Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ ДЛЯ ДОСЛІДЖЕНЬ У СФЕРІ REINFORCEMENT LEARNING

Текстова частина до курсової роботи за спеціальністю "Інженерія Програмного Забезпечення"

Керівник курсової роботи
Медвідь С. О.
Магістр комп'ютерних наук,
старший викладач
(прізвище та ініціали)
_____
(підпис)
"____" _____ 2025 р.

виконав студент
Лошак М. І.
(прізвище та ініціали)
"____" _____ 2025 р.

Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
проф., д.ф.-м.н.
_____А. М. Глибовець
(підпис)
„____”_____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Студенту 3-го курсу, факультету інформатики
Лошаку Максиму Івановичу

ТЕМА Розробка та впровадження веб-інтерфейсу для досліджень у сфері Reinforcement Learning з використанням Gymnasium

вихідні дані:

- Технічне завдання
- API Gymnasium

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
Наукова новизна та цінність
Огляд літератури
1 Аналіз інструментарію та обґрунтування вибору технологій
2 Дослідження платформи GymnAsium
3 Розробка аспектного перехоплювача
4 Розробка серверної частини застосунку
5 Розробка клієнтської частини застосунку
Висновки
Список літератури

Дата видачі “___” _____ 2024 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання роботи (приклад):

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	02.11.2024	
2.	Огляд технічної літератури за темою роботи.	15.11.2024	
3.	Виконання аналізу методів Reinforcement Learning	25.11.2024	
3.	Розробка алгоритму перехоплення викликів до Gymnasium API	15.12.2024	
4.	Програмування розробленого алгоритму	30.12.2024	
4.	Тестування перехоплювача на різних варіаціях коду тренування RL агентів	10.01.2025	
5.	Розробка серверної частини застосунку	25.01.2025	
6.	Розробка клієнтської частини застосунку та інтеграція ASGI для Web Socket	20.02.2025	
6.	виконання порівняльного аналізу результатів тренування агентів	01.03.2025	
7.	Написання текстової частини роботи.	20.03.2025	
8.	Створення слайдів для доповіді та підготовка дл доповіді.	22.04.2025	
11.	Остаточне оформлення текстової частини та слайдів.	2.05.2025	
12.	Захист курсової роботи (проекту)	14.05.2025	

Студент _____

Керівник _____

“_____” _____

Зміст

Анотація

У даній курсовій роботі розроблено веб-орієнтовану систему для підтримки досліджень із Reinforcement Learning (RL) на базі платформи Gymnasium. Наукова новизна полягає в інтеграції аспектного перехоплення викликів Gymnasium API для автоматичного збору метрик із потоковою передачею даних на інтерактивний веб-інтерфейс. Це дозволяє усунути необхідність ручного втручання в код агента або середовище, забезпечуючи стандартизований збір статистики та її аналіз і візуалізацію даних у режимі реального часу. Ключовою перевагою є зниження технічних бар'єрів для проведення експериментів, підвищення відтворюваності і прискорення RL-досліджень.

Ключові слова: Reinforcement Learning, Gymnasium, веб-інтерфейс, аспектне програмування, візуалізація даних.

Вступ

Reinforcement Learning (RL) є одним із ключових напрямків сучасного машинного навчання [1], який дозволяє алгоритмам навчатися шляхом взаємодії з оточенням та прийняття рішень для досягнення оптимальних результатів. Однією з популярних платформ для експериментів у цій галузі є Gymnasium [2], яка надає дослідникам зручні інструменти для створення та тестування алгоритмів RL. Веб-інтерфейси з відкритим вихідним кодом сприяють залученню нових користувачів і розробників [3], [4], що дозволяє підтримувати спільноту і пришвидшувати процес розвитку RL-алгоритмів.

Незважаючи на активний розвиток RL, залишається питання зручності користування такими платформами, як Gymnasium [2], [5], що обмежує можливості дослідників, які не мають спеціалізованих технічних навичок. Веб-інтерфейси, які дозволяють легко обирати задачі, запускати процеси RL та зберігати результати експериментів, вже продемонстрували здатність залучати ширшу спільноту розробників і науковців [3], [6]. Додатковий функціонал для архівування проєктів зробить роботу з результатами експериментів більш впорядкованою та продуктивною.

