

Отчет о проделанной работе на тему 18: Использование LLM для управления агентами в MMORPG

Стефан Ребриков
НИУ ВШЭ
Москва, Россия
robstef85@gmail.com

Виктор Волков
ТГУ
Томск, Россия
viktor.volkov.mailbox@gmail.com

Александра Кулешова
МГТУ им. Н.Э. Баумана
Москва, Россия
sun-kulich@yandex.ru

Олег Саутенков
Сколтех
Москва, Россия
oleg.sautenkov@skoltech.ru

Роман Халиков
Сколтех
Москва, Россия
rom.khalikov@gmail.com

Пётр Анохин
AIRI
Москва, Россия
anokhin@airi.com

I. ВВЕДЕНИЕ

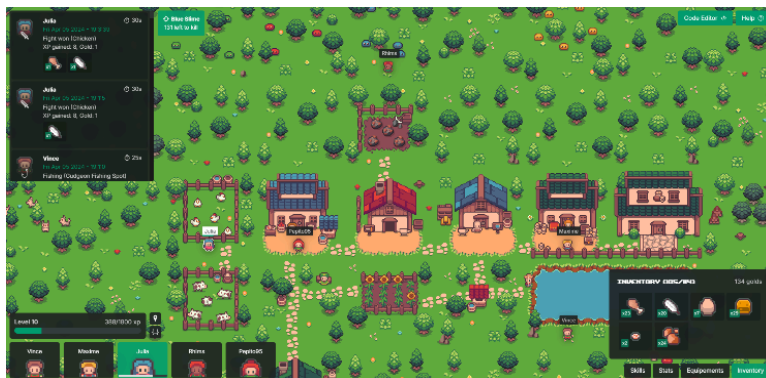
За последние несколько лет развития больших языковых моделей стала понятна их сила при решении различных задач, связанных не только с текстовыми данными, но и при выполнении задач, связанных с планированием, рассуждениями, принятием решений. Это понимание дало толчок к развитию языковых моделей, как агентов. Этот термин из сферы обучения с подкреплением подразумевает некую обучающуюся систему, которая, взаимодействуя с окружающим миром и, получая обратную связь, учится выполнять поставленные задачи. Агенты на основе больших языковых моделей точно так же способны, применяя текстовые данные, выполнять задачи и учиться на основе собственных действий. Более того, было показано, что, создавая целые системы таких агентов (мультиагентные системы), возможно с большей эффективностью решать более сложные и многокомпонентные задачи, например, написание кода, прохождение игры или симуляция некоего сообщества [1]. Такие системы активно исследуются с 2023 года. Агенты в системах могут обладать различными инструкциями и соответствующей ролью, взаимодействовать не только с окружением, но и между собой, решать вопросы как совместно, так и в соревновательном режиме.

В этом отчете мы демонстрируем исследование на тему возможности применения мультиагентной системы на основе большой языковой модели Llama 3.1 на 70 миллиардов параметров для прохождения онлайн игры Artifacts ММО. Нам удалось продумать общую архитектуру проекта и составить базовые компоненты системы для успешного старта игры.

II. ЗАДАЧА

A. Описание игры

Artifacts ММО представляет собой небольшую 2D игру с видом сверху (Изображение 1), где игроку дают в управление до 5 персонажей, каждый из которых имеет собственные характеристики, развивающиеся по ходу игры. Ключевой особенностью этой игры является то, что управлять персонажами возможно при помощи специально написанной API. При отправке POST/GET запросов через протокол HTTP, возможно отправлять команды для совершения базовых действий для каждого персонажа, просматривать карту, статус персонажа и т.д. Главной целью данной игры является убийство самого сильного монстра на карте. Для выполнения этой цели игроку необходимо повышать уровень персонажей, создавать новые предметы экипировки, выполнять задания и собирать ресурсы. Дизайн игры предполагает, что для выполнения финальной задачи требуется сделать большое количество однотипных действий. Это мотивирует автоматизировать как можно большее число действий.



Изображение 1. Пользовательский интерфейс игры для ручного управления

В. Исследовательская задача

Подобная игра является примером исследовательской среды для мультиагентных систем. Проект предлагает продемонстрировать работоспособность мультиагентной системы на основе доступной бесплатно большой языковой модели Llama 3.1. Требуется придумать архитектуру такой системы, в рамках которой несколько агентов смогут сообща выполнять хотя бы базовые задачи для прогресса в игре, а также продемонстрировать работу хотя бы части архитектуры.

