The 9th International Conference in Deep Learning in Computational Physics
July, 2-4, 2025 SINP MSU, Moscow, Russia

The creation of reasonable robot control behavior in the form of executable code

(Формирование обоснованной стратегии управления роботом в виде исполняемого кода)

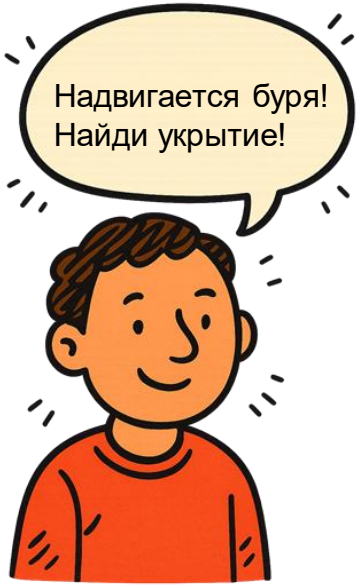
Skorokhodov Maksim [1,2], Latalin Vladislav [2], Rybka Roman [2], Sboev Alexander [1,2]

[1] National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russia
[2] National Research Center "Kurchatov Institute", Moscow, Russia

Скороходов Максим

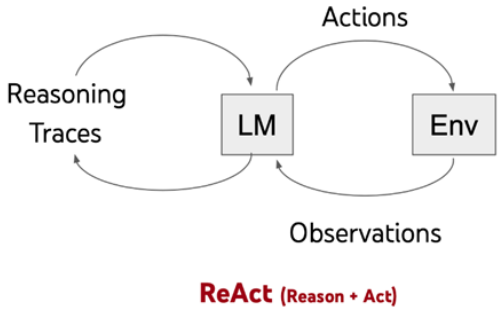
Мотивация

Автономное управление роботами становится особенно важным в условиях, где невозможно обеспечить постоянный контроль со стороны человека. В таких ситуациях робот должен уметь самостоятельно интерпретировать высокоуровневые инструкции. Возникает необходимость в системах, способных формулировать цели, планировать и адаптироваться к среде без внешней помощи.

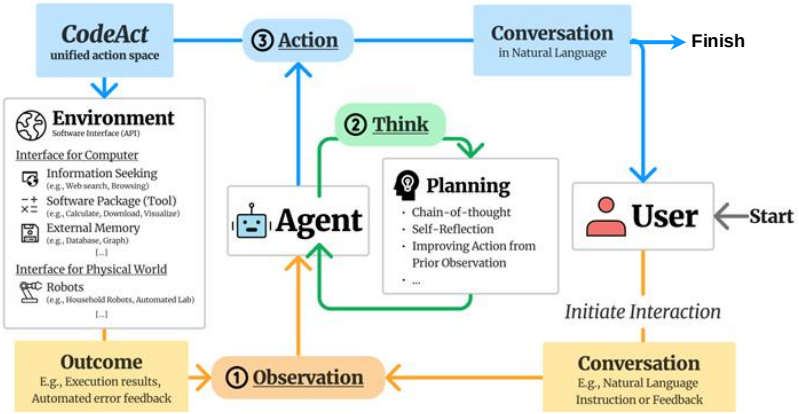


Связанные работы

Агентные подходы по взаимодействию с Большими Языковыми Моделями (LLM) для решения широкого спектра задач.



Работа ReAct объединяет способности LLM к рассуждению и выполнению действий в едином интерактивном цикле между средой Env и языковой моделью на основе наблюдений.



Работа CodeAct предлагает подход к созданию LLM-агентов, который использует исполняемый Python код в качестве универсального пространства действий.

