ЛАБОРАТОРНА РОБОТА № 2. ДОСЛІДЖЕННЯ ЕКОСИСТЕМИ LLAMAHUB ТА СТВОРЕННЯ AI-АГЕНТА НА БАЗІ LLAMAINDEX

Мета: Навчитися самостійно досліджувати та аналізувати готові компоненти з репозиторію LlamaHub. Здобути навички інтеграції сторонніх інструментів (Tools) у власні проєкти на базі LlamaIndex. Розвинути креативне мислення для проєктування унікальних та практично корисних AI-агентів. Закріпити знання про архітектуру FunctionAgent, важливість системних промптів та автономні робочі процеси..

Теоретичні відомості

AI-агент – це автономна система, що використовує велику мовну модель (LLM) як центральний «мозок» для прийняття рішень та виконання дій. Ефективність агента залежить не тільки від потужності LLM, але й від його здатності взаємодіяти із зовнішнім світом. Цю взаємодію забезпечують **інструменти (Tools)** – функції, які агент може викликати для доступу до даних, сервісів або виконання операцій, що недоступні самій LLM (наприклад, пошук в інтернеті, робота з файлами, доступ до API).

LlamaIndex надає потужний фреймворк для створення таких агентів. FunctionAgent – це сучасний тип агента, що покладається на вбудовану в LLM здатність до «виклику функцій» (tool calling). Агент отримує від користувача високорівневу мету, самостійно розбиває її на послідовність кроків («думка → дія») і викликає відповідні інструменти, доки не досягне результату. Поведінка, роль та план дій агента визначаються через **системний промпт** (system prompt), який є найважливішим елементом конфігурації.

LlamaHub (llamahub.ai) – це централізований репозиторій, що містить сотні готових інструментів, конекторів даних та інших компонентів для LlamaIndex. Використання готових інструментів значно прискорює розробку, дозволяючи швидко інтегрувати агента з популярними сервісами (Google, Wikipedia, Arxiv, фінансові API тощо). Кожен інструмент зазвичай представлений у вигляді ToolSpec – класу, що об’єднує групу пов’язаних функцій.

ЗАВДАННЯ ДЛЯ ВСІХ ВАРІАНТІВ

Крок 1. Дослідження LlamaHub (Творчий етап)

1. Відвідайте сайт **LlamaHub** (llamahub.ai).
2. Перейдіть до розділу «Agent Tools».
3. **Проаналізуйте доступні інструменти.**

   Зверніть увагу на їхні описи, приклади використання та вимоги (наприклад, наявність API-ключа).

4. **Оберіть ОДИН інструмент (або групу інструментів з одного ToolSpec), який вас найбільше зацікавив.**

   Критерії вибору

   Особистий інтерес, потенційна корисність, можливість використання без платного API (або з безкоштовним тарифом).

   Приклади цікавих інструментів

   WikipediaToolSpec, ArxivToolSpec, OpenWeatherMapToolSpec, YahooFinanceToolSpec, RequestsToolSpec, DuckDuckGoSearchToolSpec.

5. **Придумайте та опишіть ідею для AI-агента, який би використовував обраний інструмент.**

   - *Дайте агенту назву (напр., «AI-Планувальник Погоди», «Науковий Асистент»).*
   - *Сформулюйте для нього конкретну, високорівневу мету, яка демонструє можливості інструмента.*
   - *Опишіть, як саме агент буде використовувати інструмент для досягнення мети.*

Крок 2. Технічна реалізація

1. **Налаштуйте середовище.**

Створіть новий Python-проєкт. Встановіть llama-index-core, llama-index-llms-ollama та пакет для обраного вами інструменту (наприклад, pip install llama-index-tools-arxiv).