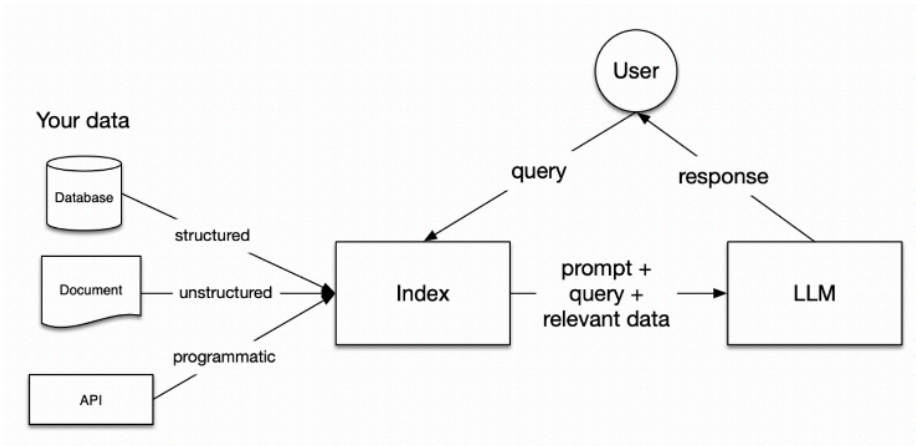
III. Методы

А. Железо и архитектура языковой модели

Как уже упоминалось ранее, большой языковой моделью для решения данной задачи являлась Llama 3.1 с 70 миллиардами параметров (Llama v3.1:70b-instruct-q6_K). Модель была загружена предобученной с использованием библиотеки ollama, использовалась версия модели instruct, более подходящей для ответов на вопросы и решения задач. Для запуска модели использовался сервер университета ИТМО с 4 видеокартами A100.

В. Применение RAG

Для записи и хранения большого числа информации, которую возможно при необходимости давать модели для увеличения контекста, была использована техника Retrieval-Augmented Generation или RAG. Простая архитектура на RAG представлена на изображении 2. Суть RAG заключается в том, чтобы хранить набор эмбедингов, по которым возможно провести поиск в зависимости от некоей метрики расстояния между эмбедингом запроса и набором эмбедингов данных с целью дополнения изначального запроса. В нашей работе мы применяли RAG для того, чтобы находить требующиеся рецепты для производства предметов, а также для получения информации об окружении (получения карты местности).



Изображение 2. Архитектура простейшего RAG

С. Планирование задач

Первая попытка запуска системы заключалась в подаче на вход всей необходимой для решения задачи информации, а именно состояние карты, список рецептов для производства предметов, примеры функций и характеристики монстров. Вся эта информация дополнялась инструкцией в конце, описывающей достаточно крупную, комплексную цель. На выходе LLM путалась в поданном контексте и не могла представить правильный и конкретный план. В результате чего было принято решение сделать многоагентную систему с декомпозицией задачи. Существует несколько подходов для организации планирования [2]. Во-первых большую задачу возможно разбить на несколько маленьких с целью упрощения контекста задач агентов. При декомпозиция-сперва (Decomposition-First) задача делится на простые подзадачи и только потом каждая из задач выполняется. При методах попеременной декомпозиции (Interleaved Decomposition) LLM генерирует подцель, затем эта подцель исполняется, чтобы потом эта процедура повторилась, пока не будет выполнена изначально поставленная цель. Другим подходом является генерация нескольких планов, чтобы выбрать один по некоему критерию. Также возможно организовать специализированный под задачу планировщик – для некоторых задач используют либо специально предобученные планировщики, либо подходящие под специфику задачи алгоритмы планировки. С целью доработки планов существует подход “размышление и критика”. При таком подходе помимо декомпозиции модель так же рассуждает, дает оценку и критику касаясь сгенерированного плана. Для дополнения этих систем возможно применять специализированный RAG или дообучение большой языковой модели. В этой работе был выбран вариант с декомпозицией-сперва, поскольку в данном случае возможно получение плана действий и разделение его на нескольких агентов.