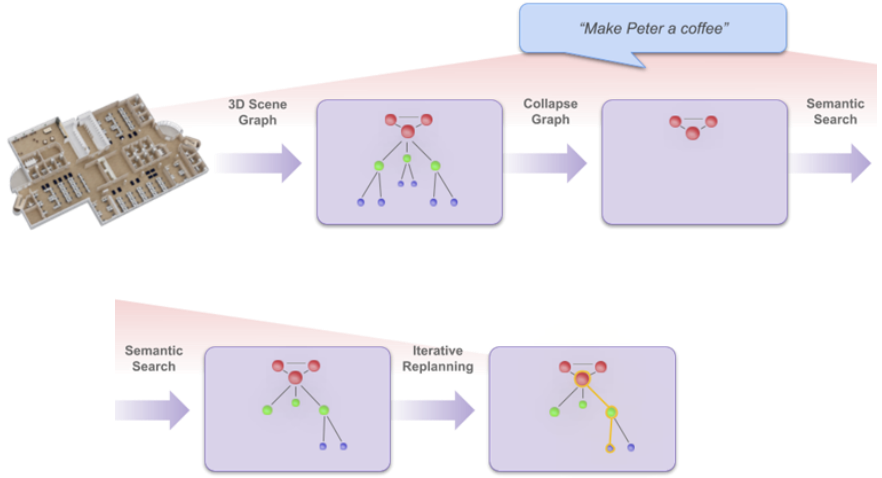
https://react-lm.github.io/

https://arxiv.org/abs/2402.01030

Связанные работы

Применение Больших Языковых Моделей (LLM) для управления роботами на основе метода Prompt Engineering и агентных систем.

Работа SayCan решает задачу составления плана последовательных действий для робота на основе Prompt Engineering.



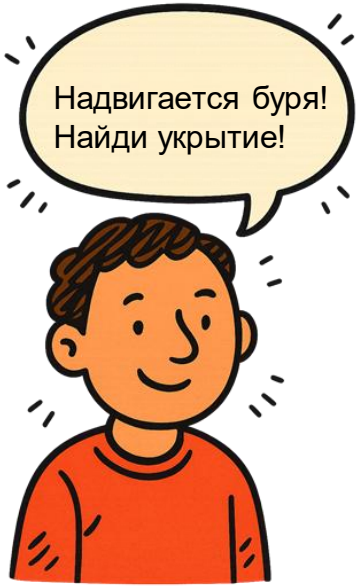
Работа SayPlan предлагает агентную систему по планированию действий робота с учетом информации о внешнем мире.

https://say-can.github.io/

https://sayplan.github.io/

Цель

Разработать систему управления поведением робота с использованием Больших Языковых Моделей (LLM) на основе генерации исполняемого кода и внешнего источника информации об окружающем мире.



