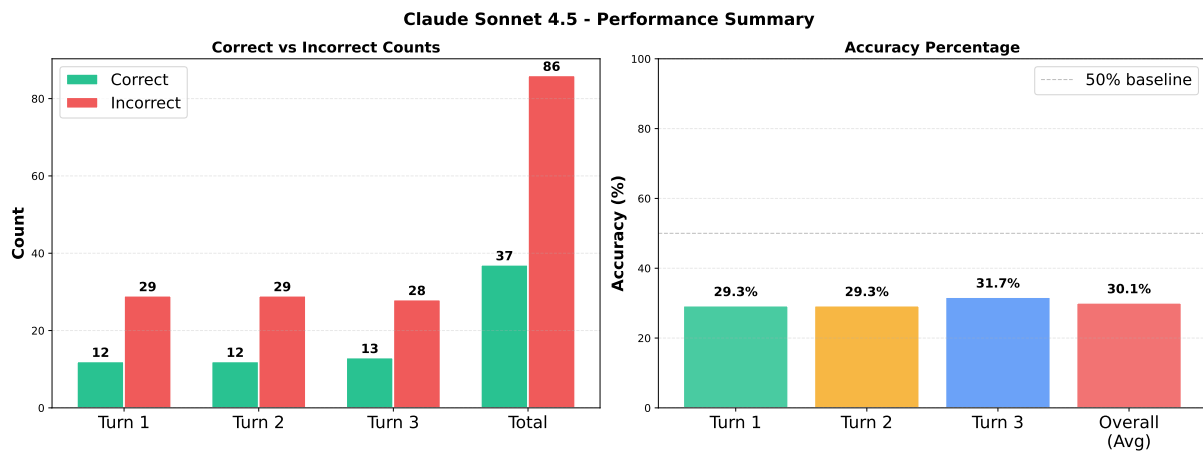


Figure 10: Claude Sonnet 4.5 performance on the 41 CFL instances over three independent generations (Turns 1–3). The plot shows turn-wise correct vs. incorrect counts and the pooled total, together with per-turn and overall accuracies.

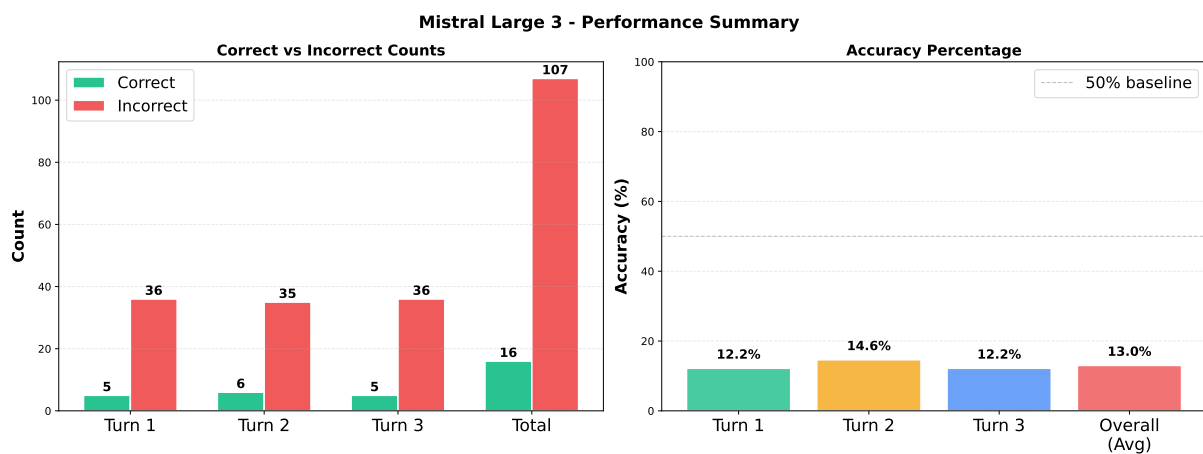


Figure 11: Mistral Large 3 performance on the 41 CFL instances over three independent generations (Turns 1–3). The plot summarizes correct/incorrect counts per turn and pooled, and reports per-turn accuracies with the pooled accuracy.

B CFLs Used in the Evaluation

This appendix presents a representative set of Context-Free Languages (CFLs) used in our evaluation. The selected languages cover a variety of structural patterns commonly studied in formal language theory.

B.1 CFLs Specified in Mathematical Notation

Example	CFL
1	$L = \{a^{2n}b^{3n} \mid n \geq 0\}$
2	$L = \{0^n1^m0^n \mid n, m \geq 0\}$
3	$L = \{a^mb^{(2m+1)} \mid m \geq 1\}$
4	$L = \{wxw^R \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
5	$L = \{a^nb^mc^{n+m} \mid n \geq 1, m \geq 1\}$
6	$L = \{a^ib^jc^k \mid i, j, k \geq 0, i + j = k\}$
7	$L = \{a^{2m}c^{4n}d^mb^m \mid m, n \geq 0\}$
8	$L = \{0^n1^m \mid n \geq 1, m \geq 1, m > n + 2\}$

Table 4: Sample of mathematically specified Context-Free Languages (CFLs) used in this study.

B.2 CFLs Specified Using Textual Description

Example	CFL
9	$L = \{ \text{all strings that have some number of } a\text{'s, followed by some number of } b\text{'s, followed by the same number of } c\text{'s as } b\text{'s, followed by the same number of } d\text{'s as } a\text{'s (with at least one } a \text{ and at least one } b) \}$
10	$L = \{ \text{all strings that look like: some number of } a\text{'s, then some number of } b\text{'s, then the same number of } a\text{'s as the previous } b\text{'s, then the same number of } b\text{'s as the first } a\text{'s} \}$
11	$L = \{ \text{all strings made only of } () [] \text{ that are correctly balanced and correctly typed, meaning every opening symbol is eventually closed by the matching kind, and nesting is always consistent} \}$
12	$L = \{ \text{strings that look like: some } a\text{'s followed by some } b\text{'s, where } \#a \geq \#b \}$
13	$L = \{ \text{strings that look like: some } 1\text{'s, then some } 0\text{'s, then } 1\text{'s again (allowing either block to be empty), where the last } \#1 = (\text{first } \#1) + 2 \cdot (\#0) \}$
14	$L = \{ \text{strings where every prefix has at least as many } a\text{'s as } b\text{'s} \}$

Table 5: Sample textually described Context-Free Languages (CFLs) used in the experiments.

C PDA Construction Prompt

This Section presents the prompt template used to evaluate model performance on PDA construction. For each instance in our dataset, the placeholder for the language definition is replaced with a specific CFL.

```
# Given a language (L), you are tasked with
constructing a corresponding Pushdown Automaton
(PDA) and presenting it in formal definition
form.
```

```
## PDA Formal Definition
A PDA is formally defined as a 7-tuple:
P = (Q, Σ, Γ, δ, q_0, Z_0, F)
```

```
Where,
Q: set of states
Σ: input alphabet
Γ: stack alphabet
q_0: initial state
Z_0: initial stack symbol
F: set of accepting states
```

```
Rules:
- Use 'ε' for the empty string.
- Output only the PDA definition.
- Transitions must follow the format:
δ(q, x, γ) = (p, α). where q,p ∈ Q, x ∈ Σ ∪ {ε},
γ ∈ Γ, α ∈ Γ*.
```

```
Language L = { <instance-specific language> }
```

D Error Analysis

To better understand the failure modes of large language models in PDA synthesis, we analyze error patterns across difficulty levels, models, and language specification formats. In particular, we distinguish between two primary failure types: *construction failures* (BUILDFAIL), where the generated PDA cannot be parsed or instantiated, and *logical errors*, where a syntactically valid automaton fails to correctly recognize the target language during execution.

To examine how task complexity influences model behavior, we categorize the 41 context-free languages (CFLs) in our benchmark into three difficulty levels based on human judgment. Specifically, a group of participants with background knowledge in formal language theory were asked to assess each CFL and categorize it according to the perceived difficulty of constructing a corresponding pushdown automaton. Based on this evaluation, the dataset was partitioned into Easy (n=9), Medium (n=17), and Hard (n=15) categories. In the following analysis, we report model performance and error patterns with respect to these difficulty levels.

Figures 12-5 provide complementary views of these behaviors.