Метою даної роботи є розробка та впровадження веб-орієнтованої системи для підтримки RL досліджень з використанням Gymnasium, яка забезпечить користувачу інтуїтивне керування експериментами та візуальний аналіз результатів. Для досягнення поставленої мети необхідно вирішити такі завдання: 1) провести аналіз сучасних засобів та технологій, придатних для реалізації інтерфейсу підтримки RL-досліджень; 2) обґрунтувати вибір архітектури системи, зокрема серверної частини (бекенду) для збору і збереження даних та клієнтської частини (фронтенду) для їх візуалізації; 3) розробити механізм аспектного перехоплення викликів Gymnasium API для автоматичного збору статистики взаємодії агента з середовищем; 4) спроєктувати і реалізувати серверну частину застосунку, що отримуватиме, структуруватиме та зберігатиме зібрані дані в базі даних, а також надаватиме API для доступу до них; 5) розробити інтерактивний веб-інтерфейс, який відображатиме отримані результати навчання (графіки показників, таблиці

статистики тощо) та оновлюватиме їх у режимі реального часу; 6) перевірити працездатність та зручність створеної системи на практичних прикладах, проаналізувати результати та сформулювати рекомендації щодо її подальшого розвитку.

Об’єктом дослідження в цій роботі є процес взаємодії RL-агента з середовищем у Gymnasium та методи автоматизації моніторингу експериментів. Саме аналіз та збір даних про часові витрати і приріст винагороди є критичними для оцінки ефективності алгоритмів та подальшого покращення моделей [1].

Дана курсова робота спрямована на розробку та впровадження такого веб-інтерфейсу для Gymnasium. Основна мета проєкту — підвищити зручність і доступність використання RL-досліджень, що є важливим кроком для розширення аудиторії користувачів та прискорення процесу розробки інноваційних RL-алгоритмів.

Наукова новизна та цінність

Запропонований у цій роботі підхід демонструє наукову новизну завдяки синтезу різних методологічних рішень, що дозволяють спростити інфраструктуру проведення експериментів у сфері Reinforcement Learning. Синтез у даному контексті проявляється у поєднанні двох ключових компонентів: динамічного перехоплення викликів Gymnasium API та автоматичного збору статистичних даних із застосуванням сучасних веб-технологій для відображення результатів у режимі реального часу. Досліднику більше не потрібно вручну інтегрувати засоби моніторингу й оцінки роботи агента: достатньо лише скористатися модифікованим середовищем, яке логує всі необхідні метрики та передає їх на сервер в режимі реального часу. Така процедура суттєво знижує ризик появи помилок, які часто виникають при реалізації збору даних дослідником, а також заощаджує час підготовки експерименту. Цей синтез дозволяє не лише мінімізувати ручне втручання у процес збору даних, але й забезпечує уніфікацію процесу аналізу, що сприяє швидшому виявленню проблем та точнішій налаштуванню гіперпараметрів. Особливе значення має веб-інтерфейс, вбудований у розроблену платформу. Згідно з сучасними тенденціями візуальної аналітики для штучного інтелекту, інтерактивні панелі моніторингу підвищують прозорість моделей і заохочують до формування нових інсайтів [4], [7]. Подібні до нашого підходи (наприклад, AutoRL X від IBM Research) доводять: онлайн-відслідковування агентів у реальному часі допомагає глибше зрозуміти алгоритм і спростити процес налаштування гіперпараметрів. Аналогічні дослідження (Neuwirth & Riley, 2020) вказують на те, що інтерактивна візуалізація в навчанні RL-агентів забезпечує інтуїтивніше розуміння механізмів ухвалення рішень, сприяє швидшому виявленню проблем і полегшує процес експериментів із різними конфігураціями [7].

Огляд Літератури

Сфера Reinforcement Learning (RL) стрімко розширюється завдяки появі численних фреймворків, бібліотек та платформ, що надають широкий спектр засобів для розробки й тестування алгоритмів. Стандартизовані підходи до реалізації середовищ дозволяють прискорити дослідження, оскільки вони спрощують інтеграцію та взаємодію між різними агентами й оточеннями [2]. Водночас окремі дослідницькі середовища з графічними інструментами створені для спрощення розробки оточень і візуалізації їхньої роботи, однак у більшості таких рішень не вистачає інтегрованих механізмів для централізованого керування експериментами чи для гнучкого відстеження статистики [3].