Инструкция от человека

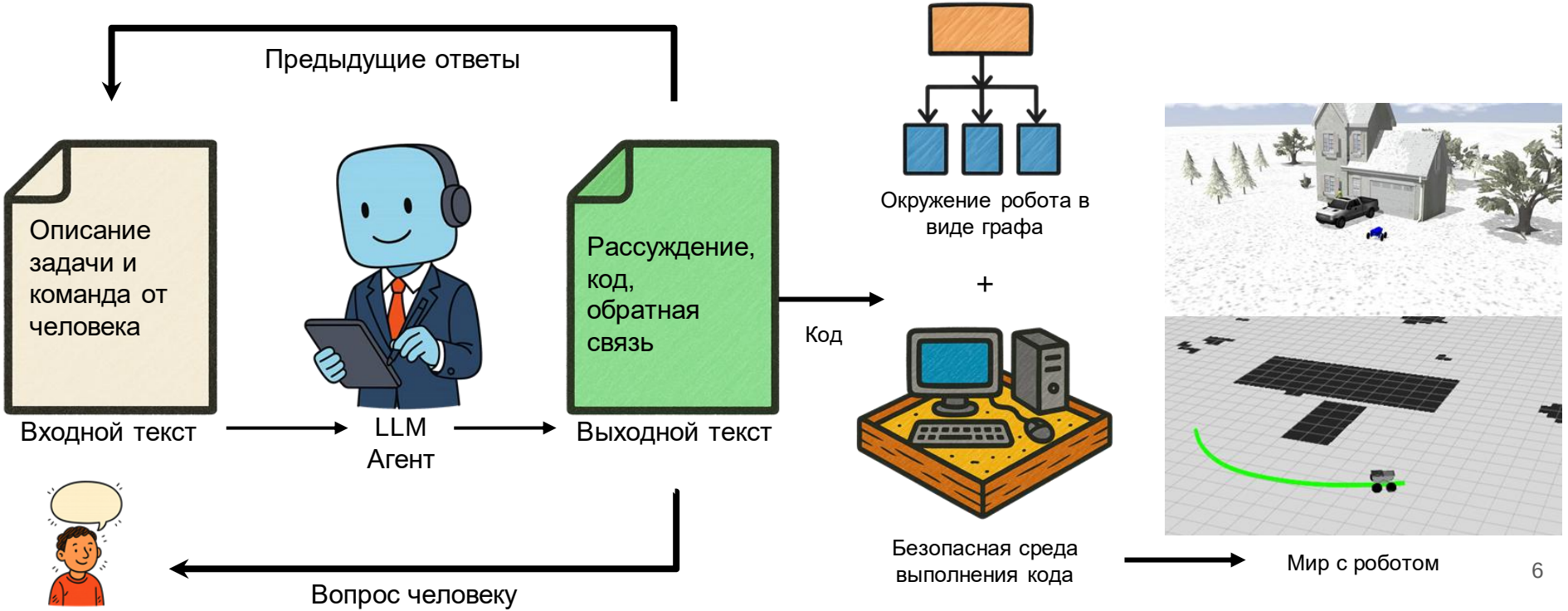
LLM пишет код

Выполнение кода

Мир с роботом

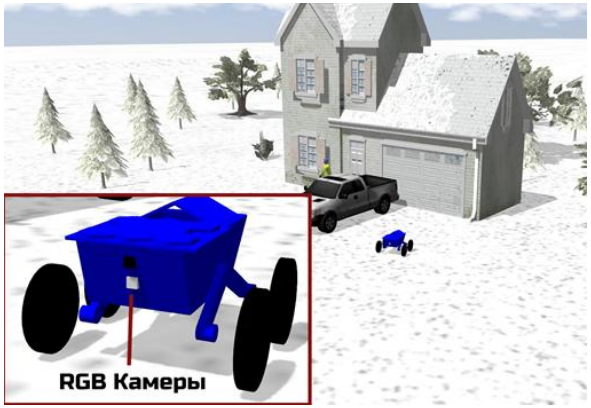
Задачи

1. Определение формата входных и выходных данных языковой модели.
2. Построение системы взаимодействия между человеком, языковой моделью и средой выполнения кода.
3. Оценка эффективности разработанной системы на тестовой выборке.



Мир с роботом

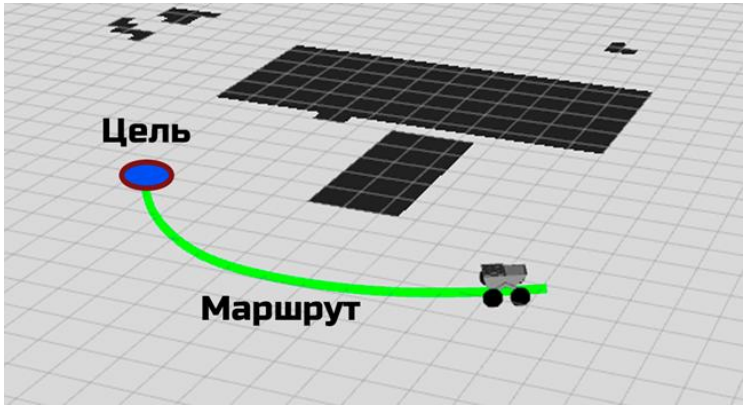
В качестве экспериментальной платформы использовался четырехколесный робот, способный перемещаться в трехмерной среде с различными объектами (деревьями, домами и т. д.).