D.1 Error Distribution Across Difficulty Levels

Figure 13 illustrates how error rates evolve as the complexity of the CFL specification increases. A consistent trend is observed: the total error rate

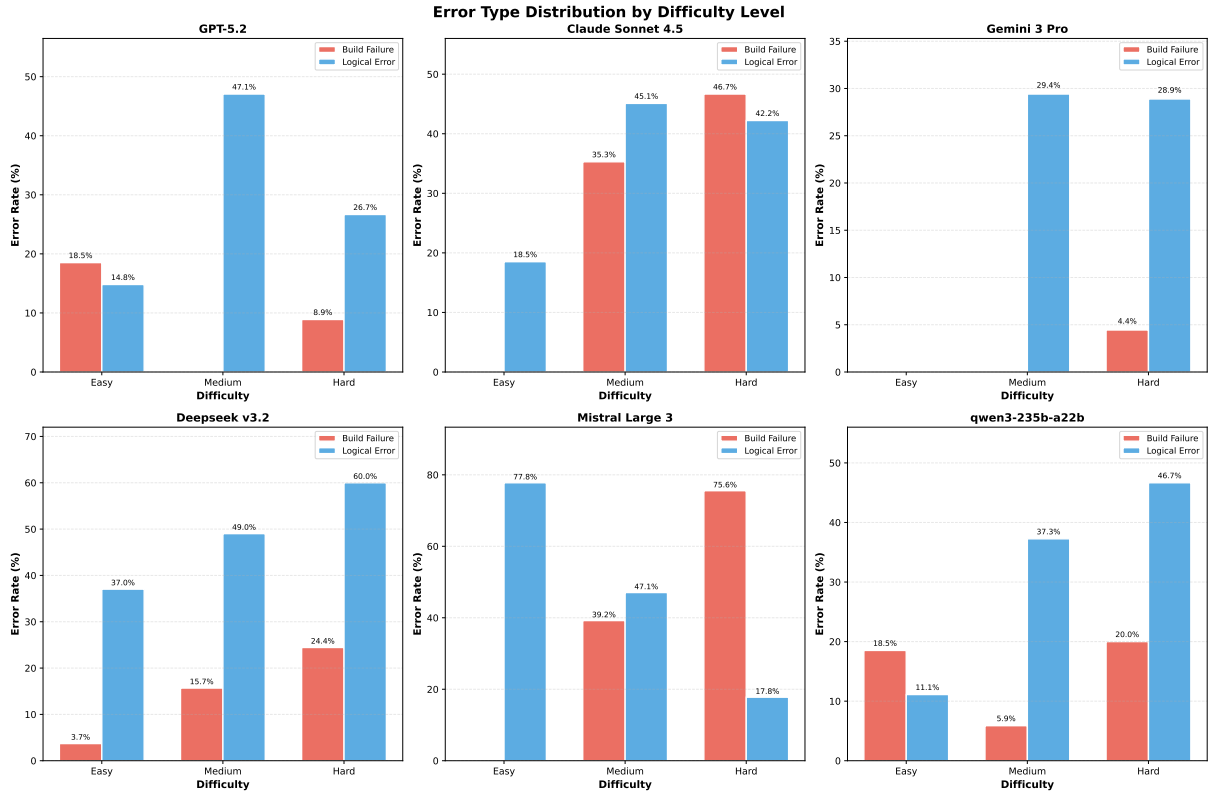


Figure 12: Per-model distribution of error types across CFL difficulty levels. Errors are divided into BUILDFAIL, where the generated PDA cannot be parsed or instantiated, and LOGICAL ERROR, where a syntactically valid PDA fails during execution. Across most models and difficulty levels, logical errors dominate, while construction failures increase as the language gets harder. The results highlight that many models generate structured outputs but struggle to implement correct transition logic.

increases from approximately 33.3% for Easy languages to 58.5% for Medium languages and 67.0% for Hard languages. Across all difficulty categories, logical errors constitute the dominant failure mode, indicating that models frequently generate syntactically valid PDA descriptions but fail to encode the correct transition logic required to recognize the language.

Construction failures are less frequent but increase noticeably as task difficulty grows. The BUILDFAIL rate rises from 6.8% for Easy tasks to 16.0% for Medium tasks and 30.0% for Hard tasks. This pattern suggests that generating well-formed automaton specifications becomes increasingly challenging as the structural complexity of the language description increases. Overall, these results indicate that while models often succeed in producing syntactically structured outputs, implementing correct stack-based behavior remains a significant challenge.

D.2 Model-Specific Error Patterns

Figure 12 provides a more detailed view of how different models contribute to the overall error distribution. Distinct error profiles emerge across systems.

GPT-5.2 exhibits moderate construction failure rates and a noticeable increase in logical errors as task difficulty increases, particularly for Medium and Hard languages. Claude Sonnet 4.5 shows a substantial increase in construction failures at higher difficulty levels, indicating that the generated PDA specifications often cannot be parsed or executed. Gemini 3 Pro produces relatively few construction failures, with most errors arising from incorrect logical transitions rather than formatting issues.

Deepseek v3.2 demonstrates a steady increase in logical errors as difficulty grows, suggesting difficulty in encoding the correct stack manipulation logic required for more complex CFLs. Mistral Large 3 exhibits high error rates across all categories, with a combination of construction failures and logical errors. Qwen3-235B-A22B shows com-

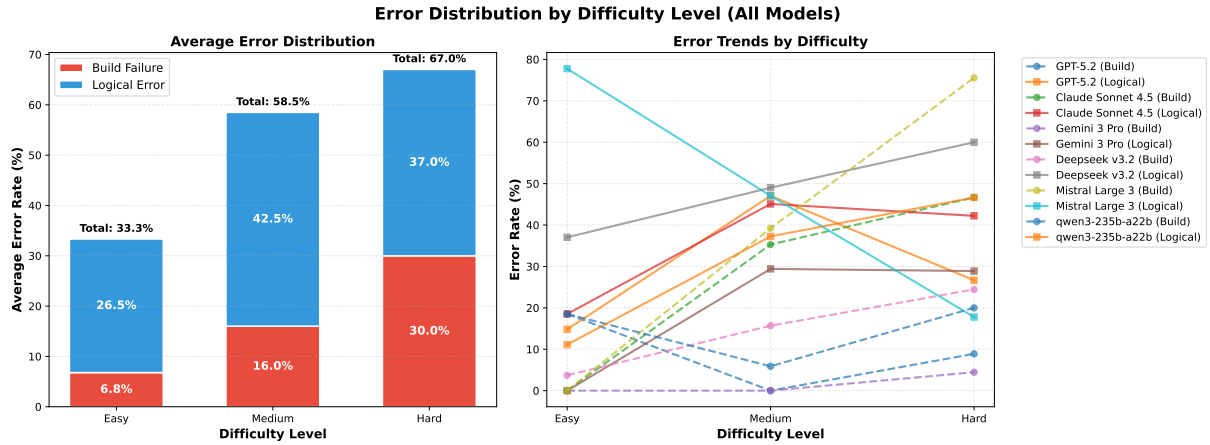


Figure 13: Aggregate error distribution across CFL difficulty levels over all models. Errors are decomposed into construction failures (BUILDFAIL) and logical execution errors. Total error rates increase with language complexity, rising from Easy to Hard tasks. Logical errors remain the dominant failure mode across all difficulty categories.

paratively lower construction failures but increasing logical errors for Medium and Hard tasks.

These model-specific patterns highlight that different systems struggle with distinct aspects of PDA synthesis, ranging from structural generation issues to incorrect computational logic.

D.3 Performance Across Difficulty Categories

Figure 14 presents model accuracy across the three difficulty categories. Performance varies substantially across models and difficulty levels. Gemini 3 Pro achieves the highest accuracy across all categories, reaching 100% accuracy on Easy tasks and maintaining strong performance on Medium (70.6%) and Hard (66.7%) languages. GPT-5.2 shows comparatively stable performance across categories, achieving accuracies of 66.7%, 52.9%, and 64.4% for Easy, Medium, and Hard tasks respectively.

Several models exhibit substantial performance degradation as difficulty increases. Claude Sonnet 4.5 achieves 81.5% accuracy on Easy tasks but drops to 19.6% on Medium tasks and 11.1% on Hard tasks. Deepseek v3.2 decreases from 59.3% on Easy languages to 35.3% and 15.6% on Medium and Hard languages respectively. Mistral Large 3 achieves relatively low accuracy across all categories (22.2%, 13.7%, and 6.7%). Qwen3-235B-A22B performs well on Easy and Medium languages (70.4% and 56.9%) but shows reduced performance on Hard tasks (33.3%).