Глибоке занурення в роботу середовищ RL, орієнтованих на процедурну генерацію контенту або на адаптивне навчання агентів, засвідчує, що потреби науковців уже давно вийшли за рамки базового набору API-функцій [23], [24], [25]. Існують платформи, які полегшують налаштування експериментів і візуалізацію результатів, проте ці рішення зазвичай сфокусовані на конкретних завданнях або вимагають додаткових зусиль, аби розширити чи кастомізувати їхній функціонал [6], [26], [27]. У деяких роботах пропонуються рекомендації щодо забезпечення стандарту відтворюваності та покращення безпеки експериментів, та все ж у готових інструментах відсутній універсальний веб-інтерфейс, що надавав би можливість ініціювати, зупиняти та архівувати експерименти на централізованій платформі [28], [29], [30].

Чимало дослідницьких ініціатив, які розгортаються у хмарному або розподіленому середовищі, мають на меті пришвидшити тренування агентів, проте не концентруються на тому, як спростити взаємодію дослідників із самими оточеннями та зібраними даними [5], [6], [31], [32]. У такий спосіб формуються прогалини, пов’язані зі зручністю використання та інструментальною підтримкою збереження та аналізу результатів у режимі реального часу. Попри наявність ряду бібліотек із відкритим кодом, “єдина платформа для інтегрованого керування RL-дослідженнями залишається здебільшого концепцією” [4].

Існують також спроби пропонувати “low-code” рішення, покликані спростити пошук оптимальних параметрів і конфігурацій agent-environment [27]. Однак вони рідко передбачають механізми, необхідні для одночасного відстеження виконання декількох алгоритмів чи порівняльного аналізу результуючих політик. Відтак залишаються невирішеними такі завдання:

- Автоматизоване перехоплення викликів до бібліотек RL без прямого втручання у вихідний код користувача.
- Зручне керування процесами навчання (запуск, зупинка, відновлення) зі збереженням докладних логів у єдиному сховищі.
- Інтуїтивно зрозумілий веб-інтерфейс, що дозволяє переглядати аналітику й порівнювати результати різних агентів у реальному часі.

Запропонований у межах цього проєкту підхід охоплює відкритий веб-інтерфейс, побудований із використанням Django та React, який поєднує базу даних для зберігання статистики, а також механізм перехоплення викликів в Gymnasium через аспектний інтерсептор. Такий інтерсептор забезпечує динамічне захоплення кожного виклику до API Gymnasium, не вимагаючи від користувача вручну модифікувати початковий код. Передбачається, що зібрані виклики автоматично потраплятимуть у єдину систему звітності, на основі якої можна формувати візуалізації, таблиці порівнянь та архівувати проміжні результати експериментів. Завдяки цьому не лише підвищується відтворюваність досліджень, а й зменшується поріг входу для нових користувачів, які не хочуть витрачати час на комплексне налаштування логування, аналізу та порівняльних обчислень.

Завдяки комплексній інтеграції механізмів збору метрик, управління проєктами та веб-візуалізації, така система потенційно задовольняє запит на більш цілісне, зручне й масштабоване рішення в галузі RL-досліджень. Використання аспектного інтерсептора сприяє мінімізації змін у коді користувачів, а модульна архітектура на основі Django дає змогу розширювати функціонал залежно від потреб конкретного дослідницького кейсу. Саме це й

позиціює проєкт як такий, що прагне заповнити наявну прогалину між високорівневими візуалізаційними інструментами й вузькоспеціалізованими бібліотеками RL. Тож поєднання технологій веб-розробки, автоматизованого моніторингу викликів та зручної інтеграції середовищ RL може стати важливим кроком для оптимізації дослідницьких процесів у спільноті Reinforcement Learning.

РОЗДІЛ 1: Аналіз інструментарію та обґрунтування вибору технологій

1.1 Аналіз технологій для реалізації проєкту

Оскільки постановка задачі вбачає використання Gymasium, бібліотеки на Python для дослідників Reinforcement Learning, вибір Python для реалізації серверної частини застосунку був очевидним. Серед потенційних кандидатів організації повнофункціональної сарверної частини на Python явним лідером є Django.