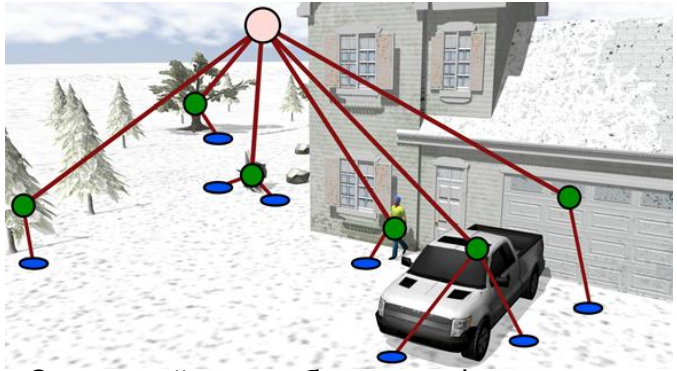
Возможности робота:

- движение по маршруту планировщика

- фотосъемка местности



Движение по локальному планировщику



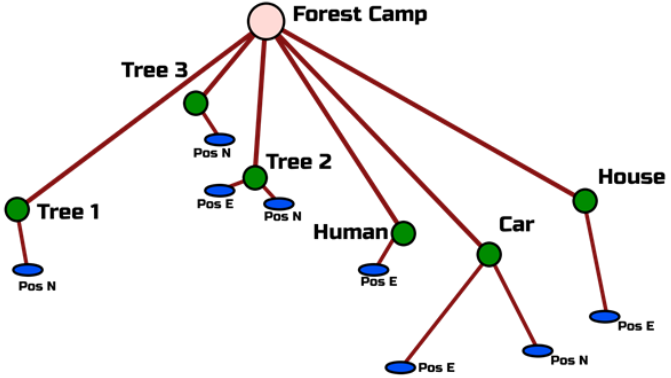
Статичный мир - объекты зафиксированы

Иерархический граф

Уровень 1: Область

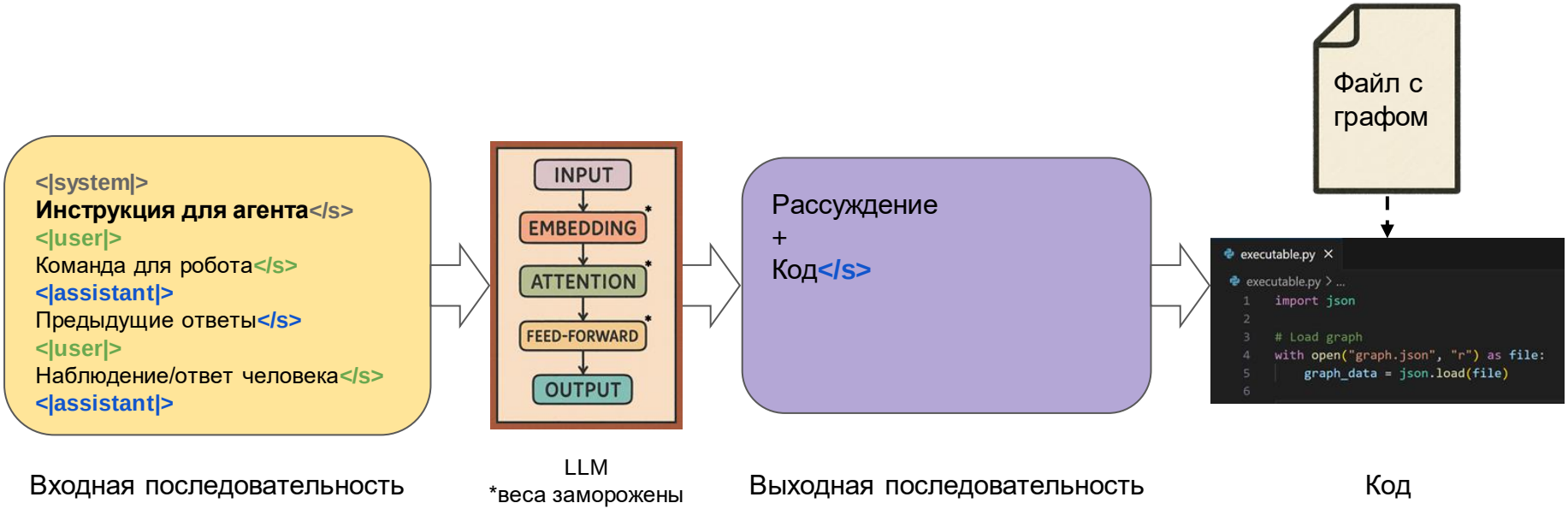
Уровень 2: Объект

Уровень 3: Позиция



Метод

Prompt Engineering (PE) используется как основной метод взаимодействия с LLM в разработанной системе управления поведением робота. Этот метод основан на способности LLM выполнять задачи из разных доменов без изменения параметров, используя только описание из входной последовательности. В отличие от обучения с учителем, PE не требует дополнительной настройки весов модели.



Тестовая выборка

Для проверки способности LLM управлять поведением робота при помощи исполняемого кода был составлен тестовый набор данных, состоящий из 60 поведенческих команд, которые можно разделить на 3 класса:

1

Прямые инструкции:

“Сфотографируй 3 дерева в лесу.”