These results demonstrate that the complexity of CFL specifications strongly influences the ability of language models to synthesize correct pushdown

automata.

D.4 Impact of Language Specification Format

Figure 5 compares model performance when context-free languages are expressed using formal mathematical notation vs. natural-language descriptions. Across most models, mathematical specifications result in higher accuracy than textual descriptions. For example, GPT-5.2 achieves 57.1% accuracy under mathematical notation compared to 63.3% under textual specifications, while Gemini 3 Pro achieves 79.4% and 71.7%, respectively.

Several models exhibit larger performance gaps between the two representations. Claude Sonnet 4.5 achieves 34.9% accuracy under mathematical notation but only 25.0% under textual descriptions. Similarly, Deepseek v3.2 achieves 36.5% accuracy with mathematical notation and 30.0% with textual specifications. These differences suggest that the format used to describe a language can significantly influence model performance.

In addition to the model-wise comparison, the boxplot in Figure 5 summarizes the overall distribution of accuracies for the two specification formats. The distribution shows that mathematical-based specifications consistently achieve higher median accuracy than textual descriptions. Moreover, the mathematical notation condition exhibits a wider spread of results, reflecting strong performance from several models (e.g., Gemini 3 Pro and GPT-5.2) while weaker models still struggle. In contrast, the textual specification condition shows a lower median and a slightly tighter distribution, indicat-

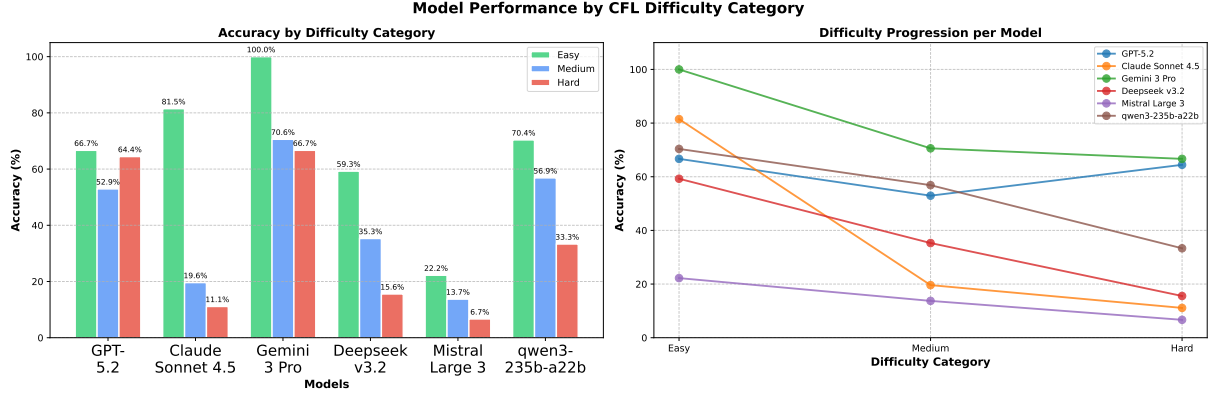


Figure 14: Model accuracy across CFL difficulty categories (Easy, Medium, and Hard). The left panel shows per-model accuracy by difficulty, while the right panel illustrates performance trends as task complexity increases. Most models experience accuracy degradation as language complexity grows, with Gemini 3 Pro maintaining the strongest performance across categories.

ing that natural-language descriptions introduce additional ambiguity that affects most models consistently. This distributional view further reinforces the observation that structured, rule-based representations generally provide a clearer signal for PDA synthesis than natural-language descriptions.

Overall, the results indicate that structured mathematical representations often align more closely with the internal patterns learned by language models, whereas natural-language descriptions may introduce ambiguity, complicating PDA synthesis.

Key Findings. The error analysis reveals several consistent patterns across models and difficulty levels. First, error rates increase substantially as the complexity of the target CFL grows, indicating that synthesizing correct pushdown automata becomes increasingly challenging as language constraints require deeper stack manipulation and more structured reasoning. Second, logical errors dominate across most settings, suggesting that models often generate syntactically valid PDA specifications but fail to implement the correct transition logic needed to recognize the language. Third, model behavior varies significantly: Gemini 3 Pro and GPT-5.2 maintain relatively stable performance across difficulty levels, while models such as Claude Sonnet 4.5 and Deepseek v3.2 exhibit sharp performance degradation as complexity increases. Finally, the format used to specify the language also influences performance, with mathematical notation generally producing higher accuracy than natural-language descriptions. Together, these findings highlight that PDA synthesis remains a challenging task for current large language models and

provide a useful diagnostic setting for evaluating structured computational reasoning.

E McNemar’s Exact Test (Paired Binary Outcomes) for Model Comparison

E.1 Overview

To assess whether two models differ systematically in accuracy on the same evaluation items, we use McNemar’s test (McNemar, 1947). This test is appropriate for paired, binary outcomes: for each item, each model is marked as Correct (1) or Incorrect (0). Because the models are evaluated on the same prompts, the outcomes are naturally paired; McNemar’s test explicitly accounts for this pairing.

Paired outcome table. For each paired item, the joint outcomes of Model A and Model B form a 2×2 contingency table:

	B Incorrect	B Correct
A Incorrect	n_{00}	n_{01}
A Correct	n_{10}	n_{11}

where n_{11} denotes items both models answered correctly, n_{00} denotes items both answered incorrectly, n_{10} denotes items where A is correct and B is incorrect (A “wins”), and n_{01} denotes items where A is incorrect and B is correct (B “wins”). McNemar’s test relies only on the discordant counts (n_{10}, n_{01}), since agreements (n_{00}, n_{11}) do not indicate which model is better.

Assumption Checks We compare Model A as baseline with each other model B using McNemar’s *exact* test because the evaluation setup satisfies the key assumptions:

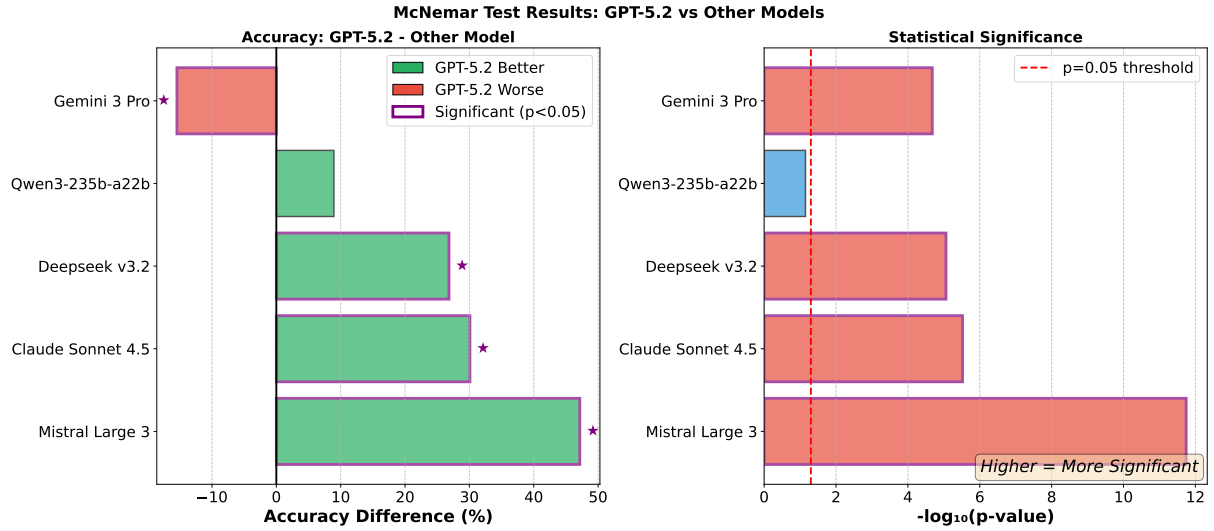


Figure 15: McNemar exact-test results for GPT-5.2 versus each comparator model on the same paired trials. Left: accuracy difference $\Delta \text{Acc.}\%$ (GPT-5.2 minus comparator); positive values indicate GPT-5.2 is more accurate. Right: statistical evidence shown as $-\log_{10}(p)$, with the dashed line marking the $p = 0.05$ significance threshold; higher values indicate stronger evidence of a difference. Significant comparisons (exact McNemar, $\alpha = 0.05$) are highlighted.

