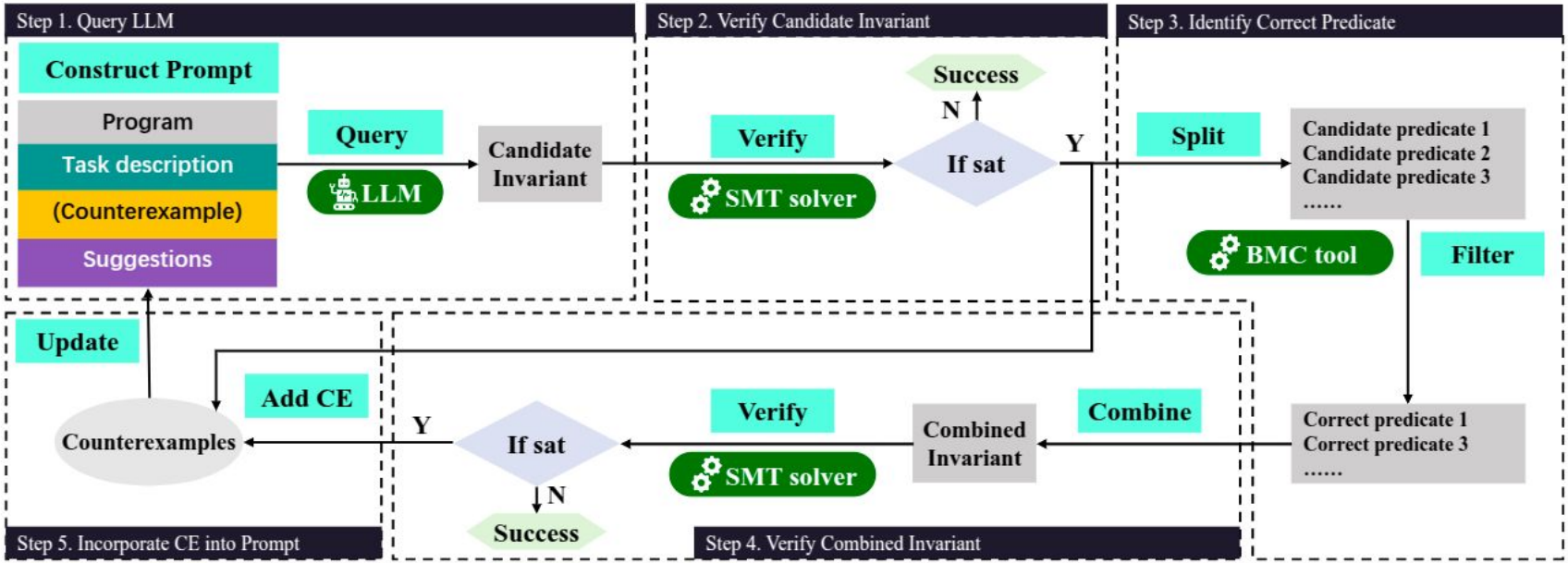
# LaM4Inv Improvements

Almir Cunha, Lucas Van-Lume, Matheus Veras, Caio Possidio, Jose Jovanney

Centro de Informática UFPE
Universidade Federal de Pernambuco

# LaM4Inv Overview



# Code Improvements

# Spaghetti Code

```
if LLM == "GPT4o":
    gptAnswer = openai.ChatCompletion.create(
            model="gpt-4o",
            messages=[{"role": "user", "content": get_prompt(cProgram,promptType,previousAns,counterexample)}]
        )
    result = gptAnswer["choices"][0]["message"]["content"]
if LLM == "GPT4":
    gptAnswer = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[{"role": "user", "content": get_prompt(cProgram,promptType,previousAns,counterexample)}]
        )
    result = gptAnswer["choices"][0]["message"]["content"]
elif LLM == "GPT4Turbo":
    gptAnswer = openai.ChatCompletion.create(
            model="gpt-4-turbo",
            messages=[{"role": "user", "content": get_prompt(cProgram,promptType,previousAns,counterexample)}]
        )
    result = gptAnswer["choices"][0]["message"]["content"]
elif LLM == "GPT3.5Turbo":
    gptAnswer = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": get_prompt(cProgram,promptType,previousAns,counterexample)}]
        )
    result = gptAnswer["choices"][0]["message"]["content"]
elif LLM == "Man":
    print("Input loop invariant:")
    result=input()
elif LLM == "Exist":
    if readexistanscount < len(existans):
        result = existans[readexistanscount]
    else:
        result = ""
elif LLM == "Llama3":
    result = Llama3chat.getLlamaAnswer(get_prompt(cProgram,promptType,previousAns,counterexample))

result=add_parentheses_to_pow_args(result)
result=extract_assert_statements(result)
for tmp in result:
    if '?' in tmp:
        result.remove(tmp)
        result.extend(rewrite_case_split_into_disjunction(tmp))

smtlib2=convert.convert_c_assert_list_to_smtlib2(result)

if BMC == True:
    for index in range(len(result)):
        CAssertion=result[index]
```

```
if BMC == True:
    for index in range(len(result)):
        CAssertion=result[index]
        CAssertionlist, outer_operator=spilit.c_assert_spilit(CAssertion)
        if outer_operator=='||':
            ifaddall=esbmc_and(cProgram,CAssertion[CAssertion.find("(")+1:CAssertion.rfind(")")])
            if ifaddall:
                tempansset=[]
                for subassertion in CAssertionlist:
                    if subassertion not in tempansset and undefined_function(subassertion):
                        ifAdd=esbmc_or(cProgram,subassertion)
                        if ifAdd:
                            tempansset.append(subassertion)
                resans=""
                for res in tempansset:
                    resans=resans+"("+res+")"+"||"
                if resans[0:-2] not in AnsSet:
                    AnsSet.append(resans[0:-2])
        else:
            for subassertion in CAssertionlist:
                if subassertion not in AnsSet and undefined_function(subassertion):
                    ifAdd=esbmc_and(cProgram,subassertion)
                    if ifAdd:
                        AnsSet.append(subassertion)
else:
    AnsSet=[]
return smtlib2,result,AnsSet
```