“Патрулируй между домами.”

“Встань напротив дальнего дома с северной стороны.”

2

Абстрактные инструкции:

“Надвигается буря! Найди надежное укрытие!”

“Проведи разведку местности.”

“Стало темно, проверь в чем дело.”

3

Внепространственные инструкции:

“Сруби дерево.”

“Лети на Луну.”

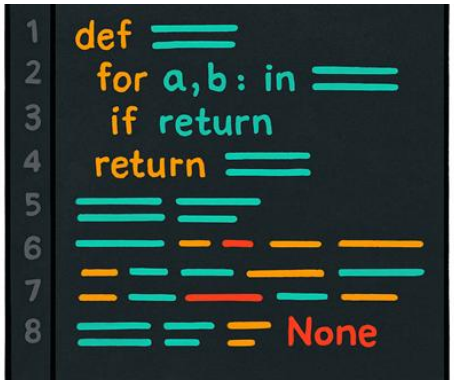
“Укради у человека одежду.”

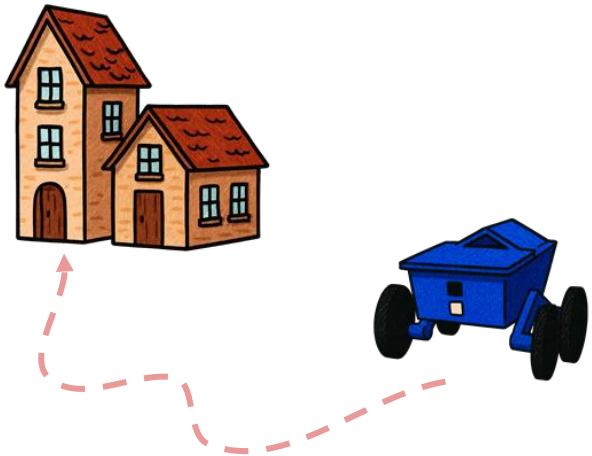
Эксперимент

При помощи подготовленной выборки были проведены **тесты на различных моделях** - GPT-4.1-mini, Deepseek Chat, Deepseek R1, Qwen2.5 Coder. Разнообразие моделей по архитектуре, типу обучения и количеству параметров позволяет оценить их эффективность в задачах генерации управляющего кода.

Оценка правильно выполненных инструкций осуществлялась **вручную людьми** по следующим критериям:

- соблюдены логические выводы и последовательность действий шаг за шагом;
- созданный моделями исполняемый код приводит к успешному выполнению инструкции;
- отсутствует недостоверная информация о входном контексте.