- **Paired observations:** The same set of $N = 123$ evaluation items is answered by both model A and the comparison model B. Therefore, each item yields a paired outcome (one outcome per model on the same item).
- **Binary outcomes:** For every item and every model, the result is coded as Correct (1) or Incorrect (0), which is exactly the outcome type required by McNemar's test.
- **Independence across items:** Different evaluation items are treated as independent trials (the correctness on one item does not determine correctness on another).
- **Exact inference:** McNemar's exact test uses the binomial distribution over discordant pairs and does not rely on large-sample approximations. This is especially suitable when the number of disagreements is small or imbalanced.

Hypothesis Formulation We test whether the two models have equal accuracy on paired items. Equivalently, under the null hypothesis the two types of disagreements occur equally often.

- **Null Hypothesis (H_0):** $P(n_{10}) = P(n_{01})$. The models have equal accuracy on the paired items.

- **Alternative Hypothesis (H_1):** $P(n_{10}) \neq P(n_{01})$. There is a systematic difference in model accuracy.

The test is conducted at a significance level of $\alpha = 0.05$. We reject H_0 when $p < 0.05$. The direction is determined by the discordant counts: if $n_{10} > n_{01}$, Model A is more accurate on the paired set; if $n_{01} > n_{10}$, the comparison model B is more accurate.

Exact test , p-value and effect size. Let $n_d = n_{10} + n_{01}$ denote the total number of discordant pairs (i.e., items on which the two models disagree). McNemar's exact test conditions on n_d and treats the number of Model A wins among discordant items, n_{10} , as a binomial random variable under the null hypothesis:

$$n_{10} \sim \text{Binomial}(n_d, 0.5).$$

Under H_0 (equal accuracy on paired items), each discordant item is equally likely to favor either model, so Model A is expected to win about half of the disagreements. The two-sided exact p-value is therefore computed as twice the probability, under $\text{Binomial}(n_d, 0.5)$, of observing a result at least as extreme as the larger of the two discordant counts:

$$p = 2 \sum_{i=b}^{n_d} \binom{n_d}{i} (0.5)^i (1 - 0.5)^{n_d-i},$$

$$b = \max(n_{10}, n_{01}).$$

Equivalently, since $(1 - 0.5) = 0.5$, this simplifies to

$$p = 2 \sum_{i=b}^{n_d} \binom{n_d}{i} \left(\frac{1}{2}\right)^{n_d}, \quad b = \max(n_{10}, n_{01}).$$

To quantify magnitude in addition to statistical significance, we report the paired accuracy gap $\Delta_{\text{Acc.}\%}$ and optionally compute a simple paired advantage (effect size) measure:

$$\Delta = \frac{n_{10} - n_{01}}{N},$$

where N is the total number of paired items. Positive Δ indicates net advantage for Model A; negative Δ indicates net advantage for Model B. This measure is directly interpretable as the net fraction of items on which one model outperforms the other.

E.2 Comparison Results

This section summarizes the pairwise model comparisons on the evaluation set. For clarity, we present results one baseline model at a time: each table and figure shows how that baseline model performs relative to all other models on the same paired trials, along with which differences are statistically significant.

Comparison with GPT-5.2	$\Delta_{\text{Acc.}\%}$	n_{10}	n_{01}	p	Significance
Mistral Large 3	+47.15	66	8	1.81×10^{-12}	Yes
Claude Sonnet 4.5	+30.08	50	13	3.02×10^{-6}	Yes
Deepseek v3.2	+26.83	44	11	8.70×10^{-6}	Yes
Qwen3-235B-A22B	+8.94	21	10	7.08×10^{-2}	No
Gemini 3 Pro	-15.45	1	20	2.10×10^{-5}	Yes

Table 6: Paired significance tests on pooled trials (123 paired outcomes per model). n_{10} counts cases where GPT-5.2 is correct and the other model is incorrect; n_{01} is the reverse.

E.2.1 Test result for GPT-5.2.

The results are presented in Table 6 and visualized in Figure 15. Overall, GPT-5.2 shows higher accuracy than Mistral Large 3, Claude Sonnet 4.5, and DeepSeek v3.2, with statistically significant differences. Its advantage over Qwen3-235B-A22B is modest and not statistically significant, while Gemini 3 Pro performs significantly better than GPT-5.2.

Interpretation for GPT-5.2. McNemar’s exact test evaluates the below things:

- For comparisons where $p < 0.05$, we reject the null hypothesis of equal accuracy and conclude that the difference on the paired items is statistically significant. The direction of the effect is determined by the discordant counts: GPT-5.2 significantly outperforms Mistral Large 3, Claude Sonnet 4.5, and Deepseek v3.2 ($n_{10} > n_{01}$), whereas Gemini 3 Pro significantly outperforms GPT-5.2 ($n_{01} > n_{10}$).
- For the comparison with Qwen3-235B-A22B, $p > 0.05$; therefore, we fail to reject the null hypothesis. The observed difference is not statistically significant, and we cannot conclude that either model systematically outperforms the other on this evaluation set.

Effect Size for GPT-5.2. The observed effect sizes are:

- GPT-5.2 outperforms Mistral Large 3 by a large margin ($\Delta = +0.4715$).
- GPT-5.2 outperforms Claude Sonnet 4.5 by a moderate margin ($\Delta = +0.3008$).
- GPT-5.2 outperforms Deepseek v3.2 by a moderate margin ($\Delta = +0.2683$).
- Gemini 3 Pro outperforms GPT-5.2 ($\Delta = -0.1545$).

E.2.2 Test result for Claude Sonnet 4.5.

The results are presented in Table 7 and Figure 16. Overall, Claude Sonnet 4.5 attains higher accuracy than Mistral Large 3 with a statistically significant difference. Its performance differences relative to Deepseek v3.2 are small and not statistically significant. In contrast, Qwen3-235B-A22B, GPT-5.2, and Gemini 3 Pro achieve higher accuracy than Claude Sonnet 4.5, with statistically significant differences.

Interpretation for Claude Sonnet 4.5. McNemar’s exact test evaluates the following interpretation:

- For comparisons where $p < 0.05$, we reject the null hypothesis of equal accuracy and conclude that the difference on the paired items is statistically significant. The direction of the

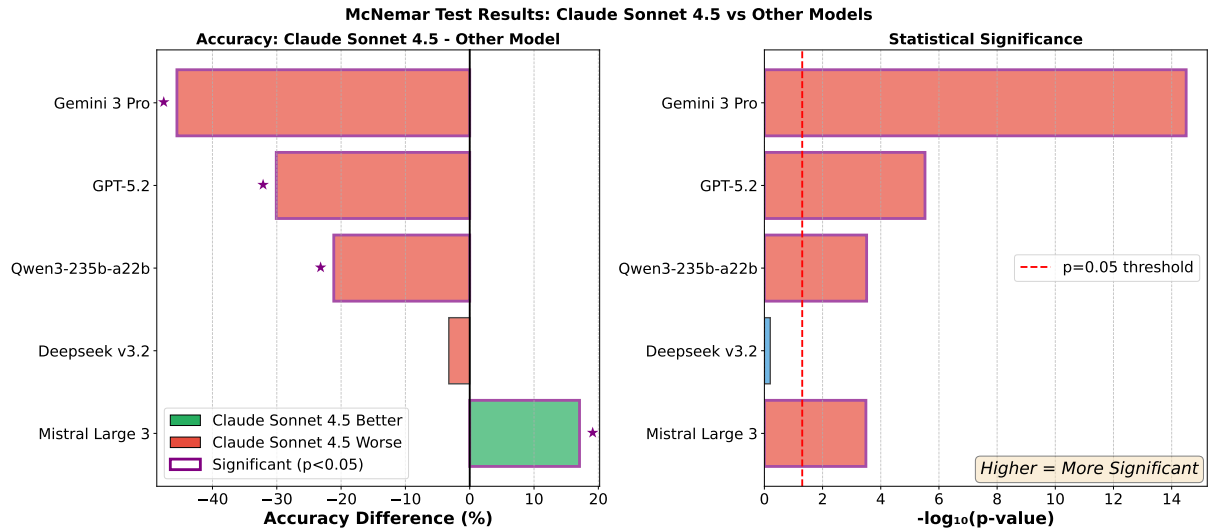


Figure 16: McNemar exact-test results for Claude Sonnet 4.5 versus each comparator model on the same paired trials. Left: accuracy difference $\Delta\text{Acc.}\%$ (Claude Sonnet 4.5 minus comparator); positive values indicate Claude Sonnet 4.5 is more accurate. Right: statistical evidence shown as $-\log_{10}(p)$, with the dashed line marking the $p = 0.05$ significance threshold; higher values indicate stronger evidence of a difference. Significant comparisons (exact McNemar, $\alpha = 0.05$) are highlighted.