D. Финальная архитектура

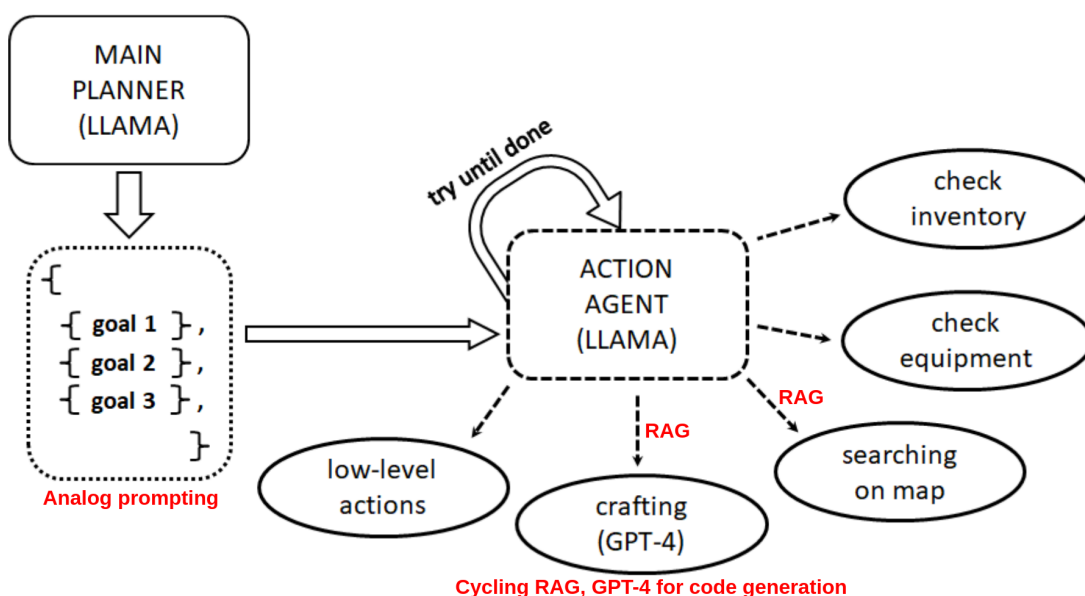
Планирование агента осуществлялось с помощью комплексной архитектуры, указанной на изображении 3. Было введено два агента-планировщика: Main planner или Global Planner декомпозирует задачи на верхнем уровне. Action Agent или Tactic Planner обучался использованию набора функций:

- Проверка инвентаря
- Проверка снаряжения
- Поиск объектов на карте
- Создание предметов
- Общие действия

Main Planner декомпозировал задачи по образу и подобию запроса для LLaма и выдавал структурированные цели.

Action Agent выбирал цель и подбирал наиболее подходящую функцию из пяти, ранее ему указанных. Функции *check inventory*, *check equipment* и *low-level actions* были определены заранее. Для функции поиска объектов на карте был использован RAG, где обновлялся словарь. Для многоступенчатого крафта предметов был использован RAG для получения списка необходимых предметов и затем был осуществлен запрос создать Python-функцию производства предмета для ChatGPT-4, так как она лучше всего работала с генерацией кода.

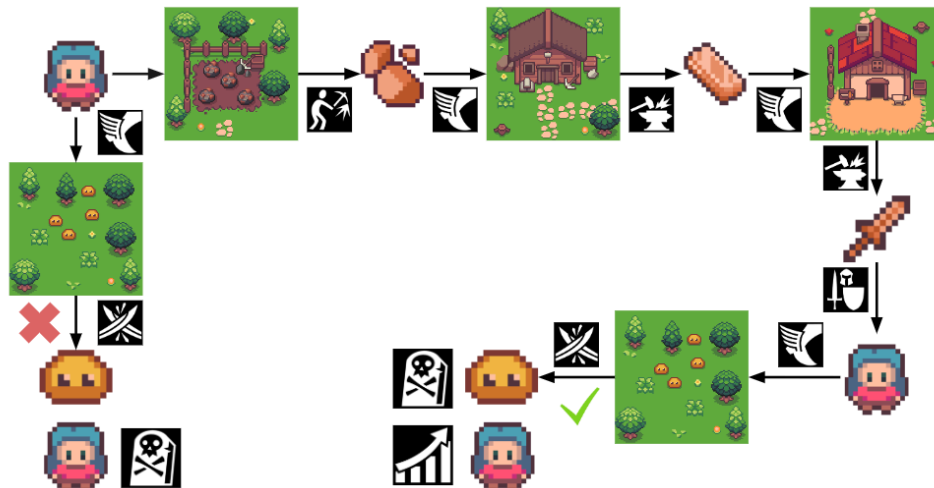
Архитектура LLM-агента v1.0



Изображение 3. Финальный вариант архитектуры

IV. РЕЗУЛЬТАТЫ

В качестве итогового теста получившейся системы модели была задана задача убийства желтого слизня без каких-либо уточняющих инструкций. Эту задачу невозможно было бы сделать без предварительной подготовки и моделью был предложен план, показанный на изображении 4.



Изображение 4. План по убийству желтого слизня

В итоге модель справилась с поставленной задачей и представила подробный декомпозированный план действий (вывод модели можно посмотреть в приложении). Далее по этому план был получен код для выполнения HTTP-запросов и успешно выполнен для управления персонажем. Поставленная задача была выполнена за менее чем 20 минут.