Результаты

В качестве метрики оценки используется показатель успешности (Success Rate, SR) — отношение правильно выполненных инструкций к общему количеству инструкций в тестовой выборке.

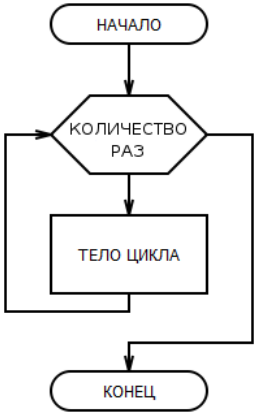
$$\mathrm{SR} = \frac{N_{correct}}{N} \cdot 100\%$$

	Прямые, %	Абстрактные, %	Внепространственные, %	**Общий, %**
Deepseek Chat	55	60	30	48
Deepseek R1	90	65	60	**72**
GPT 4.1 mini	55	75	30	53
Qwen2.5-Coder-14B-Instruct	25	30	20	25

Преимущества предлагаемого подхода

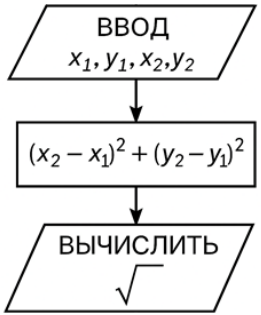
Циклические действия

"Сфотографируй 3 дерева в лесу"



Математические функции

"Встань напротив дальнего дома с северной стороны"



Исправление ошибок

Для выполнения инструкции предлагаю следующий код:
`<code>...</code>`

Ошибка выполнения кода.
Причина: . . .

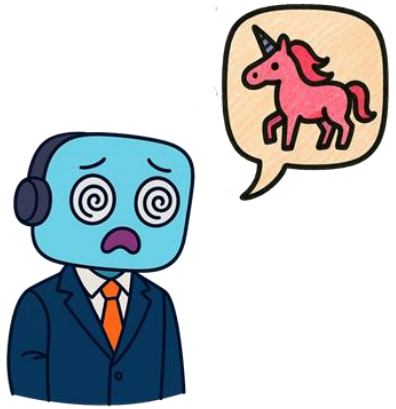
Вот исправленный вариант кода
`<code>...</code>`

Ошибки

- Модель **не может исправить свой код**, например, случай чтения графа из файла, что приводит к появлению избыточных конструкций и как следствие к новым ошибкам в выборках **прямых и абстрактных инструкций**.

- Модель **пытается выполнить код** для команд из выборки **внепространственных инструкций**, однако, ожидается, что будет сформировано сообщение со статусом **error** без кода.

- **Галлюцинации** при генерации ответа – модель не следует заданным во входной последовательности правилам.