Comparison with Claude Sonnet 4.5	$\Delta\text{Acc.}\%$	n_{10}	n_{01}	p	Significance
Mistral Large 3	+17.07	27	6	3.24×10^{-4}	Yes
Deepseek v3.2	-3.25	17	21	6.27×10^{-1}	No
Qwen3-235B-A22B	-21.14	12	38	3.06×10^{-4}	Yes
GPT-5.2	-30.08	13	50	3.02×10^{-6}	Yes
Gemini 3 Pro	-45.53	2	58	3.18×10^{-15}	Yes

Table 7: Paired significance tests on pooled trials (123 paired outcomes per model). n_{10} counts cases where Claude Sonnet 4.5 is correct and the other model is incorrect; n_{01} is the reverse.

effect is determined by the discordant counts: Claude Sonnet 4.5 significantly outperforms Mistral Large 3 ($n_{10} > n_{01}$), whereas Qwen3-235B-A22B, GPT-5.2, and Gemini 3 Pro each significantly outperform Claude Sonnet 4.5 ($n_{01} > n_{10}$).

- For the comparison with Deepseek v3.2, $p > 0.05$; therefore, we fail to reject the null hypothesis. The observed difference is not statistically significant, and we cannot conclude that either model systematically outperforms the other on this evaluation set.

Effect Size for Claude Sonnet 4.5. The observed effect sizes are:

- Claude Sonnet 4.5 outperforms Mistral Large 3 by a moderate margin ($\Delta = +0.1707$).
- Qwen3-235B-A22B outperforms Claude Sonnet 4.5 by a large margin ($\Delta = -0.2114$).
- GPT-5.2 outperforms Claude Sonnet 4.5 by a moderate margin ($\Delta = -0.3008$).
- Gemini 3 Pro outperforms Claude Sonnet 4.5 by a very large margin ($\Delta = -0.4553$).

E.2.3 Test result for Gemini 3 Pro.

The results are presented in Table 8 and Figure 17. Overall, Gemini 3 Pro achieves higher accuracy than GPT-5.2, Qwen3-235B-A22B, Deepseek v3.2, Claude Sonnet 4.5, and Mistral Large 3, with statistically significant differences in all pairwise comparisons.

Interpretation for Gemini 3 Pro. McNemar’s exact test evaluates the following interpretation:

- Since $p < 0.05$ for all comparisons, we reject the null hypothesis in every case and conclude that there are statistically significant differences in accuracy on the paired items. The direction is determined by the discordant counts: for each model, $n_{10} > n_{01}$, indicating that Gemini 3 Pro is significantly more accurate than the comparison model on this evaluation set.

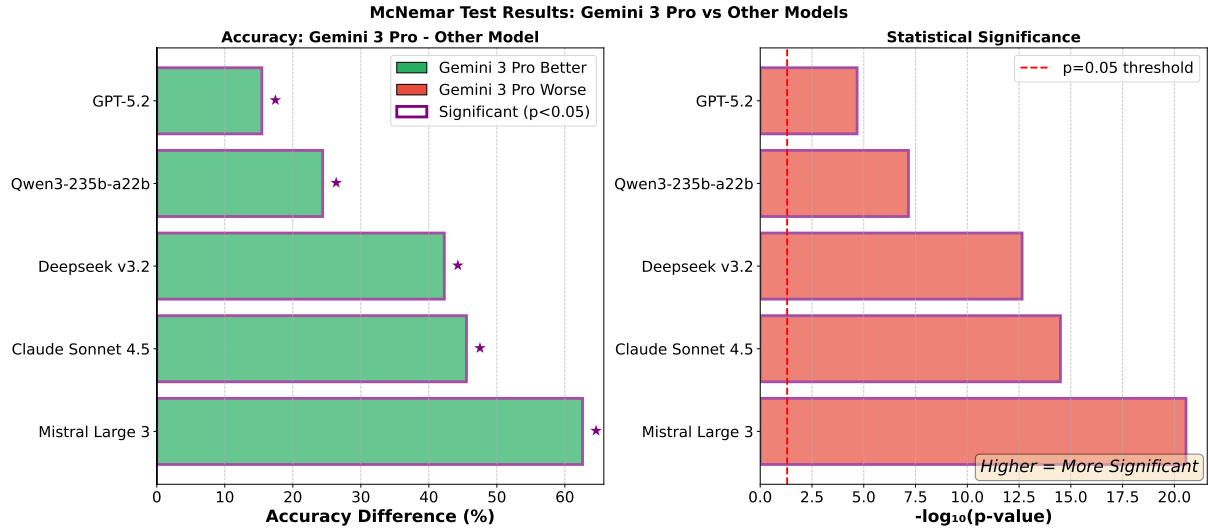


Figure 17: McNemar exact-test results for Gemini 3 Pro versus each comparator model on the same paired trials. Left: accuracy difference $\Delta\text{Acc.}\%$ (Gemini 3 Pro minus comparator); positive values indicate Gemini 3 Pro is more accurate. Right: statistical evidence shown as $-\log_{10}(p)$, with the dashed line marking the $p = 0.05$ significance threshold; higher values indicate stronger evidence of a difference. Significant comparisons (exact McNemar, $\alpha = 0.05$) are highlighted.

Comparison with Gemini 3 Pro	$\Delta\text{Acc.}\%$	n_{10}	n_{01}	p	Significance
GPT-5.2	+15.45	20	1	2.10×10^{-5}	Yes
Qwen3-235B-A22B	+24.39	32	2	6.94×10^{-8}	Yes
Deepseek v3.2	+42.28	55	3	2.26×10^{-13}	Yes
Claude Sonnet 4.5	+45.53	58	2	3.18×10^{-15}	Yes
Mistral Large 3	+62.60	79	2	2.75×10^{-21}	Yes

Table 8: Paired significance tests on pooled trials (123 paired outcomes per model). n_{10} counts cases where Gemini 3 Pro is correct and the other model is incorrect; n_{01} is the reverse.

Effect Size for Gemini 3 Pro. The observed effect sizes are:

- Gemini 3 Pro outperforms GPT-5.2 by a modest margin ($\Delta = +0.1545$).
- Gemini 3 Pro outperforms Qwen3-235B-A22B by a moderate margin ($\Delta = +0.2439$).
- Gemini 3 Pro outperforms Deepseek v3.2 by a large margin ($\Delta = +0.4228$).
- Gemini 3 Pro outperforms Claude Sonnet 4.5 by a large margin ($\Delta = +0.4553$).
- Gemini 3 Pro outperforms Mistral Large 3 by a very large margin ($\Delta = +0.6260$).

E.2.4 Test result for Deepseek v3.2.

The results are presented in Table 9 and Figure 18. Overall, Deepseek v3.2 achieves higher accuracy than Mistral Large 3 with a statistically significant difference. Its performance relative to Claude Sonnet 4.5 is very close and not statistically significant. In contrast, Qwen3-235B-A22B, GPT-5.2, and Gemini 3 Pro perform significantly better than Deepseek v3.2 on the paired evaluation set.

Comparison with Deepseek v3.2	$\Delta\text{Acc.}\%$	n_{10}	n_{01}	p	Significance
Mistral Large 3	+20.33	27	2	1.62×10^{-6}	Yes
Claude Sonnet 4.5	+3.25	21	17	6.27×10^{-1}	No
Qwen3-235B-A22B	-17.89	13	35	2.09×10^{-3}	Yes
GPT-5.2	-26.83	11	44	8.70×10^{-6}	Yes
Gemini 3 Pro	-42.28	3	55	2.26×10^{-13}	Yes

Table 9: Paired significance tests on pooled trials (123 paired outcomes per model). n_{10} counts cases where Deepseek v3.2 is correct and the other model is incorrect; n_{01} is the reverse.