# Spaghetti Code

```
if Counter_example.kind=="n" and len(PT)<50:
    for i in range(0,2):
        lengthAnsSetbefore=len(AnsSet)
        pt,gptans,AnsSet = GPT.get_answer(cProgram,2,gptAnswer[Iteration],str(Counter_example.assignment),AnsSet,existans,readexistanscount)
        readexistanscount +=1
        lengthAnsSetafter=len(AnsSet)
        AnsSetChanged=(lengthAnsSetafter!=lengthAnsSetbefore) or AnsSetChanged
        for count in range(len(gptans)):
            if gptans[count] not in gptAnswer:
                PT.append(pt[count])
                gptAnswer.append(gptans[count])
        print("GPT Answer: ", gptAnswer)
        result_file.write("GPT Answer: "+str(gptAnswer)+"\n")
        print("AnsSet: ", AnsSet)
        result_file.write("AnsSet: "+str(AnsSet)+'\n')
elif Counter_example.kind=="p"and len(PT)<50:
    for i in range(0,2):
        lengthAnsSetbefore=len(AnsSet)
        pt,gptans,AnsSet = GPT.get_answer(cProgram,1,gptAnswer[Iteration],str(Counter_example.assignment),AnsSet,existans,readexistanscount)
        readexistanscount +=1
        lengthAnsSetafter=len(AnsSet)
        AnsSetChanged=(lengthAnsSetafter!=lengthAnsSetbefore) or AnsSetChanged
        for count in range(len(gptans)):
            if gptans[count] not in gptAnswer:
                PT.append(pt[count])
                gptAnswer.append(gptans[count])
        print("GPT Answer: ", gptAnswer)
        result_file.write("GPT Answer: "+str(gptAnswer)+"\n")
        print("AnsSet: ", AnsSet)
        result_file.write("AnsSet: "+str(AnsSet)+'\n')
elif Counter_example.kind=="i"and len(PT)<50:
    for i in range(0,2):
        lengthAnsSetbefore=len(AnsSet)
        pt,gptans,AnsSet = GPT.get_answer(cProgram,3,gptAnswer[Iteration],str(Counter_example.assignment),AnsSet,existans,readexistanscount)
        readexistanscount +=1
        lengthAnsSetafter=len(AnsSet)
        AnsSetChanged=(lengthAnsSetafter!=lengthAnsSetbefore) or AnsSetChanged
        for count in range(len(gptans)):
            if gptans[count] not in gptAnswer:
                PT.append(pt[count])
                gptAnswer.append(gptans[count])
        print("GPT Answer: ", gptAnswer)
        result_file.write("GPT Answer: "+str(gptAnswer)+"\n")
```

# Spaghetti Code

- Benchmarks
- GPT_chat
  - GPT.py
  - Llama3chat.py
  - convert.py
  - readexistans.py
  - spilit.py
- Result
- SMT_Solver
  - SMT_verifier.py
- Utilities
  - SMT_parser.py
  - TimeController.py
- windows-release
- .gitignore
- Config.py
- README.md
- RunAllLinear.py
- averageTimeAndProposal.py
- extract_preconditions.py
- main.py
- requirements.txt
- requirementsllama.txt

```python
def esbmc_or(cProgram,subassertion):
    assertion='assert(!('+subassertion+'));'
    esbmcProgram=cProgram.replace('unknown()','rand()%2==0')
    check_dir_path = "./check/"
    if not os.path.exists(check_dir_path):
        os.makedirs(check_dir_path)
    file=open(check_dir_path+config.resultpath+".c","w")
    leftcount=0
    rightcount=0
    judge=True
    for lines in esbmcProgram.splitlines():
        if "while" in lines:
            lines=assertion+"\n"+lines+"\n"+assertion
        if "assume" in lines:
            condition=lines[lines.find("(")+1:lines.rfind(")")]
            lines="if(!("+condition+")) return 0;"
        if "{" in lines:
            leftcount+=1
        if "}" in lines:
            rightcount+=1
        if leftcount>1 and leftcount-1==rightcount and judge:
            file.write(assertion+"\n}\n}")
            judge=False
            break
        file.write(lines+"\n")
    file.close()
    if Verification == "esbmc":
        if maxkinduction:
            command = [ESBMC_BIN_PATH, ".\\check\\"+config.resultpath+".c", "--floatbv", "--k-induction", "--max-k-step", str(
        else:
            command = [ESBMC_BIN_PATH, ".\\check\\"+config.resultpath+".c", "--floatbv", "--k-induction"]
    else:
        if maxkinduction:
            command = [".\\cbmc\\bin\\cbmc.exe", ".\\check\\"+config.resultpath+".c", "--unwind", str(maxkstep)]
        else:
            command = [".\\cbmc\\bin\\cbmc.exe", ".\\check\\"+config.resultpath+".c", "--unwind", "50"]

    val = run_command_with_timeout(command, timeout_seconds)

    if val != "subprocess.TimeoutExpired":
```