2. **Напишіть код для AI-агента. Ваш скрипт повинен містити:**

- *Ініціалізацію локальної LLM, що підтримує tool calling (qwen3:8b або іншої).*
- *Ініціалізацію обраного інструмента з LlamaHub.*
- *Створення кастомних інструментів (наприклад, для запису у файл), якщо це потрібно для вашої ідеї.*
- *Створення FunctionAgent з детальним системним промптом, який пояснює агенту його роль, мету та як користуватися наданими інструментами, включаючи шаблон для фінального звіту.*
- *Формулювання конкретного завдання (task), яке агент має виконати автономно.*
- *Запуск агента та вивід результату. Кінцевим результатом роботи агента має бути артефакт (наприклад, згенерований текстовий файл).*

Крок 3. Підготовка звіту

Оформіть звіт за наступною структурою:

1. *Тема, мета роботи.*
2. *Обраний інструмент з LlamaHub: назва та обґрунтування вибору.*
3. *Опис ідеї та завдання для агента: назва агента, його мета та сценарій використання.*
4. *Хід роботи: повний код скрипта з коментарями та описом ключових частин.*
5. *Результати виконання: скріншот консолі з логами роботи агента (verbose=True) та вміст фінального артефакту (згенерованого файлу).*
6. *Висновки: що вдалося реалізувати, з якими труднощами зіткнулися, та як можна покращити агента.*

ХІД РОБОТИ

Обраний інструмент з LlamaHub

Назва: *ArxivToolSpec (ArXiv Search Tool)*

Чому обрано: *цей інструмент надає прямий доступ до найбільшого відкритого архіву наукових статей Arxiv.org і не вимагає API-ключа, що дозволяє створити корисного асистента для студентів та дослідників для швидкого пошуку актуальної наукової інформації, що є дуже релевантним для освітнього процесу.*

Опис ідеї та завдання для агента

Назва агента: *«Науковий Дайджест»*

Мета: *агент повинен автономно знаходити на Arxiv найсвіжіші статті за заданою темою, витягувати ключову інформацію (назву, авторів, анотацію, посилання) та форматувати її у зручний для читання дайджест, який зберігається у Markdown-файл.*

Завдання для демонстрації: *«Знайди 2 найновіші статті на Arxiv за темою 'Multimodal Large Language Models'. Створи детальний звіт з їх назвами, авторами, анотаціями та посиланнями. Збережи звіт у файл 'multimodal_llm_digest.md'».*

1. **Налаштування середовища та підготовка інструментів.**

Спочатку встановлюємо необхідні бібліотеки. Далі в коді ініціалізуємо LLM, завантажуємо *ArxivToolSpec* з LlamaHub та створюємо власний простий інструмент write_report для запису фінального звіту у файл.

```
import asyncio
import os
from llama_index.llms.ollama import Ollama
from llama_index.core.agent.workflow import FunctionAgent
from llama_index.tools.arxiv import ArxivToolSpec
from llama_index.core.tools import FunctionTool
```

```
print("Ініціалізація LLM (qwen3:8b)...")
llm = Ollama(model="qwen3:8b", request_timeout=300.0)

print("Створення інструментів...")
arxiv_spec = ArxivToolSpec()

def write_report(filename: str, content: str) -> str:
    """Записує фінальний звіт у Markdown файл."""
    try:
        with open(filename, 'w', encoding='utf-8') as f:
            f.write(content)
        return f"Звіт успішно збережено у файл {filename}"
    except Exception as e:
        return f"Помилка при збереженні звіту: {e}"

all_tools = arxiv_spec.to_tool_list() +
[FunctionTool.from_defaults(fn=write_report)]
```

На цьому кроці ми готуємо «сенсори та маніпулятори» для нашого агента. *ArxivToolSpec* дозволить йому «бачити» науковий світ, а *write_report* – створювати фізичний результат своєї роботи.

2. **Створення та конфігурація агента.**

Створюємо екземпляр *FunctionAgent*. Ключовим елементом є деталізований системний промпт, який не лише визначає роль агента, але й надає чіткий шаблон для форматування фінального звіту, що є критично важливим для отримання якісного та структурованого результату.

```
print("Створення агента 'Науковий Дайджест'...")
system_prompt = """
Ти - AI-асистент "Науковий Дайджест". Твоя задача - шукати наукові статті на
Arxiv.org та створювати структурований звіт.

Твій план дій:
1.  **Пошук:** Використай інструмент `arxiv_search` для пошуку статей за
ключовими словами.
2.  **Аналіз:** Для кожної знайденої статті уважно витягни наступну
інформацію:
    *   Назва статті (Title)
    *   Список авторів (Authors)
    *   Повна анотація (Abstract/Summary)
    *   Посилання на статтю (URL)
3.  **Формування звіту:** Створи звіт у форматі Markdown. Дотримуйся **дуже
чітко** наступного шаблону для кожної статті:
```

```
    ---
    ### **Назва статті**
    **Автори:** *Ім'я Автора 1, Ім'я Автора 2, ...*
    **Посилання:** [Читати на Arxiv](URL_статті)

    **Анотація:**
    > Повний текст анотації...
    ---

4.  **Збереження:** Використай інструмент `write_report`, щоб зберегти фінальний звіт у файл.
"""

agent = FunctionAgent(
    tools=all_tools,
    llm=llm,
    system_prompt=system_prompt,
    verbose=True
)
```