Interpretation for Deepseek v3.2. McNemar's exact test evaluates the following interpretation:

- For comparisons where $p < 0.05$, we reject the null hypothesis of equal accuracy and conclude that the difference on the paired items

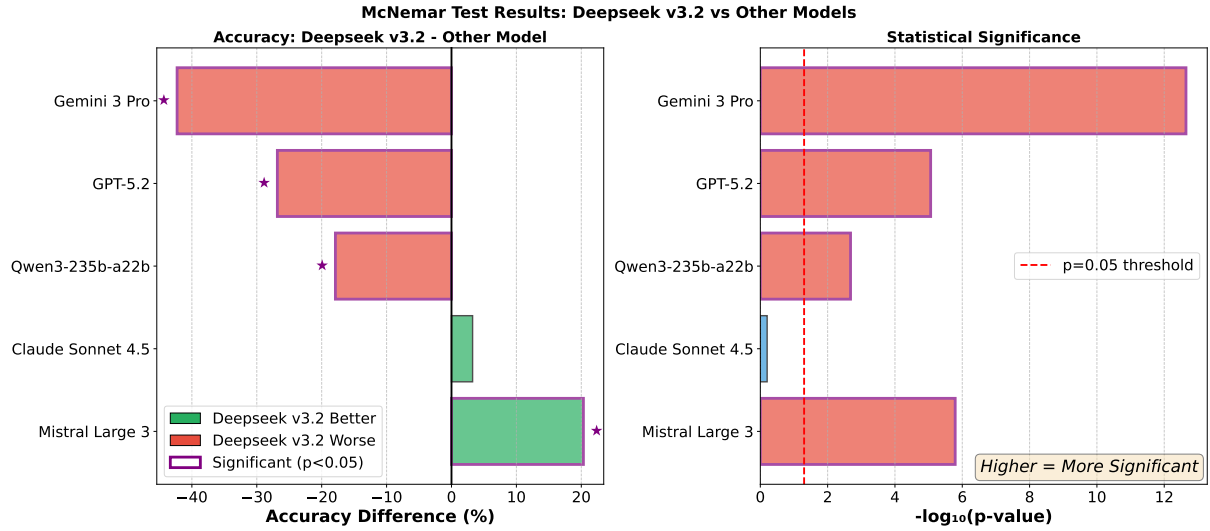


Figure 18: McNemar exact-test results for Deepseek v3.2 versus each comparator model on the same paired trials. Left: accuracy difference $\Delta\text{Acc.}\%$ (Deepseek v3.2 minus comparator); positive values indicate Deepseek v3.2 is more accurate. Right: statistical evidence shown as $-\log_{10}(p)$, with the dashed line marking the $p = 0.05$ significance threshold; higher values indicate stronger evidence of a difference. Significant comparisons (exact McNemar, $\alpha = 0.05$) are highlighted.

is statistically significant. The direction of the effect is determined by the discordant counts: Deepseek v3.2 significantly outperforms Mistral Large 3 ($n_{10} > n_{01}$), whereas Qwen3-235B-A22B, GPT-5.2, and Gemini 3 Pro each significantly outperform Deepseek v3.2 ($n_{01} > n_{10}$).

- For the comparison with Claude Sonnet 4.5, $p > 0.05$; therefore, we fail to reject the null hypothesis. The observed difference is not statistically significant, and we cannot conclude that either model systematically outperforms the other on this evaluation set.

Effect Size for Deepseek v3.2. The observed effect sizes are:

- Deepseek v3.2 outperforms Mistral Large 3 by a moderate margin ($\Delta = +0.2033$).
- Qwen3-235B-A22B outperforms Deepseek v3.2 by a moderate margin ($\Delta = -0.1789$).
- GPT-5.2 outperforms Deepseek v3.2 by a large margin ($\Delta = -0.2683$).
- Gemini 3 Pro outperforms Deepseek v3.2 by a very large margin ($\Delta = -0.4228$).

E.2.5 Test result for Mistral Large 3.

The results are presented in Table 10 and Figure 19. Overall, Mistral Large 3 achieves lower accuracy than all other models in the comparison

set, and each difference is statistically significant. In particular, Claude Sonnet 4.5, Deepseek v3.2, Qwen3-235B-A22B, GPT-5.2, and Gemini 3 Pro all perform significantly better than Mistral Large 3 on the paired evaluation items.

Interpretation for Mistral Large 3. McNemar’s exact test evaluates the following interpretation:

- Since $p < 0.05$ for all comparisons, we reject the null hypothesis in every case and conclude that there are statistically significant differences in accuracy on the paired items. The direction is determined by the discordant counts: for each model, $n_{01} > n_{10}$, indicating that the comparison model is significantly more accurate than Mistral Large 3 on this evaluation set.

Effect Size for Mistral Large 3. The observed effect sizes are:

- Claude Sonnet 4.5 outperforms Mistral Large 3 by a moderate margin ($\Delta = -0.1707$).
- Deepseek v3.2 outperforms Mistral Large 3 by a moderate margin ($\Delta = -0.2033$).
- Qwen3-235B-A22B outperforms Mistral Large 3 by a large margin ($\Delta = -0.3821$).
- GPT-5.2 outperforms Mistral Large 3 by a large margin ($\Delta = -0.4715$).

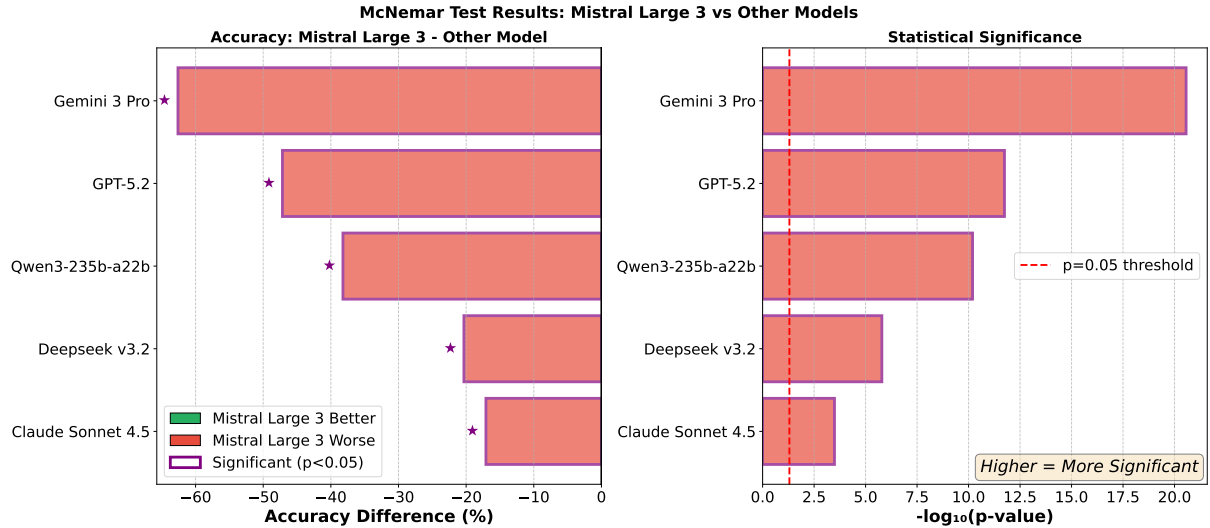


Figure 19: McNemar exact-test results for Mistral Large 3 versus each comparator model on the same paired trials. Left: accuracy difference $\Delta\text{Acc.}\%$ (Mistral Large 3 minus comparator); positive values indicate Mistral Large 3 is more accurate. Right: statistical evidence shown as $-\log_{10}(p)$, with the dashed line marking the $p = 0.05$ significance threshold; higher values indicate stronger evidence of a difference. Significant comparisons (exact McNemar, $\alpha = 0.05$) are highlighted.

Comparison with Mistral Large 3	$\Delta\text{Acc.}\%$	n_{10}	n_{01}	p	Significance
Claude Sonnet 4.5	-17.07	6	27	3.24×10^{-4}	Yes
Deepseek v3.2	-20.33	2	27	1.62×10^{-6}	Yes
Qwen3-235B-A22B	-38.21	5	52	6.40×10^{-11}	Yes
GPT-5.2	-47.15	8	66	1.81×10^{-12}	Yes
Gemini 3 Pro	-62.60	2	79	2.75×10^{-21}	Yes

Table 10: Paired significance tests on pooled trials (123 paired outcomes per model). n_{10} counts cases where Mistral Large 3 is correct and the other model is incorrect; n_{01} is the reverse.