# Code Improvements

Files

main

Go to file

- bmc
  - __init__.py
  - bmc.py
  - esbmc.py
- code_handler
  - __init__.py
  - c_code_handler.py
  - c_formula_handler.py
  - code_handler.py
  - formula_handler.py
- generator
- inv_smt_solver
- llm
  - __init__.py
  - llm.py
  - openai.py
  - transformers.py
- predicate_filtering
- results
- smt
  - __init__.py
  - solver.py
  - z3_solver.py

LaM4Inv / llm / **openai.py**

Code | Blame | 62 lines (52 loc) · 2.1 KB

```python

class OpenAI(LLM):
    def __init__(self, model: OpenAIModel, api_key: str = None, base_url: str = None):
        self.model = model
        self.client = OpenAIClient(api_key=api_key, base_url=base_url)
        self._unsupported_params = {
            ChatGPTModel.O1_MINI: ["presence_penalty"],
            ChatGPTModel.O3_MINI: ["presence_penalty"],
        }

    def _get_messages(self, chat: Chat) -> list:
        return [
            {
                "role": message.role.value,
                "content": message.content,
            }
            for message in chat.messages
        ]

    def _get_presence_penalty(self, options: ChatOptions) -> float:
        if not options or options.presence_penalty is None:
            return NOT_GIVEN
        if self.model in self._unsupported_params and 'presence_penalty' in self._unsupported_params[self.model]:
            return NOT_GIVEN
        return 2*options.presence_penalty

    def _get_temperature(self, options: ChatOptions) -> float:
        if not options or options.temperature is None:
            return NOT_GIVEN
        if self.model in self._unsupported_params and 'temperature' in self._unsupported_params[self.model]:
            return NOT_GIVEN
        return 2*options.temperature

    def chat(self, chat: Chat, options: ChatOptions = None) -> str:
        completions = self.client.chat.completions.create(
            model=self.model.value,
            messages=self._get_messages(chat),
            presence_penalty=self._get_presence_penalty(options),
            temperature=self._get_temperature(options),
        )
        response = completions.choices[0].message.content
```

# Improved Context

# Weak Context

```
if LLM == "GPT4o":
    gptAnswer = openai.ChatCompletion.create(
            model="gpt-4o",
            messages=[{"role": "user", "content": get_prompt(cProgram,promptType,previousAns,counterexample)}]
        )
    result = gptAnswer["choices"][0]["message"]["content"]
```

# Weak Context

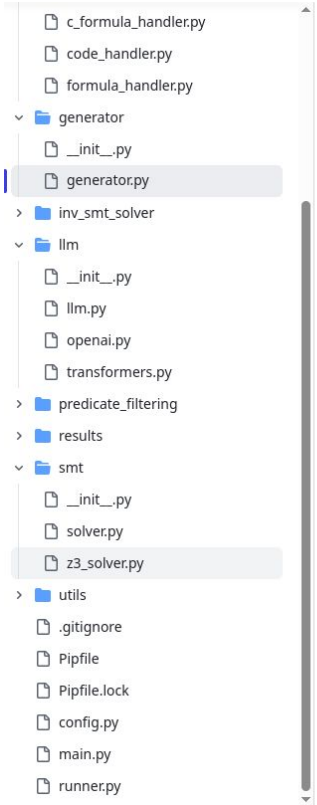
```
def get_prompt1(cProgram,promptType,previousAns,counterexample):
    if promptType==0:
        cProgram=cProgram+" Print loop invariants as valid C assertions that help prove the assertion. \
In order to get a correct answer, You may want to consider both the situation of not entering the loop and the situation of jumping out of the loop. \
If some of the preconditions are also loop invariant, you need to add them to your answer as well. \
Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'"
    elif promptType==1:
        cProgram=cProgram+" Print loop invariants as valid C assertions that help prove the assertion. \
Your previous answer '"+previousAns+"'is too strict and not reachable. \
The Reachability of the loop invariant means that the loop invariant I can be derived based on the pre-condition P, i.e. P ⇒ I. \
The following is a counterexample given by z3: "+counterexample+". \
In order to get a correct answer, You may want to consider the initial situation where the program won't enter the loop. \
Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'"
    elif promptType==2:
        cProgram=cProgram+" Print loop invariants as valid C assertions that help prove the assertion. \
Your previous answer '"+previousAns+"'is too weak and not provable. \
The Provability of the loop invariant means that after unsatisfying loop condition B, we can prove the post-condition Q, i.e. (I ∧ ¬ B) ⇒ Q. \
The following is a counterexample given by z3: "+counterexample+". \
In order to get a correct answer, you may want to consider the special case of the program executing to the end of the loop. If some of the preconditions are also loop invariant, you need to add
Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'"
    elif promptType==3:
        cProgram=cProgram+" Print loop invariants as valid C assertions that help prove the assertion. \
Your previous answer '"+previousAns+"'is not inductive. \
The Inductive of the loop invariant means that if the program state satisfies loop condition B, the new state obtained after the loop execution S still satisfies, i.e. {I ∧ B} S {I}. \
The following is a counterexample given by z3: "+counterexample+". \
In order to get a correct answer, You may want to consider the special case of the program executing to the end of the loop. \
Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'"
    return cProgram

def get_prompt2(cProgram,promptType,previousAns,counterexample):
    cProgram=cProgram+" Print loop invariants as valid C assertions that help prove the assertion. \
Use '&&' or '||' if necessary. Don't explain. Your answer should be 'assert(...);'"
    return cProgram
```