REFERENCES

- [1] Guo, Taicheng, et al. "Large language model based multi-agents: A survey of progress and challenges." *arXiv preprint arXiv:2402.01680* (2024).
- [2] Xu Huang, et al. "Understanding the planning of LLM agents: A survey" *arXiv: <https://arxiv.org/pdf/2402.02716>* (2024)

Log:

```
{
  "main_goal": "Kill Yellow Slime",
  "plan_steps": [
    {
      "sub_goal_1": "Reach Yellow Slime Location",
      "notes_for_sub_goal_1": "Move to location where Yellow Slime is known to spawn"
    },
    {
      "sub_goal_2": "Prepare for Combat",
      "notes_for_sub_goal_2": "Check inventory and equipment, craft or equip necessary items if needed"
    },
    {
      "sub_goal_3": "Engage in Combat with Yellow Slime",
      "notes_for_sub_goal_3": "Use combat abilities to defeat the Yellow Slime"
    }
  ]
}
```

sub_goal_1: Reach Yellow Slime Location

notes_for_sub_goal_1: Move to location where Yellow Slime is known to spawn

Agent response: Locate Yellow Slime on the map of the environment

Extracted object: Yellow Slime

map provided

Agent response: Perform action: move(4, -1)

Action performed: move(4, -1)

Action result: Success

Agent response: Task done

sub_goal_2: Prepare for Combat

notes_for_sub_goal_2: Check inventory and equipment, craft or equip necessary items if needed

Agent response: Perform action: check_inventory()

Action performed: check_inventory()

Action result: [{ 'slot': 1, 'code': '', 'quantity': 0 }, { 'slot': 2, 'code': '', 'quantity': 0 }, { 'slot': 3, 'code': '', 'quantity': 0 }, { 'slot': 4, 'code': '', 'quantity': 0 }, { 'slot': 5, 'code': '', 'quantity': 0 }, { 'slot': 6, 'code': '', 'quantity': 0 }, { 'slot': 7, 'code': '', 'quantity': 0 }, { 'slot': 8, 'code': '', 'quantity': 0 }, { 'slot': 9, 'code': '', 'quantity': 0 }, { 'slot': 10, 'code': '', 'quantity': 0 }, { 'slot': 11, 'code': '', 'quantity': 0 }, { 'slot': 12, 'code': '', 'quantity': 0 }, { 'slot': 13, 'code': '', 'quantity': 0 }, { 'slot': 14, 'code': '', 'quantity': 0 }, { 'slot': 15, 'code': '', 'quantity': 0 }, { 'slot': 16, 'code': '', 'quantity': 0 }, { 'slot': 17, 'code': '', 'quantity': 0 }, { 'slot': 18, 'code': '', 'quantity': 0 }, { 'slot': 19, 'code': '', 'quantity': 0 }, { 'slot': 20, 'code': '', 'quantity': 0 }]

Agent response: Perform action: check_equiped_items()

Action performed: check_equiped_items()

Action result: Equiped items: {'weapon_slot': 'wooden_stick', 'shield_slot': None, 'helmet_slot': None, 'body_armor_slot': None, 'leg_armor_slot': None, 'boots_slot': None, 'ring1_slot': None, 'ring2_slot': None, 'amulet_slot': None, 'artifact1_slot': None, 'artifact2_slot': None, 'artifact3_slot': None, 'consumable1_slot': None, 'consumable1_slot_quantity': None, 'consumable2_slot': None, 'consumable2_slot_quantity': None}

Agent response: Perform action: check_craftable_items()

Action performed: check_craftable_items()

Action result: Items available for crafting: copper_dagger

Agent response: Perform action: craft_item(copper_dagger)

Action performed: craft_item(copper_dagger)

Action result: Copper Dagger crafted successfully!

Agent response: Perform action: equip_item(weapon, copper_dagger)

Action performed: equip_item(weapon, copper_dagger)

Action result: This slot is not empty, empty it, then try equipping item again

Agent response: Perform action: unequip_item(weapon_slot)

Action performed: unequip_item(weapon)

Action result: Item removed from slot

Agent response: Perform action: equip_item(weapon, copper_dagger)

Action performed: equip_item(weapon, copper_dagger)

Action result: Item equipped in slot

Agent response: Task done

sub_goal_3: Engage in Combat with Yellow Slime

notes_for_sub_goal_3: Use combat abilities to defeat the Yellow Slime

Agent response: Perform action: fight()

Action performed: fight()

Action result: Mob not found on this tile

Agent response: Locate Yellow Slime on the map of the environment

Extracted object: Yellow Slime

map provided

Agent response: Perform action: move(4, -1)

Action performed: move(4, -1)

Action result: Success

Agent response: Perform action: fight()

Action performed: fight()

Action result: Fight won

Agent response: Task done