Для френтенду було обрано React і, на відміну від бекенду у випадку з фронтенд-технологіями було багато переваг, як в React, так і в Angular, тому ознайомимося із детальним обґрунтуванням вибору даного стеку:

1.1.1 Django (серверна частина)

- **Повнофункціональність**: Django є фреймворком з підходом "batteries-included", що надає широкий спектр вбудованих функцій, таких як ORM для роботи з базами даних, система автентифікації, панель адміністратора та інші. Це прискорює розробку та забезпечує узгодженість коду [8], [9].
- **Безпека**: Django забезпечує високий рівень безпеки завдяки вбудованим механізмам захисту від поширених вразливостей, таких як SQL-ін'єкції та CSRF-атаки [9], [10], [11].
- **Масштабованість**: Фреймворк підходить для розробки як малих, так і великих проєктів, забезпечуючи легку масштабованість та підтримку високих навантажень [8], [9].
- **Community**: Django має велику та активну спільноту розробників, що забезпечує доступ до численних ресурсів, пакетів та розширень [10], [11].

1.1.2 React (клієнтська частина)

- **Гнучкий у використанні**: React дозволяє створювати інтерфейси користувача з використанням компонентів, що дозволяє повторно використовувати код за умови якісної організації [12].

- **Висока продуктивність**: Завдяки віртуальному DOM, React забезпечує швидке оновлення інтерфейсу без перезавантаження сторінки, що покращує користувацький досвід [12], [13].
- **Широка екосистема**: React має велику кількість бібліотек та інструментів, що розширюють його можливості та спрощують розробку складних інтерфейсів [14].
- **Community**: Як і Django, React має активну спільноту, що забезпечує швидке вирішення проблем та доступ до численних навчальних матеріалів [12].

1.2 Аналіз обраних технологій і порівняння з альтернативами

Критерій	**Django**	**Flask**
Архітектура	Повнофункціональний фреймворк з підходом "batteries-included".	Мікрофреймворк, що надає лише базові функції, залишаючи вибір додаткових компонентів розробнику.
Швидкість розробки	Швидка розробка завдяки вбудованим інструментам та автоматизації багатьох завдань.	Потребує більше часу на налаштування та інтеграцію додаткових бібліотек для реалізації необхідного функціоналу.
Масштабованість	Підходить для великих проєктів з високими вимогами до функціоналу, безпеки та ефективності.	Краще підходить для невеликих проєктів або мікросервісів.

Критерій	**React**	**Angular**
Архітектура	Бібліотека для побудови інтерфейсів користувача з	Менша; дотримується встановленої структури

	компонентним підходом.	та підходів, але завдяки цьому більш прямолінійний а розробці.
Гнучкість	Висока; дозволяє вибирати додаткові бібліотеки та інструменти за потребами проєкту.	Менша; дотримується встановленої структури та підходів, але завдяки цьому більш прямолінійний а розробці.
Продуктивність	Висока завдяки віртуальному DOM та ефективному оновленню компонентів.	Також висока, але може вимагати більше ресурсів через складність фреймворку.
Крива навчання	Плавна; легше почати, але потребує часу для освоєння екосистеми. (Це можна вважати плюсом на користь React в рамках даного курсового проєкту)	Більш стрімка через складність та кількість вбудованих концепцій. (Що не є оптимальним в контексті даного проєкту)

1.3 Методи перехоплення викликів функцій: огляд

Після аналізу можливих підходів, таких як декоратори, проксі та аспекти, було прийнято рішення використати аспектний інтерсептор, оскільки він забезпечує найкраще поєднання гнучкості, масштабованості та відповідності вимогам задачі. Основними перевагами аспектного підходу в цьому контексті є наступні:

1. Повне охоплення всіх методів без зміни вихідного коду.

Аспектні інтерсептори забезпечують перехоплення будь-яких викликів методів об’єкта, незалежно від їх кількості чи призначення [15]. На відміну від проксі, які часто вимагають явного переліку методів для

обробки, аспектний інтерсептор автоматично динамічно охоплює всі функції API (`step()`, `reset()`, `render()` тощо). Це дозволяє створити єдину обгортку для всіх методів, що значно спрощує інтеграцію та зменшує ризик пропустити важливі виклики [16]. Основною перевагою над декораторами є те, що аспектний підхід не вимагає вручного додавання декораторів до кожного методу, що дозволяє зберегти чистоту коду, зменшити його обсяг та кількість змін початкового коду користувача [16].

2. Гнучкість у налаштуванні логування та збору статистики.

Аспектний інтерсептор легко налаштовується для збору інформації про:

- Час виклику методів (`track_time`).
- Аргументи, які передаються у функції (`track_args`).
- Результати, що повертаються функціями (`track_res`).
- Кількість викликів кожного методу (`collect_statistics`).

У порівнянні з іншими підходами, цей рівень деталізації та адаптивності є значно простішим для впровадження та масштабування.

3. Незалежність від структури об'єкта та відсутність конкретики.