# Improved Context With Chat Completions

- code_handler.py
- formula_handler.py
- generator
  - __init__.py
  - generator.py
- inv_smt_solver
- llm
  - __init__.py
  - llm.py
  - openai.py
  - transformers.py
- predicate_filtering
- results
- smt
  - __init__.py
  - solver.py
  - z3_solver.py
- utils
- .gitignore
- Pipfile
- Pipfile.lock
- config.py
- main.py
- runner.py

```python
from abc import ABC, abstractmethod
from pydantic import BaseModel
from enum import Enum

class ChatOptions(BaseModel):
    presence_penalty: float = None
    temperature: float = None

class ChatMessageRole(Enum):
    user = "user"
    assistant = "assistant"

class ChatMessage(BaseModel):
    role: ChatMessageRole
    content: str

class Chat(BaseModel):
    messages: list[ChatMessage] = []

    def add_user_message(self, message: str):
        self.messages.append(ChatMessage(role=ChatMessageRole.user, content=message))

    def add_assistant_response(self, message: str):
        self.messages.append(ChatMessage(role=ChatMessageRole.assistant, content=message))

    def reset(self):
        self.messages = []

class LLM(ABC):
    @abstractmethod
    def chat(self, chat: Chat, options: ChatOptions = None) -> str:
        pass

    def __str__(self) -> str:
        return self.model.value
```

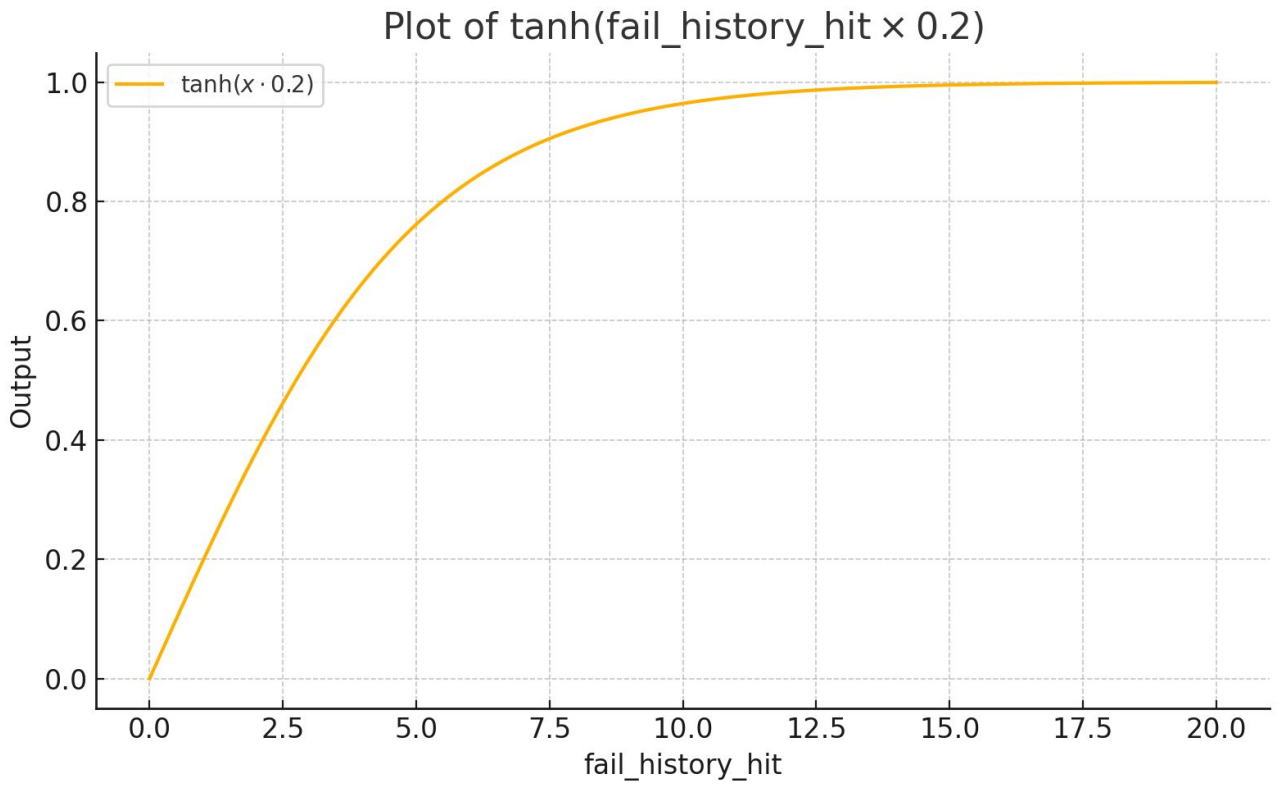
# Improved Context With Chat Completions




```python
from llm.llm import LLM, ChatOptions, Chat
from inv_smt_solver.counter_example import CounterExample, CounterExampleKind
from code_handler.code_handler import CodeHandler

class Generator:
    def __init__(self, code_handler: CodeHandler):
        self.code_handler = code_handler
        self._chat = Chat()

    def get_messages(self):
        return self._chat.messages

    def _get_base_llm_message(self) -> str: ...
"""

    def _format_feedback(self, fails: list[tuple[str, CounterExample]]) -> str: ...
        return fails_prompt

    def _get_feedback_llm_message(self, last_fails: list[tuple[str, CounterExample]]) -> str: ...
"""

    def _parse_llm_response(self, output: str) -> list[str]: ...
        return expressions

    def generate(self, llm: LLM, feedback: list[tuple[str, CounterExample]] = None, chat_options: ChatOptions = None) -> list[str]:
        if feedback:
            message = self._get_feedback_llm_message(feedback)
        else:
            message = self._get_base_llm_message()
        self._chat.add_user_message(message)

        response = llm.chat(self._chat, chat_options)
        self._chat.add_assistant_response(response)

        return self._parse_llm_response(response)

    def reset(self):
        self._chat.reset()
```