- Gemini 3 Pro outperforms Mistral Large 3 by a very large margin ($\Delta = -0.6260$).

E.2.6 Test result for Qwen3-235B-A22B.

The results are presented in Table 11 and Figure 20. Overall, Qwen3-235B-A22B achieves higher accuracy than Mistral Large 3, Claude Sonnet 4.5, and Deepseek v3.2, with statistically significant differences. Its performance relative to GPT-5.2 shows a modest disadvantage and is not statistically significant. In contrast, Gemini 3 Pro performs significantly better than Qwen3-235B-A22B on the paired evaluation set.

Interpretation for Qwen3-235B-A22B. McNemar’s exact test evaluates the following interpreta-

tion:

- For comparisons where $p < 0.05$, we reject the null hypothesis of equal accuracy and conclude that the difference on the paired items is statistically significant. The direction of the effect is determined by the discordant counts: Qwen3-235B-A22B significantly outperforms Mistral Large 3, Claude Sonnet 4.5, and Deepseek v3.2 ($n_{10} > n_{01}$), whereas Gemini 3 Pro significantly outperforms Qwen3-235B-A22B ($n_{01} > n_{10}$).
- For the comparison with GPT-5.2, $p > 0.05$; therefore, we fail to reject the null hypothesis. The observed difference is not statistically significant, and we cannot conclude that either model systematically outperforms the other on this evaluation set.

Effect Size for Qwen3-235B-A22B. The observed effect sizes are:

- Qwen3-235B-A22B outperforms Mistral Large 3 by a large margin ($\Delta = +0.3821$).
- Qwen3-235B-A22B outperforms Claude Sonnet 4.5 by a moderate margin ($\Delta = +0.2114$).
- Qwen3-235B-A22B outperforms Deepseek v3.2 by a moderate margin ($\Delta = +0.1789$).

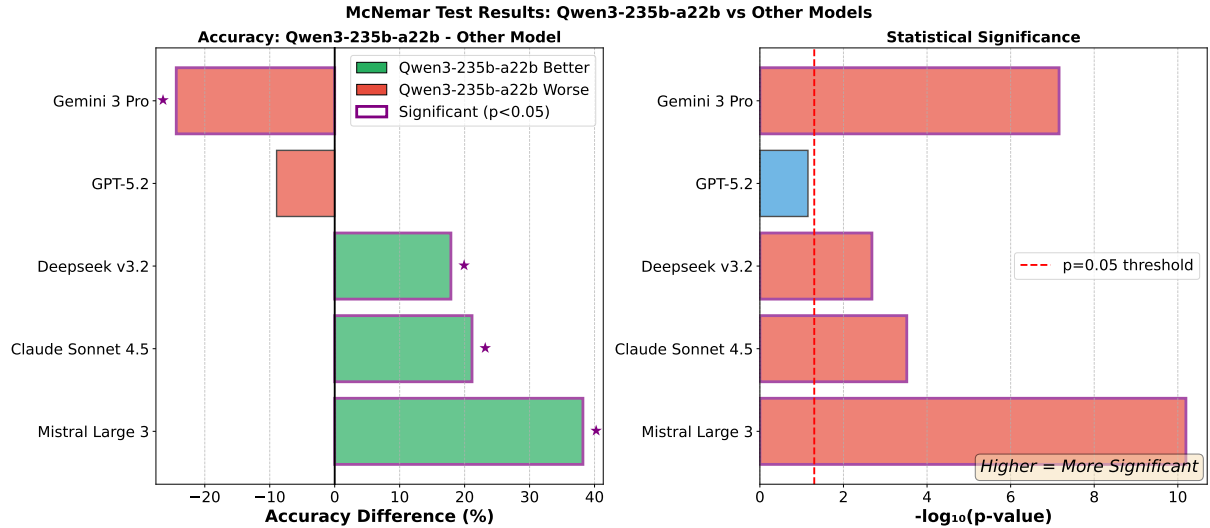


Figure 20: McNemar exact-test results for Qwen3-235B-A22B versus each comparator model on the same paired trials. Left: accuracy difference $\Delta\text{Acc.}\%$ (Qwen3-235B-A22B minus comparator); positive values indicate Qwen3-235B-A22B is more accurate. Right: statistical evidence shown as $-\log_{10}(p)$, with the dashed line marking the $p = 0.05$ significance threshold; higher values indicate stronger evidence of a difference. Significant comparisons (exact McNemar, $\alpha = 0.05$) are highlighted.

Comparison with Qwen3-235B-A22B	$\Delta\text{Acc.}\%$	n_{10}	n_{01}	p	Significance
Mistral Large 3	+38.21	52	5	6.40×10^{-11}	Yes
Claude Sonnet 4.5	+21.14	38	12	3.06×10^{-4}	Yes
Deepseek v3.2	+17.89	35	13	2.09×10^{-3}	Yes
GPT-5.2	-8.94	10	21	7.08×10^{-2}	No
Gemini 3 Pro	-24.39	2	32	6.94×10^{-8}	Yes

Table 11: Paired significance tests on pooled trials (123 paired outcomes per model). n_{10} counts cases where Qwen3-235B-A22B is correct and the other model is incorrect; n_{01} is the reverse.

- Gemini 3 Pro outperforms Qwen3-235B-A22B by a moderate margin ($\Delta = -0.2439$).

F Statistical Analysis: Multiple-Comparison Correction using the Benjamini–Hochberg Procedure

We evaluate differences in model accuracy using *pairwise* McNemar’s exact tests on matched (paired) evaluation items. Because pairwise testing produces multiple hypothesis tests, the probability of false positives (Type I errors) increases if each test is judged using an uncorrected threshold (e.g., $p < 0.05$). To address this, we apply the Benjamini–Hochberg (BH) (Benjamini and Hochberg, 1995) procedure to control the *false discovery rate* (FDR), defined as the expected proportion of false positives among all rejected null hypotheses. Unlike family-wise error rate controls (e.g., Bonferroni (Abdi, 2007)), BH maintains higher power while providing principled error control in multiple-testing settings.

Number of comparisons (m). We compare 6 models, yielding

$$m = \binom{6}{2} = \frac{6 \cdot 5}{2} = 15$$

pairwise McNemar tests (one per unordered model pair).

Assumption Checks We compare Model A with each other model B using McNemar’s *exact* test because the evaluation setup satisfies the key assumptions:

- **Paired observations:** The same set of $N = 123$ evaluation items is answered by both model A and the comparison model B. Therefore, each item yields a paired outcome (one outcome per model on the same item).
- **Binary outcomes:** For every item and every model, the result is coded as Correct (1) or Incorrect (0), which is exactly the outcome type required by McNemar’s test.
- **Independence across items:** Different evaluation items are treated as independent trials

Rank	Comparison	p -value	BH thr.	BH adj. p	BH sig.	$\Delta\text{Acc.}\%$	n_{10}	n_{01}
1	Gemini 3 Pro vs Mistral Large 3	2.75×10^{-21}	0.003333	4.12×10^{-20}	Yes	+62.60	79	2
2	Claude Sonnet 4.5 vs Gemini 3 Pro	3.18×10^{-15}	0.006667	2.38×10^{-14}	Yes	-45.53	2	58
3	Gemini 3 Pro vs Deepseek v3.2	2.26×10^{-13}	0.010000	1.13×10^{-12}	Yes	+42.28	55	3
4	GPT-5.2 vs Mistral Large 3	1.81×10^{-12}	0.013333	6.78×10^{-12}	Yes	+47.15	66	8
5	Mistral Large 3 vs Qwen3-235B-A22B	6.40×10^{-11}	0.016667	1.92×10^{-10}	Yes	-38.21	5	52
6	Gemini 3 Pro vs Qwen3-235B-A22B	6.94×10^{-8}	0.020000	1.73×10^{-7}	Yes	+24.39	32	2
7	Deepseek v3.2 vs Mistral Large 3	1.62×10^{-6}	0.023333	3.48×10^{-6}	Yes	+20.33	27	2
8	GPT-5.2 vs Claude Sonnet 4.5	3.02×10^{-6}	0.026667	5.65×10^{-6}	Yes	+30.08	50	13
9	GPT-5.2 vs Deepseek v3.2	8.70×10^{-6}	0.030000	1.45×10^{-5}	Yes	+26.83	44	11
10	GPT-5.2 vs Gemini 3 Pro	2.10×10^{-5}	0.033333	3.15×10^{-5}	Yes	-15.45	1	20
11	Claude Sonnet 4.5 vs Qwen3-235B-A22B	3.06×10^{-4}	0.036667	0.000405	Yes	-21.14	12	38
12	Claude Sonnet 4.5 vs Mistral Large 3	3.24×10^{-4}	0.040000	0.000405	Yes	+17.07	27	6
13	Deepseek v3.2 vs Qwen3-235B-A22B	0.002088	0.043333	0.002409	Yes	-17.89	13	35
14	GPT-5.2 vs Qwen3-235B-A22B	0.070756	0.046667	0.075810	No	+8.94	21	10
15	Claude Sonnet 4.5 vs Deepseek v3.2	0.627103	0.050000	0.627103	No	-3.25	17	21