Системний промпт виступає як технічне завдання для LLM. Чим він детальніший, тим менше у моделі «свободи» для неправильної інтерпретації завдання. Включення шаблону виводу – найкраща практика для задач генерації структурованого контенту.

3. **Запуск агента та перевірка результату.**

Формулюємо завдання для агента та запускаємо його. Після виконання перевіряємо, чи був створений файл і який його вміст.

```
async def main():
    task = (
        "Знайди 2 найновіші статті на Arxiv за темою 'Multimodal Large Language Models'. "
        "Створи детальний звіт з їх назвами, авторами, анотаціями та посиланнями. "
        "Збережи звіт у файл 'multimodal_llm_digest.md'."
    )

    print(f"\n🚀 Запускаю агента із завданням: '{task}'\n")
    response = await agent.run(user_msg=task)

    print("\n✅ Завдання виконано!")
    print(f"Фінальна відповідь агента: {response}")
```

```
    report_path = 'multimodal_llm_digest.md'
    if os.path.exists(report_path):
        print(f"\nЗвіт знайдено в '{report_path}'. Його вміст:")
        with open(report_path, 'r', encoding='utf-8') as f:
            print("---")
            print(f.read().strip())
            print("---")
    else:
        print(f"Звітний файл '{report_path}' не було створено.")

if __name__ == "__main__":
    asyncio.run(main())
```

Завдання є єдиною точкою входу для користувача. Далі агент працює повністю автономно, слідуючи інструкціям зі свого системного промпту, доки не досягне фінальної мети – створення звіту.

ПРИКЛАД РОБОТИ АГЕНТА

```
✅ Завдання виконано!
Фінальна відповідь агента: <think>
</think>

Звіт про дві найновіші статті на тему "Multimodal Large Language Models" збережено у файлі **multimodal_llm

---

### **LEAML: Label-Efficient Adaptation to Out-of-Distribution Visual Tasks for Multimodal Large Language M
**Автори:** *Chen Zhang, Yaxin Liu, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yux
**Посилання:** [Читати на Arxiv](http://arxiv.org/pdf/2510.03232v1)

**Анотація:**
> Multimodal Large Language Models (MLLMs) have achieved strong performance on general visual benchmarks bu

---

### **Reward Models are Metrics in a Trench Coat**
**Автори:** *Chen Zhang, Yaxin Liu, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yux
**Посилання:** [Читати на Arxiv](http://arxiv.org/pdf/2510.03231v1)

**Анотація:**
> The emergence of reinforcement learning in post-training of large language models has sparked significant

---

Файл готовий для подальшого використання. Якщо потрібно, я можу допомогти з аналізом або відкриттям звіту.
```

LEAML: Label-Efficient Adaptation to Out-of-Distribution Visual Tasks for Multimodal Large Language Models
Автори: *Chen Zhang, Yaxin Liu, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng*
Посилання: [Читати на Arxiv](http://arxiv.org/pdf/2510.03232v1)

Анотація:
> Multimodal Large Language Models (MLLMs) have achieved strong performance on general visual benchmarks but struggle with out-of-distribution (OOD) tasks in specialized domains such as medical imaging, where labeled data is limited and expensive. We introduce LEAML, a label-efficient adaptation framework that leverages both scarce labeled VQA samples and abundant unlabeled images. Our approach generates domain-relevant pseudo question-answer pairs for unlabeled data using a QA generator regularized by caption distillation. Importantly, we selectively update only those neurons most relevant to question-answering, enabling the QA Generator to efficiently acquire domain-specific knowledge during distillation. Experiments on gastrointestinal endoscopy and sports VQA demonstrate that LEAML consistently outperforms standard fine-tuning under minimal supervision, highlighting the effectiveness of our proposed LEAML framework.