# Adaptive Presence Penalty

# Adaptive Presence Penalty

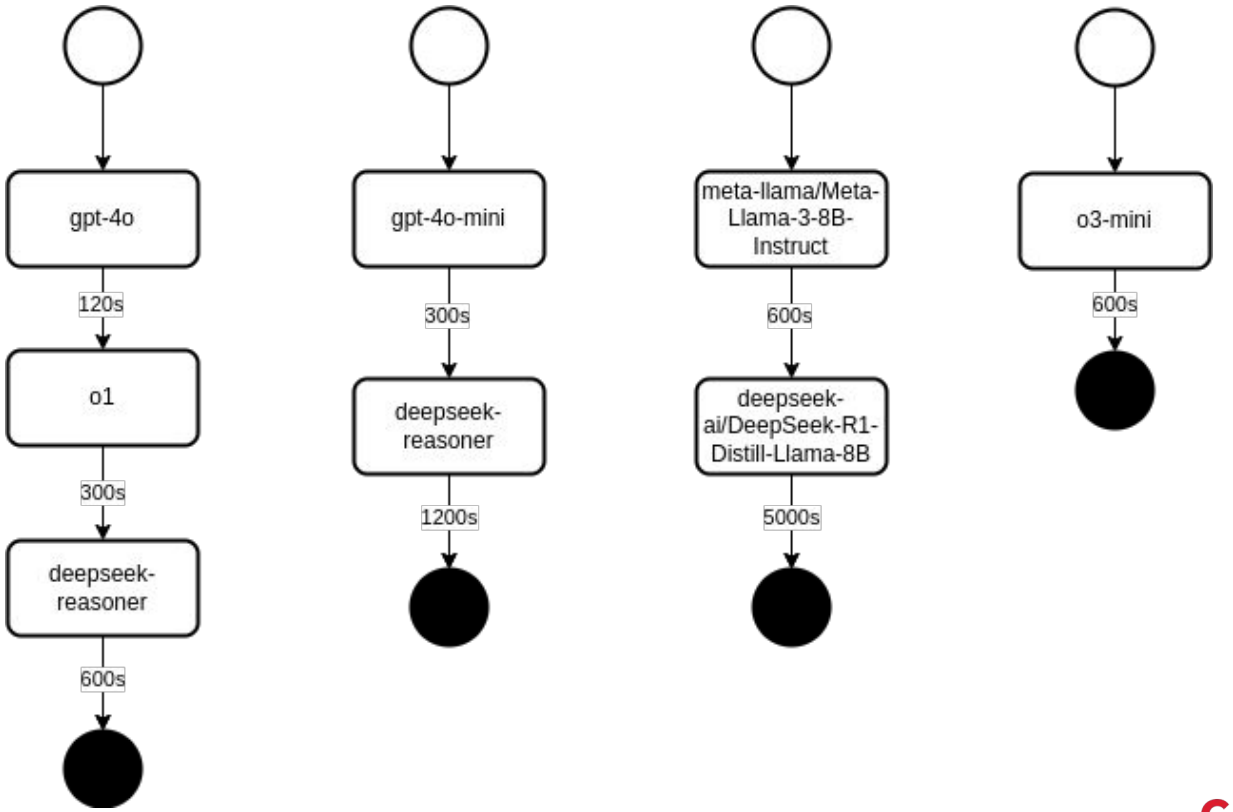


# Adaptive Presence Penalty

```python
class Runner:
    def __init__(…

    def _log_solution(self, solution: str, llm: LLM, start_time: float, predicate_filtering: bool):…

    def _handle_solution(self, solution: str, llm: LLM, start_time: float, predicate_filtering:bool = Fals

    def _next_pipeline_step(self) -> tuple[LLM, float]:…

    def _get_presence_penalty(self) -> float:
        return math.tanh(self._fail_history_hit * self.presence_penalty_scale)

    def _predicate_filtering(self, candidates: list[str]) -> str:…

    def _verify_candidates(self, candidates: list[str]) -> tuple[str, list[str]]:
        fails = []
        for candidate in candidates:
            self._logger.info(f'Verifying candidate: {candidate}')

            if candidate in self._fail_history:
                self._fail_history_hit += 1
                fails.append((candidate, (self._fail_history[candidate])))
                self._logger.info(f'Candidate already in fail history: {candidate}')
                continue

            formula = self.formula_handler.extract_formula(candidate)
            smt_lib2_formula = self.formula_handler.to_smt_lib2(formula)
            counter_example = self.inv_smt_solver.get_counter_example(smt_lib2_formula)
            if counter_example is None:
                return candidate, fails

            self._logger.info(f'Candidate failed verification')
            fails.append((candidate, counter_example))

            self._logger.info(f'Adding candidate to fail history: {candidate}')
            self._fail_history[candidate] = counter_example

        return None, fails
```