Table 12: Benjamini–Hochberg (BH) correction applied to $m = 15$ pairwise McNemar tests (FDR $q = 0.05$). P-values are sorted in ascending order (Rank). “BH thr.” denotes the BH critical value $\frac{i}{m}q$ for rank i , “BH adj. p ” denotes BH-adjusted p-values, and “BH sig.” indicates significance after correction. $\Delta\text{Acc.}\%$ is the paired accuracy gap (positive favors the first-listed model). n_{10} and n_{01} are discordant counts as defined for each pairwise McNemar comparison.

(the correctness on one item does not determine correctness on another).

- **Exact inference:** McNemar’s exact test uses the binomial distribution over discordant pairs and does not rely on large-sample approximations. This is especially suitable when the number of disagreements is small or imbalanced.

Hypothesis Formulation We test whether the two models have equal accuracy on paired items. Equivalently, under the null hypothesis the two types of disagreements occur equally often.

- **Null Hypothesis (H_0):** $P(n_{10}) = P(n_{01})$.
The models have equal accuracy on the paired items.
- **Alternative Hypothesis (H_1):** $P(n_{10}) \neq P(n_{01})$.
There is a systematic difference in model accuracy.

BH-adjusted p-values. Let m denote the total number of hypothesis tests, and let $p_{(1)} \leq \dots \leq p_{(m)}$ be the raw p-values sorted in ascending order, where $p_{(i)}$ is the i -th smallest p-value (rank i). For the i -th smallest raw p-value, the BH-adjusted value is computed as

$$\tilde{p}_{(i)} = \min_{j \geq i} \left\{ \frac{m}{j} p_{(j)} \right\},$$

where j indexes ranks from i to m . The value $\tilde{p}_{(i)}$ is truncated to at most 1 if necessary. A comparison is considered significant after correction when $\tilde{p}_{(i)} \leq q$, where q is the target false discovery rate (FDR) level (e.g., $q = 0.05$).

Benjamini–Hochberg (BH) procedure. To correct for multiple testing, we apply the BH procedure as follows:

- **Ordering:** All $m = 15$ p-values are sorted in ascending order (Rank $i = 1$ to 15).
- **FDR Level:** We choose a desired FDR level $Q = 0.05$.
- **Threshold Calculation:** Let $p_1, \dots, p_m$ be the $m = 15$ raw p-values from the pairwise McNemar tests, sorted as $p_{(1)} \leq \dots \leq p_{(m)}$ with corresponding ranks $i = 1, \dots, m$. For an FDR level $q = 0.05$, BH defines the rank-specific critical values (“BH thresholds”)

$$t_i = \frac{i}{m} q.$$

- **Rejection Criterion:** We find the largest p-value such that $p_{(k)} \leq t_k$ and reject the null hypotheses for that result and all results with smaller p-values. Equivalently, using BH-adjusted p-values, we reject all hypotheses with $\tilde{p}_{(i)} \leq 0.05$.

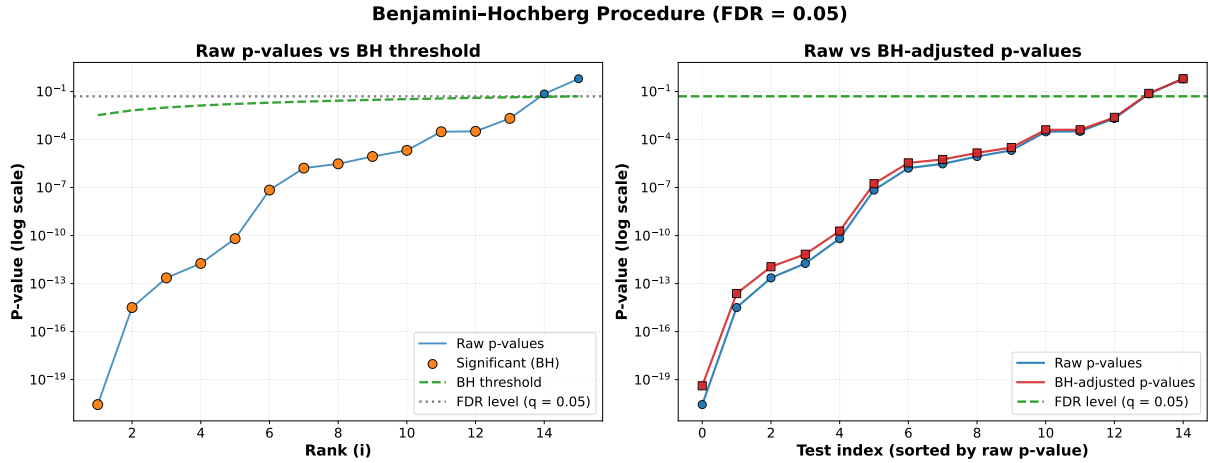


Figure 21: Benjamini–Hochberg (BH) multiple-comparison correction at FDR $q = 0.05$ for the set of pairwise McNemar tests (p-values sorted in ascending order). **Left:** raw p -values (log scale) plotted against their BH rank-specific critical values; points below the BH threshold are declared significant after FDR control. **Right:** comparison of raw and BH-adjusted p -values (log scale) across tests, with the horizontal line indicating the target FDR level ($q = 0.05$); adjusted values at or below this line correspond to rejections under BH.

Results and interpretation. Table 12 and Figure 21 reports the raw p -values, BH thresholds, and BH-adjusted p -values for all $m = 15$ pairwise comparisons. Using BH at $q = 0.05$,

- Since $\tilde{p} < 0.05$, there is enough evidence to reject the null hypothesis for 13 out of 15 pairwise comparisons, and we conclude that these model pairs exhibit statistically significant accuracy differences after BH correction. In particular, Gemini 3 Pro differs significantly from Mistral Large 3 and shows the strongest evidence among all comparisons. For GPT-5.2, the results indicate statistically significant advantages over Mistral Large 3, Claude Sonnet 4.5, and Deepseek v3.2 on the paired evaluation set. We also reject the null hypothesis for GPT-5.2 versus Gemini 3 Pro; the negative accuracy gap indicates that Gemini 3 Pro is more accurate than GPT-5.2.
- Since $\tilde{p} \geq 0.05$, there is not enough evidence to reject the null hypothesis for GPT-5.2 vs. Qwen3-235B-A22B and for Claude Sonnet 4.5 vs. Deepseek v3.2. Therefore, after BH correction, we cannot claim a systematic difference in accuracy for these two pairs.

Key Findings. The Benjamini–Hochberg correction reveals that most model pairs exhibit statistically significant differences in accuracy on the paired evaluation set. Specifically, 13 out of the 15 pairwise McNemar tests remain significant after

controlling the false discovery rate at $q = 0.05$, indicating that performance differences among the evaluated models are generally robust rather than due to random variation. The strongest statistical evidence appears in comparisons involving Gemini 3 Pro, which significantly outperforms several other models, including Mistral Large 3 and GPT-5.2. GPT-5.2 also shows statistically significant advantages over Mistral Large 3, Claude Sonnet 4.5, and Deepseek v3.2. In contrast, two model pairs—GPT-5.2 vs. Qwen3-235B-A22B and Claude Sonnet 4.5 vs. Deepseek v3.2—do not show statistically significant differences after correction, suggesting comparable performance between those models on this evaluation benchmark.