Аспектний інтерсептор працює динамічно, використовуючи такі інструменти, як `getattr`, для перехоплення методів, не вимагаючи конкретних назв методів, до яких буде застосований перехоплювач [17]. На мою думку, це особливо важливо в середовищі Gymnasium, де API може змінюватися між версіями або мати різні набори методів у різних типах середовищ, таким чином наш код не ризикує стати `deprecated` через короткий проміжок часу.

4. Зручність для обробки великої кількості функцій API.

Gymnasium API може містити десятки функцій, і забезпечення підтримки кожної з них через проксі або декоратори було б неефективним. Аспектний підхід автоматизує цей процес, що дозволяє сконцентруватися на логіці перехоплення замість вручну прописаних методів [16].

5. Масштабованість для майбутніх задач.

Якщо завдання з часом ускладниться (наприклад, потрібно буде перехоплювати виклики у багатопоточному середовищі або в асинхронних викликах), аспектний підхід легко адаптується. Існують бібліотеки, такі як `wrapt` або `aspectlib`, які надають можливості для більш складних аспектних сценаріїв, дозволяючи розширювати функціональність без значного переписування коду.

6. Відокремлення перехоплення від основної логіки.

Аспектний інтерсептор дозволяє реалізувати програмування, орієнтоване на аспекти (Aspect-Oriented Programming, AOP), при якому перехоплення викликів методів із їх логуванням, моніторингом або збором статистики здійснюється окремо від основної логіки програми [16]. Це забезпечує високу модульність та повторне використання коду [17].

7. Поширене використання AOP.

Підхід, орієнтований на аспекти, є перевіреним стандартом у багатьох галузях розробки програмного забезпечення [15]. Його застосовують у таких фреймворках та бібліотеках, як:

- Spring AOP
- Python Wrapt
- Aspects.py

1.3.1 Переваги аспектного перехоплювача для Gymnasium API

- **Динамічність:** Gymnasium API має велику кількість методів, і перехоплення всіх викликів без внесення змін у вихідний код середовища є ключовою вимогою. Аспектний інтерсептор обробляє це автоматично.
- **Гнучкість налаштувань:** впроваджений нами аспектний інтерсептор дозволяє користувачеві вмикати чи вимикати окремі функції (логування, збір статистики, відстеження часу тощо), забезпечуючи повний контроль над перехопленням.
- **Мінімальне втручання:** Оскільки Gymnasium постійно розвивається, аспектний підхід дозволяє легко оновлювати інтерсептор без потреби у зміні структури програмного коду.

РОЗДІЛ 2: Дослідження платформи Gymnasium

Перш ніж розпочати розробку власних експериментів та досліджень, було важливо досконало ознайомитися зі структурою та можливостями бібліотеки Gymnasium. Ця бібліотека є стандартизованим набором інструментів для створення та оцінювання методів навчання з підкріпленням (Reinforcement Learning, RL) [2]. Нижче наведено короткий опис того, для чого саме використовується Gymnasium, чому вона була обрана для цього проєкту та які кроки виконувались задля тренування агента у файлі train.py.
Gymnasium (раніше відома як OpenAI Gym) є ключовим набором середовищ та інструментів, які надають уніфікований інтерфейс для розробки та оцінки методів RL. Вона дає змогу стандартизувати процес розробки агентів і спрощує експерименти, оскільки в усіх середовищах реалізовані однакові методи взаємодії, зокрема reset(), step(), render() та close(). Це важливо для забезпечення відтворюваності та уніфікації результатів. Gymnasium було обрано для проєкту завдяки широкому вибору середовищ із дискретним простором дій, можливості швидкої інтеграції з популярними фреймворками (PyTorch, TensorFlow) та активній підтримці дослідницької спільноти.

2.1 Структура платформи та підготовка середовища до тренування

На початковому етапі було створено файл train.py, у якому ініціалізується середовище за допомогою gym.make("LunarLander-v2"). Після виклику reset() агент переходить в початковий стан і може починати взаємодію із середовищем, виконуючи дискретні дії. Кожен виклик env.step(action) повертає наступний стан, винагороду та індикатор завершення епізоду (рисунок 2.1). У цьому ж файлі реалізовано QNetwork і буфер повторних перегравань, куди зберігаються досвіди для подальшого навчання. На ранніх етапах тренування застосовується epsilon-greedy стратегія, щоб забезпечити баланс між дослідженням середовища та використанням вже здобутих знань. Згідно цієї стратегії спочатку порівнюється випадкове число з epsilon [18]. Якщо це число менше за epsilon, агент виконує випадкову дію (у даному випадку envs.single_action_space.sample()), щоб досліджувати середовище. В іншому

випадку, агент обирає дію, що приносить найбільше очікуване значення винагороди (береться argmax з нейронної мережі).