# Pipeline

# Pipeline


gpt-4o
120s
o1
300s
deepseek-reasoner
600s
gpt-4o-mini
300s
deepseek-reasoner
1200s
meta-llama/Meta-Llama-3-8B-Instruct
600s
deepseek-ai/DeepSeek-R1-Distill-Llama-8B
5000s
o3-mini
600s

# Pipeline

```python
class Runner:
    def __init__(…

    def _log_solution(self, solution: str, llm: LLM, start_time: float, predicate_filtering: bool):…

    def _handle_solution(self, solution: str, llm: LLM, start_time: float, predicate_filtering:bool = False

    def _next_pipeline_step(self) -> tuple[LLM, float]:
        if self._curr_pipeline_step_index is None:
            self._curr_pipeline_step_activation_time = time.time()
            self._curr_pipeline_step_index = 0
            return self.pipeline[0]

        time_spent = time.time() - self._curr_pipeline_step_activation_time
        curr_step = self.pipeline[self._curr_pipeline_step_index]
        if time_spent >= curr_step[1] and self._curr_pipeline_step_index == len(self.pipeline) - 1:
            return (None, None)
        if time_spent >= curr_step[1]:
            self._reset_generator()
            self._curr_pipeline_step_index += 1
            self._curr_pipeline_step_activation_time = time.time()

        return self.pipeline[self._curr_pipeline_step_index]

    def _get_presence_penalty(self) -> float:…
```