Reward Models are Metrics in a Trench Coat
Автори: *Chen Zhang, Yaxin Liu, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng*
Посилання: [Читати на Arxiv](http://arxiv.org/pdf/2510.03231v1)

Анотація:
> The emergence of reinforcement learning in post-training of large language models has sparked significant interest in reward models. Reward models assess the quality of sampled model outputs to generate training signals. This task is also performed by evaluation metrics that monitor the performance of an AI model. We find that the two research areas are mostly separate, leading to redundant terminology and repeated pitfalls. Common challenges include susceptibility to spurious correlations, impact on downstream reward hacking, methods to improve data quality, and approaches to meta-evaluation. Our position paper argues that a closer collaboration between the fields can help overcome these issues. To that end, we show how metrics outperform reward models on specific tasks and provide an extensive survey of the two areas. Grounded in this survey, we point to multiple research topics in which closer alignment can improve reward models and metrics in areas such as preference elicitation methods, avoidance of spurious correlations and reward hacking, and calibration-aware meta-evaluation.

LEAML: Label-Efficient Adaptation to Out-of-Distribution Visual Tasks for Multimodal Large Language Models

Автори: *Chen Zhang, Yaxin Liu, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng* **Посилання:** Читати на Arxiv

Анотація:

Multimodal Large Language Models (MLLMs) have achieved strong performance on general visual benchmarks but struggle with out-of-distribution (OOD) tasks in specialized domains such as medical imaging, where labeled data is limited and expensive. We introduce LEAML, a label-efficient adaptation framework that leverages both scarce labeled VQA samples and abundant unlabeled images. Our approach generates domain-relevant pseudo question-answer pairs for unlabeled data using a QA generator regularized by caption distillation. Importantly, we selectively update only those neurons most relevant to question-answering, enabling the QA Generator to efficiently acquire domain-specific knowledge during distillation. Experiments on gastrointestinal endoscopy and sports VQA demonstrate that LEAML consistently outperforms standard fine-tuning under minimal supervision, highlighting the effectiveness of our proposed LEAML framework.

Reward Models are Metrics in a Trench Coat

Автори: *Chen Zhang, Yaxin Liu, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng, Yizhe Zhang, Shuohang Wang, Yuxin Peng* **Посилання:** Читати на Arxiv

Анотація:

The emergence of reinforcement learning in post-training of large language models has sparked significant interest in reward models. Reward models assess the quality of sampled model outputs to generate training signals. This task is also performed by evaluation metrics that monitor the performance of an AI model. We find that the two research areas are mostly separate, leading to redundant terminology and repeated pitfalls. Common challenges include susceptibility to spurious correlations, impact on downstream reward hacking, methods to improve data quality, and approaches to meta-evaluation. Our position paper argues that a closer collaboration between the fields can help overcome these issues. To that end, we show how metrics outperform reward models on specific tasks and provide an extensive survey of the two areas. Grounded in this survey, we point to multiple research topics in which closer alignment can improve reward models and metrics in areas such as preference elicitation methods, avoidance of spurious correlations and reward hacking, and calibration-aware meta-evaluation.

ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

Загальне завдання для всіх варіантів

Розробити унікального AI-агента на базі LlamaIndex, який автономно вирішує практичну задачу, використовуючи інструменти з екосистеми LlamaHub.

Етап 1. Дослідження та проєктування (20%)

Дослідження LlamaHub:

- Відвідайте сайт LlamaHub та перейдіть у розділ «Agent Tools».
- Ознайомтеся з різноманіттям доступних інструментів. Проаналізуйте щонайменше 3-5 ToolSpec, які здалися вам цікавими.

Вибір основного інструмента:

- Оберіть один ToolSpec, який стане ядром вашого проєкту.
- У звіті обґрунтуйте свій вибір: чому саме цей інструмент? Який потенціал ви в ньому бачите? Яку проблему він може допомогти вирішити?

Формулювання ідеї та проєктування агента:

- Придумайте назву та роль для вашого агента (напр., «AI-Композитор», «Гід по Подіях», «Аналітик Коду»).
- Сформулюйте чітку, амбітну, але досяжну високорівневу мету для агента. Мета повинна вимагати автономного виконання кількох кроків.
- Розробіть план дій (workflow) для вашого агента. Опишіть по кроках, як він буде досягати мети. Приклад: крок 1 – отримати початкові дані від користувача; крок 2 – викликати інструмент X для пошуку інформації; крок 3 – викликати інструмент Y для обробки даних; крок 4 – сформувати фінальний звіт/результат; крок 5 – зберегти результат у файл (або виконати іншу фінальну дію).