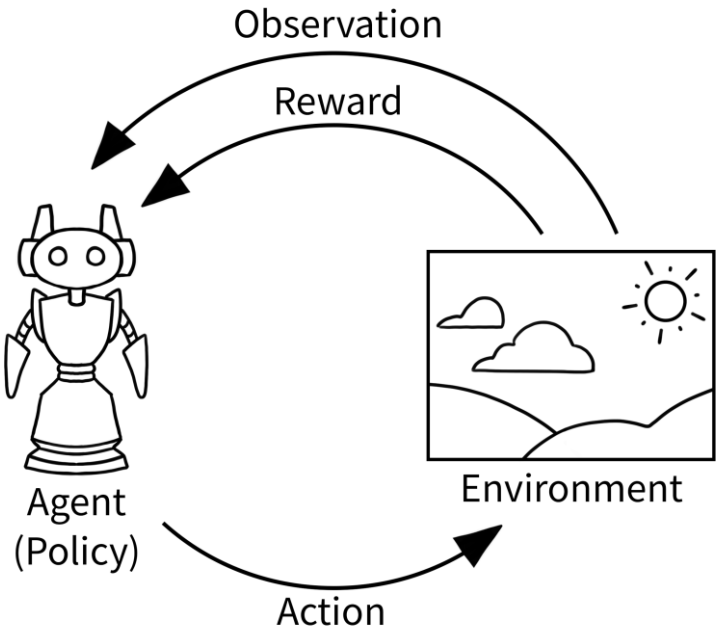


Рисунок 2.1, image credit: Farama

2.2 Аналіз викликів API Gymnasium та їх перехоплення

Щоби з’ясувати, як саме агент поводиться під час взаємодії, робилися короткі тести з виведенням інформації в консоль. Зокрема, перевірялися епізодичні винагороди та кількість кроків до завершення епізоду, а також аналізувалися найпоширеніші причини переходу в стан завершення (наприклад, вихід за межі ландера). Завдяки логуванню дій стало зрозуміло, як швидко агент здатен отримувати позитивні результати і яка саме динаміка показника винагороди прослідковується від епізоду до епізоду.

2.3 Переваги обраного підходу

Оскільки Gymnasium надає уніфікований інтерфейс для цілого спектра середовищ, це дозволило зосередитися на побудові алгоритму навчання, а не на технічних деталях симуляцій чи створенні середовищ з нуля. Стандартні методи reset() і step() також значно спростили процес відлагодження: можна було швидко змінювати параметри (наприклад, розмір буфера чи значення epsilon) й одразу бачити вплив на ефективність навчання. Завдяки цьому

початкове ознайомлення з API виявилося зручним та гнучким. У кінцевому підсумку, навіть базове налаштування в train.py надало змогу отримати робочого агента, який поступово вдосконалював свою стратегію управління ландером і досягав дедалі кращих результатів.

РОЗДІЛ 3: Розробка аспектного перехоплювача

Після розгляду різних підходів, зокрема створення *proxy* та аспектних перехоплювачів, було проведено попередні експерименти у файлах test_aspect_interceptor.py та test_proxy_interceptor.py. Початковою метою було перевірити, наскільки зручно конфігурувати й розширювати код для відстеження й логування викликів Gymnasium API. Обидва варіанти передбачали створення додаткового прошарку між середовищем і навчальним алгоритмом (рисунок 3.1), проте результати невеликих тестів підтвердили, що аспектний підхід забезпечує більшу гнучкість та менше втручання в основну логіку.

Ідея *aspect interception* полягає в тому, що кожен виклик методів середовища (таких як reset() і step()) може бути перехоплений динамічно, без потреби вручну змінювати початковий код чи надавати повний перелік методів для обробки. Файл aspect_interceptor.py демонструє реалізацію такого механізму: усі звернення до об’єкта середовища обгортаються функціональністю інтерсептора, який стежить за переданими аргументами, результатами викликів і при бажанні надсилає зібрані дані на бекенд. Такий підхід не тільки звільняє розробника від ручного опису кожної функції, а й дозволяє за потреби швидко змінювати, яку саме інформацію слід записувати.