# Balanced Pipeline

Time/cost balanced pipeline


Triviality filter

**Exploration**: populates the predicate filter set

**Exploitation:** conducts a fast reasoning to identify key candidates

**Failover:** escalate to the largest, though slower, model to find the solution

# Baseline Results

# Settings

**CPU:** Intel® Core™ i9-13900K

**RAM:** 48GiB

**GPU:** GeForce RTX 4090

# Results

**Total benchmarks:** 316

**Solutions found:** 309

**Success rate:** 97.8%

**Average runtime:** 35.69 seconds

# Improvements Results

# Settings

**CPU:** Intel® Core™ i7-11390H

**RAM:** 16GiB

# Results

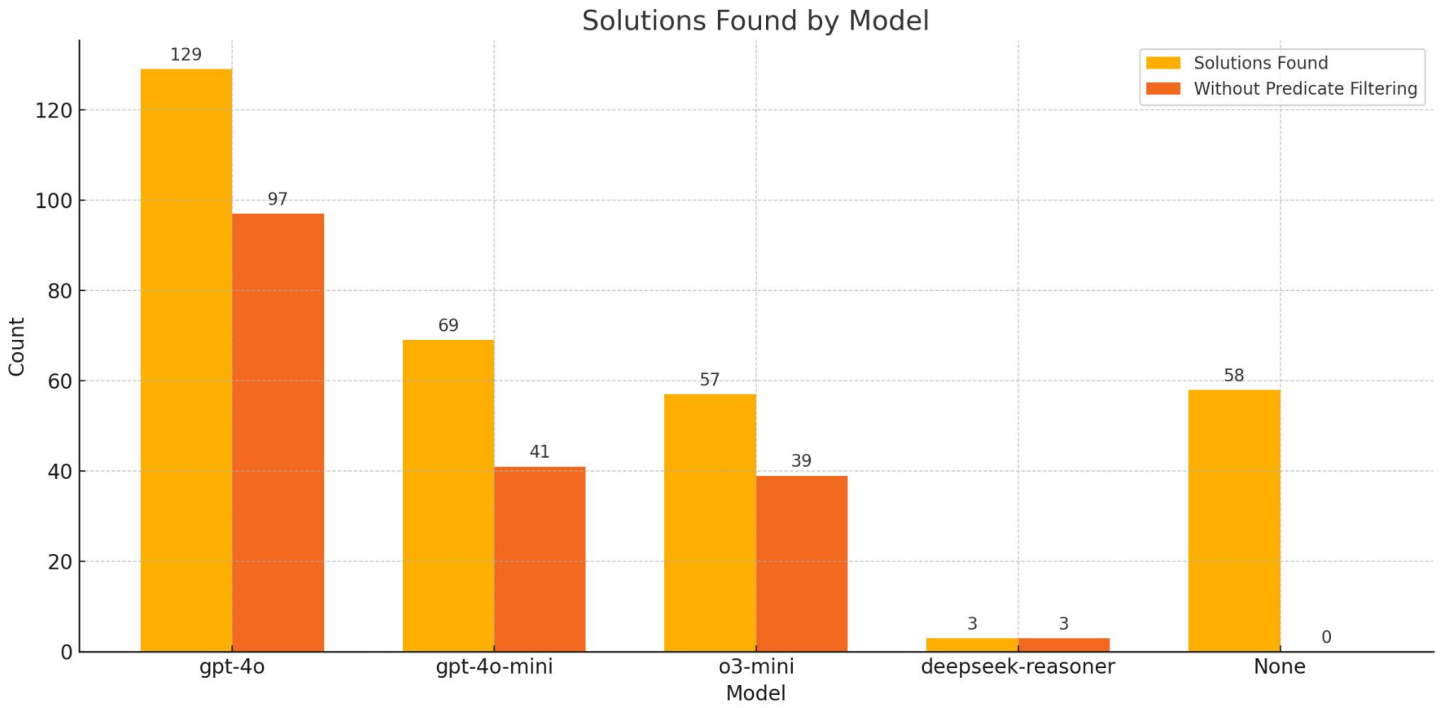
**Total benchmarks:** 316

**Solutions found:** 316

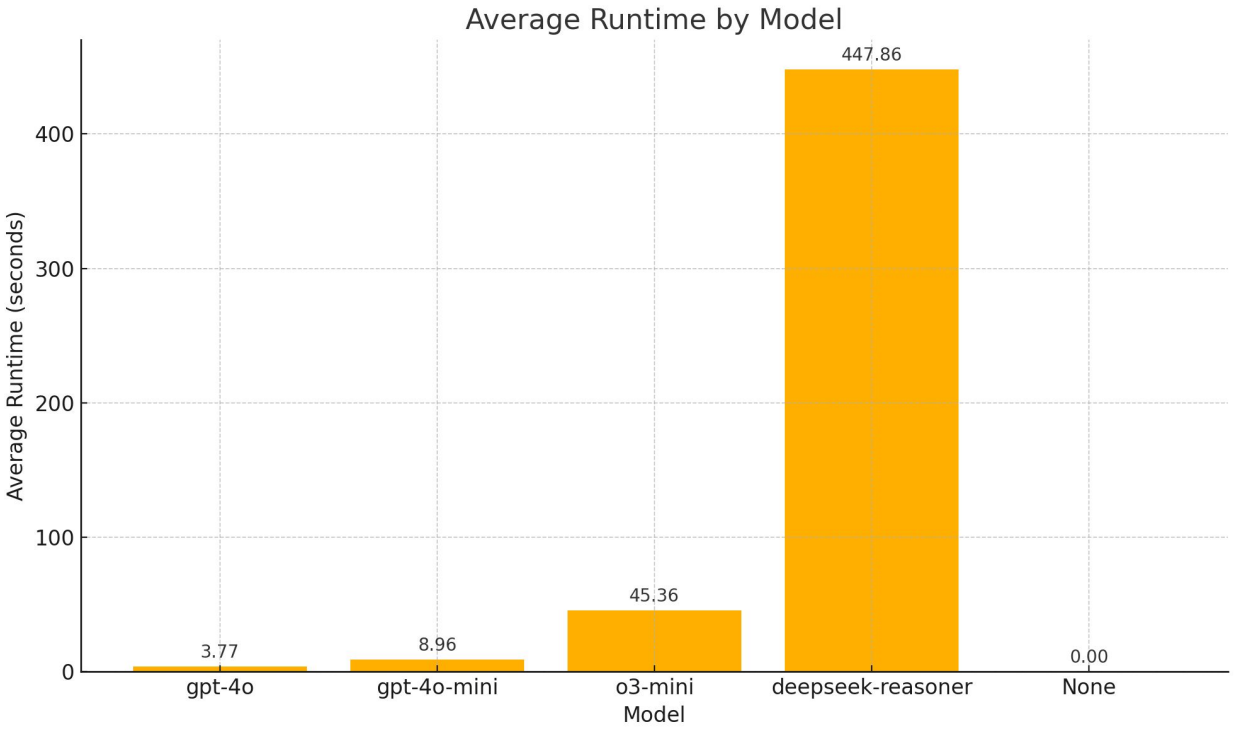
**Success rate:** **100.00%**

**Average runtime:** 28.85 seconds

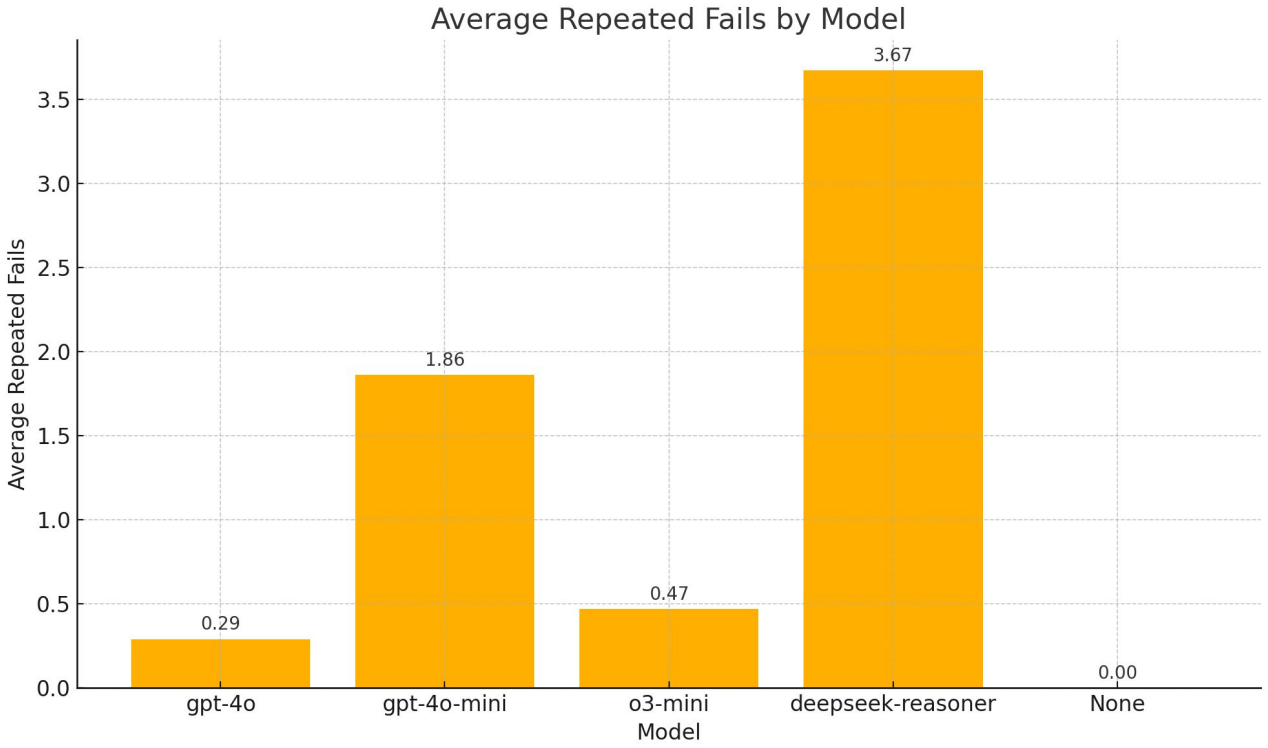
# Results


Solutions Found by Model
Solutions Found