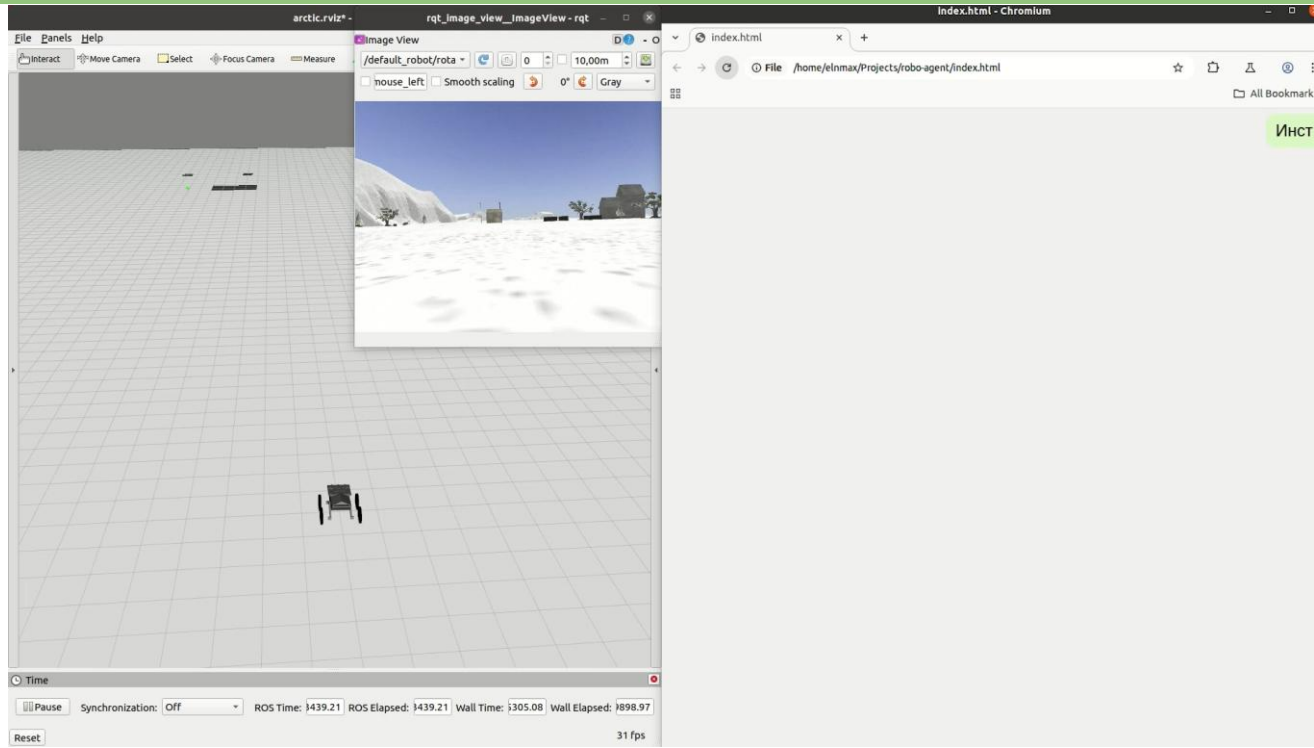


Заключение

Показано, что **генерация управляющих инструкций** непосредственно в виде исполняемого кода **повышает гибкость** управления роботом и **снижает риск** галлюцинаций, в частности, при выполнении арифметических операций.

Разработана **агентная система** на основе LLM, которая **интерпретирует инструкции** по управлению роботом в виде фрагментов кода.

Предложенный подход позволяет перенести **выполнение сложной логики** взаимодействия робота с окружением из ответов LLM в среду выполнения кода, повышая **надёжность** и **предсказуемость поведения** системы.



Будущая работа

- Поиск **оптимальной входной последовательности** для уменьшения галлюцинаций при составлении ответа моделью.

- Использование **генерации с дополненной выборкой** (Retrieval Augmented Generation, RAG) и **мультиагентных подходов** для автоматического составления кодовой базы и разделения языковых моделей по ролям, что позволит повысить качество генерируемого кода и уменьшит количество избыточных конструкций.

- Исследование **мультимодальных данных и моделей** на основе визуальной, звуковой и текстовой информации.

- Подготовка инструктивного набора данных с логическими текстовыми рассуждениями и исполняемым кодом для **настройки локальной модели** с небольшим количеством параметров **для быстрых ответов** и низких ресурсозатрат.

Спасибо за внимание!

skormax43@gmail.com

sag111@mail.ru

Telegram

Анализ эксперимента

Циклические действия

```
instruction = "Сфотографируй 3 дерева в лесу"
...
# Выбор трех деревьев
for tree_id, tree_coords in forest_trees[:3]:
    # Поиск доступных позиций для фото (уровень 2)
    photo_position = None
    for neighbor in G.neighbors(tree_id):
        if G.nodes[neighbor].get('level') == 2:
            photo_position = G.nodes[neighbor]['coordinates']
            break
        if not photo_position:
            print(f"Warning: No position found for {tree_id}")
            continue

    # Последовательность действий
    move(photo_position[0], photo_position[1])
    rotate(tree_coords[0], tree_coords[1])
    photo()
```

