Program Synthesis Failure Analysis

We performed a comprehensive analysis of program synthesis errors to evaluate the effectiveness of LLMs in generating code. For this evaluation, the LLMs were instructed to independently synthesize code 50 times based on a predefined prompt. The prompt included directives for generating two distinct checking functions: one responsible for validating the sample plan and another for identifying the planning problem embedded within the prompt. The synthesized code was subjected to semantic checks to ensure its safety and correctness. If the code successfully passed these checks, its functionality was further assessed using 10 randomly selected samples from benchmark datasets to evaluate its progress. The analysis was conducted across three task settings: Blocksworld, Logistics, and Logistics with an optimization-instruction prompt. The results of this evaluation are presented in Tables 1 to 3. We did not include the studies of SAT and Game of 24 because the models can always synthesize correct programs.

The identified errors were classified into five categories representing a spectrum of program correctness: (1) Syntactical errors (**Syn**), involving issues in code syntax; (2) Semantic errors during sanity checks on trivial functions (**Sem1**); (3) Semantic errors during sanity checks on examples provided in the prompt (**Sem2**); (4) Partial correctness (**Partial**), where the code passed some sample tests; and (5) Complete correctness (**All**), where the code passed all sample tests, indicating fully correct programs. These categories form a hierarchical structure, where programs passing a higher stage necessarily pass preceding stages.

For Blocksworld tasks, results indicate that o1 reliably generates correct code for this domain. Deepseek's models perform adequately when syntactical errors are absent. In contrast, GPT-4o frequently generates partially correct solutions, suggesting limited understanding of the underlying rules.

For Logistics tasks, both o1 and o1-mini demonstrate high success rates in generating functional code for most sample cases. However, unoptimized code relying on naive enumeration successfully solves simple instances but often triggers timeouts for more complex ones.

In Logistics tasks with optimization-specific instructions, o1's synthesized code is mostly correct when passing tests. Deepseek models exhibit difficulty but occasionally generate functional solutions. The task proves too challenging for GPT-4o, which struggles to produce working code in this setting.

Table 1. Error analysis on Blocksworld. Results show the number of error types out of **50** times of code generation running on **10** random samples.

Model	Syn	Sem1	Sem2	Partial	All
GPT-4o	15	3	8	20	4
o1-mini	5	11	2	9	23
o1	0	0	0	10	40
DeepSeek-V3	24	0	8	2	16
DeepSeek-R1	25	4	6	5	10

Table 2. Error analysis on Logistics. Results show the number of error types out of **50** times of code generation running on **10** random samples.

Model	Syn	Sem1	Sem2	Partial	All
GPT-4o	15	16	11	8	0
o1-mini	4	7	9	30	0
o1	5	0	5	40	0
DeepSeek-V3	23	1	8	18	0
DeepSeek-R1	24	0	8	18	0

Table 3. Error analysis on Logistics with the prompt for optimization. Results show the number of error types out of **50** times of code generation running on **10** random samples.

Model	Syn	Sem1	Sem2	Partial	All
GPT-4o	5	16	13	6	0
o1-mini	4	10	31	2	3
o1	5	0	27	2	16
DeepSeek-V3	19	1	28	1	1
DeepSeek-R1	15	4	29	0	2