Without Predicate Filtering
Count
0
20
40
60
80
100
120
129
97
69
41
57
39
3
3
58
0
gpt-4o
gpt-4o-mini
o3-mini
deepseek-reasoner
None
Model

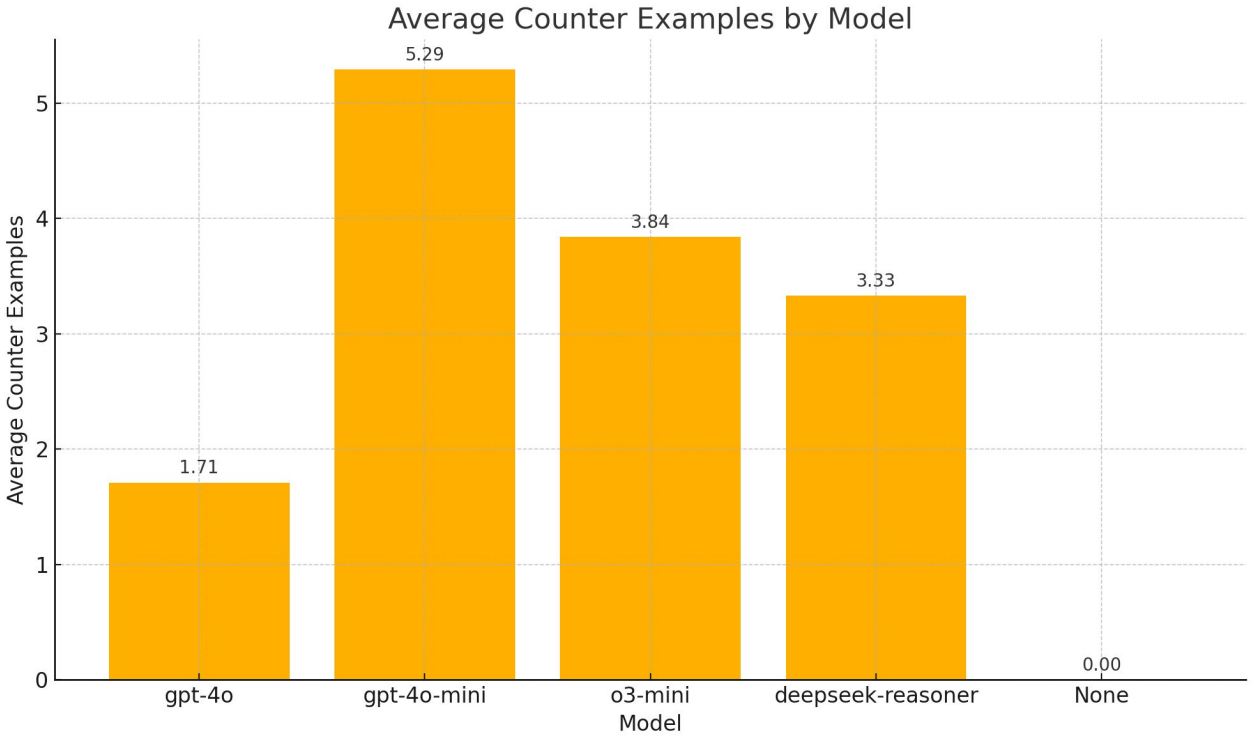
# Results


Average Runtime by Model
Average Runtime (seconds)
0
100
200
300
400
3.77
8.96
45.36
447.86
0.00
gpt-4o
gpt-4o-mini
o3-mini
deepseek-reasoner
None
Model

# Results


Average Repeated Fails by Model
Average Repeated Fails
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.29
1.86
0.47
3.67
0.00
gpt-4o
gpt-4o-mini
o3-mini
deepseek-reasoner
None
Model

# Results


Average Counter Examples by Model
Average Counter Examples
0
1
2
3
4
5
1.71
5.29
3.84
3.33
0.00
gpt-4o
gpt-4o-mini
o3-mini
deepseek-reasoner
None
Model

# Execution Limitations

- OpenAI Tier 1 rate limit
- Slow ESBMC and Z3 execution

Centro de Informática
UFPE

Universidade Federal de Pernambuco

cin.ufpe.br

# Repository

https://github.com/almirmcunhajr/LaM4Inv