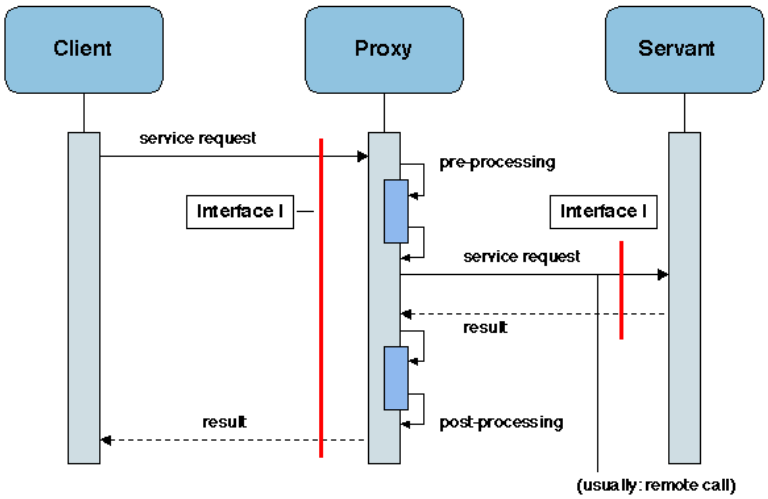
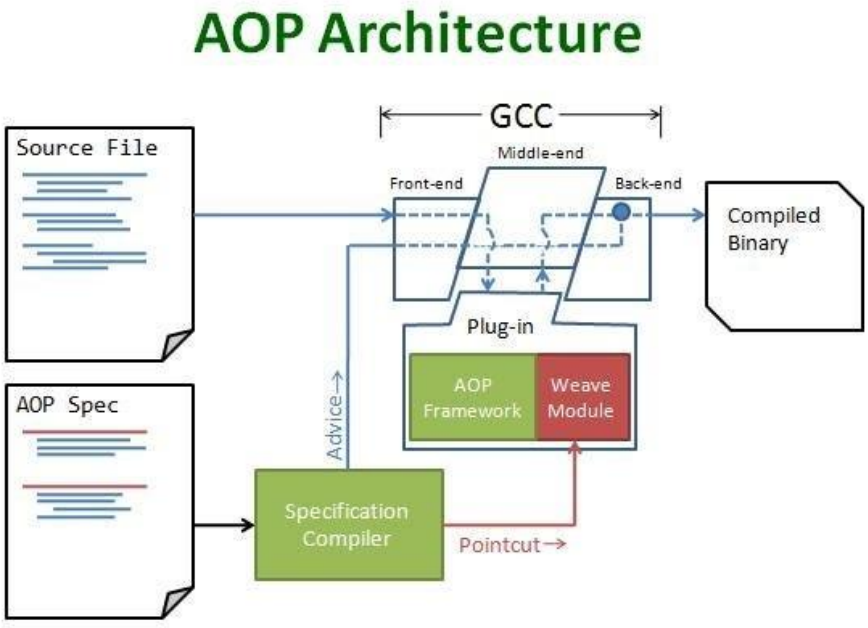


Рисунок 3.1 – Будова proxy перехоплювача, image credit: S. Krakowiak

Серед переваг аспектного перехоплювача можна виокремити можливість узагальненого керування всіма методами Gymnasium без ризику пропустити важливі виклики, а також більшу масштабованість у випадках, коли потрібно додавати розширене логування, знімати часові зрізи чи відстежувати інші статистичні показники [15]. Саме тому, ознайомившись із підходом *proxy* (що реалізований у файлі proxy_interceptor.py і перевірений за допомогою test_proxy_interceptor.py), було прийнято рішення зупинитися на аспектному перехопленні як основному механізмі.

Під час написання перехоплювача особливу увагу було приділено можливості легкого налаштування та використання його в коді для тренування. Щоб використати перехоплювач для відслідковуваввння всіх обраних метрик, досліднику достатньо замінити в коді середовище, яке надається Gymnasium API на середовище, яке повертається класом GymAspectInterceptor. Цей клас отримує в конструктор середовище Gymnasium та обгортає його, забезпечуючи перехоплення викликів функцій, пов’язаних з ним. У aspect_interceptor.py передбачені опції для керування тим, що саме слід логувати (вхідні аргументи, час виклику, результат тощо) і куди надсилати ці дані (наприклад, локальний файл чи віддалений сервер). Такий акцент на конфігурації спрощує адаптацію коду під потреби конкретного агента і середовища, даючи змогу гнучко вмикати чи вимикати різні функції без зміни основного модуля навчання.