Етап 2. Технічна реалізація (60%)

Налаштування середовища:

- Створіть Python-проєкт та встановіть усі необхідні бібліотеки: llama-index, ollama, а також пакет для обраного вами інструмента з LlamaHub.
- Переконайтеся, що локальна LLM (qwen3:8b або аналогічна) завантажена та доступна через Ollama.

Розробка інструментів:

- Ініціалізуйте обраний ToolSpec з LlamaHub. Якщо потрібно, отримайте та налаштуйте API-ключі (використовуйте безкоштовні тарифи).
- Якщо для вашої ідеї потрібен додатковий функціонал (наприклад, запис у файл, кастомна обробка даних), створіть власні інструменти за допомогою FunctionTool.

Створення агента:

- Напишіть детальний системний промпт для вашого FunctionAgent. Цей промпт – найважливіша частина вашої роботи. Він повинен містити: 1) чітке визначення ролі та «особистості» агента; 2) перелік доступних йому інструментів та пояснення, коли який використовувати; 3) покроковий план дій (workflow), який ви розробили на етапі проєктування; 4) шаблон форматування для фінального звіту (дуже рекомендовано).
- Створіть екземпляр FunctionAgent, передавши йому LLM, список інструментів та системний промпт. Не забудьте ввімкнути verbose=True.

Запуск та тестування:

- Сформулюйте конкретне завдання (task), яке відповідає меті вашого агента.
- Напишіть асинхронну функцію main() для запуску agent.run(user_msg=task).
- Запустіть скрипт і переконайтеся, що агент працює автономно та досягає очікуваного результату (наприклад, створює файл). Налагоджуйте промпт та інструменти, доки результат вас не задовольнить.

Етап 3. Документування та звіт (20%)

Оформлення звіту:

Підготуйте детальний звіт про виконання лабораторної роботи, який повинен містити:

- Розділ 1. Проєктування. Опис обраного інструмента, обґрунтування вибору, опис ідеї, ролі та плану дій агента.
- Розділ 2. Реалізація. Повний код вашого проєкту з детальними коментарями, що пояснюють логіку роботи інструментів та системного промпту.
- Розділ 3. Результати: скріншот (або текстова копія) логів з консолі, що демонструє покрокове виконання завдання агентом (його «думки» та виклики інструментів); вміст фінального артефакту, створеного агентом (текст згенерованого файлу, скріншот тощо).
- Розділ 4. Аналіз та Висновки: дайте відповідь на питання: Чи вдалося агенту повністю виконати поставлене завдання автономно? Опишіть труднощі, з якими ви зіткнулися (проблеми з промптом, непередбачувана поведінка LLM, обмеження інструментів). Запропонуйте щонайменше два напрямки для подальшого покращення вашого агента.

ІНСТРУКЦІЇ З НАЛАШТУВАННЯ ТА ЗАПУСКУ

Встановлення Ollama

Відвідайте офіційний сайт ollama.com та завантажте інсталятор для вашої операційної системи. Дотримуйтесь інструкцій зі встановлення. Після встановлення Ollama працюватиме як фоновий сервіс.

Завантаження мовної моделі

Відкрийте термінал або командний рядок та виконайте наступну команду, щоб завантажити модель Qwen 3 (8 мільярдів параметрів). Завантаження може зайняти деякий час.

Примітка! *Можна використати будь-яку модель*

```
ollama pull qwen3:8b
```

LlamaHub

Відвідайте llamahub.ai, щоб знайти потрібний компонент. Виконайте базові інструкції та імпортуйте відповідний *ToolSpec* у ваш код, ініціалізуйте його та передайте агенту, викликавши метод *.to_tool_list()*.

Запуск проєкту

Переконайтесь, що ви виконали крок 1 з «Ходу роботи» (встановили всі Python-бібліотеки). Помістіть весь розроблений код в один файл з назвою main.py. Запустіть проєкт з терміналу, перебуваючи в активному віртуальному середовищі.

```
python main.py
```

Welcome to Github Repository

github.com/F1-bot/llamaindex_sample.