Математические функции

```
instruction = "Встань напротив дальнего дома с северной стороны"
...


def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

# Поиск самого дальнего дома
farthest_house = None
max_dist = -1
for house in houses:
    coords = house['coordinates']
    dist = distance(robot_pos[0], robot_pos[1], coords[0], coords[1])
    if dist > max_dist:
        max_dist = dist
        farthest_house = house
```

Исправление ошибок

```
import json
import networkx as nx

# Загрузка графа
with open('ml_data/datasets/graph.json', 'r') as f:
    graph_data = json.load(f)

G = nx.Graph()
for node in graph_data['nodes']:
    G.add_node(node[0], **node[1])

G.add_edges_from(graph_data['edges'])
```

Code execution failed at line 'for node in graph_data['nodes']:
G.add_node(node[0], **node[1])' due to: TypeError: keywords must be strings

```
G = nx.Graph()
# Исправление: добавляем узлы через nodes_from с кортежами (node_id,
attributes)
G.add_nodes_from((node[0], node[1]) for node in graph_data['nodes'])

G.add_edges_from(graph_data['edges'])
```

Инструкция для агента

Описание роли LLM агента

Ты полезный **агент для планирования действий робота** на графе.
Твоя задача использовать **язык программирования Python** для выполнения инструкций от человека.
Пиши простой и читаемый код, избегай рекурсий, бесконечных циклов и сложных конструкций.

Дополнительные функции для интерпретатора кода - возможности робота

move(x, y)
rotate(x, y)
photo()
robot_position()

Дополнительные функции интерактивной среды:
def ***move(x: float, y: float)*** return info: str
Описание: переместить робота в заданную позицию, ТОЛЬКО "level": 2 узлы графа

def ***rotate(x: float, y: float)*** return info: str
Описание: повернуть робота к указанному объекту, ТОЛЬКО "level": 1 узлы графа

def ***photo()*** return None
Описание: сфотографировать объект, ТОЛЬКО "level": 1 узлы графа

def ***robot_position()*** return [x: float, y: float]
Описание: получить текущую позицию робота

Возможности робота:
move - переместиться в точку
rotate - повернуться к объекту
photo - сфотографировать объект

Описание внешней среды - иерархического графа

Файл с графом graph.json
Структура графа
Вершины графа

Описание графа:
допустимые библиотеки для работы с графом: json, networkx
файл с графом - ml_data/datasets/graph.json, открывать с параметром 'r'
Граф представляет собой иерархическую структуру с 3 уровнями
0 уровень - описывает области интереса, которые содержат группы объектов
1 уровень - описывает объект и координаты его центра
2 уровень - описывает позиции рядом с объектом (стороны света), в которые может переместиться робот}
Список областей: village, forest camp, forest thicket, dense forest, rock field
Список объектов: house_, tree_, broken_tree_, barrier_, human_, lamp_post_, rock_, car_
Список позиций: N - север, E - восток, S - юг, W - запад
Пример содержания файла с графом:
{"nodes": [["city", {"name": "city", "coordinates": null, "level": 0}],
["tree_042", {"name": "tree_042", "coordinates": [1.0, 2.0], "level": 1}],
["tree_042_N", {"name": "tree_042_N", "coordinates": [-1.0, 2.0], "level": 2}], ...],
"edges": [["city", "tree_042"], ["tree_042", "tree_042_N"], ...]}

Желаемый формат ответа

Использование html-like тегов
<thought>...</thought>
<code>...</code>
<status>...</status>

Статусы:
execute - вызвать интерактивную среду для выполнения кода
error - сообщение об ошибке если невозможно выполнить инструкцию
done - завершение выполнения инструкции

Формат ответа:
<thought>Последовательное рассуждение шаг за шагом для выполнения инструкции**</thought>**
<code>Исполняемый код на языке программирования Python**</code>**
<status>Статус**</status>**