AOP Architecture, image credit: **Gökhan Gökalp**

У результаті тестів у файлах test_aspect_interceptor.py та test_proxy_interceptor.py стало зрозуміло, що аспектний підхід потребує мінімальних змін у базових модулях, оскільки він не вимагає створення окремих обгорток для кожного методу середовища чи внесення змін у вихідний код [16]. Завдяки цьому уся бізнес-логіка тренування залишається прозорою, а логіка перехоплення сконцентрована в одному окремому модулі. Така структурована організація коду дозволяє швидко вносити доповнення або перебудовувати процес збору статистики, не зачіпаючи ядро програми [17].

3.1 Перевірка концепції аспектного підходу

Після завершення базової реалізації аспектного перехоплювача було здійснено перевірку дієвості цього підходу на двох прикладах. Спочатку інтерсептор було інтегровано в попередньо написаний код для тренування агента в середовищі з дискретними діями (як *clean code use case*), щоб упевнитися, що перехоплення викликів із фіксацією аргументів та результатів у консолі відбувається належним чином й не перешкоджає решті коду. Далі перехоплювач протестували в середовищі відкритого репозиторію дослідника **enajx** (GitHub, репозиторій **NDP**), де також успішно вдалося логувати виклики методів та їх параметри в консоль (рисунок 3.2). Завдяки цьому вдалося підтвердити універсальність та дієвість аспектного підходу, який легко переноситься між різними кодовими базами й не конфліктує з існуючими механізмами RL.

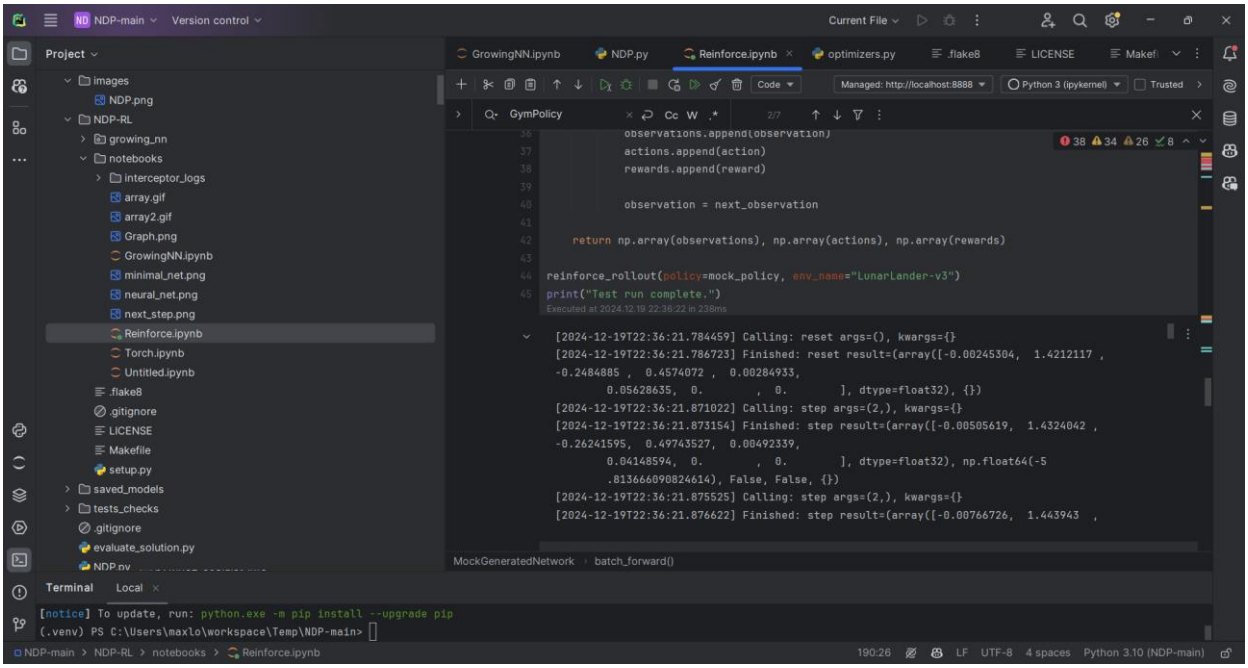


Рисунок 3.2 – виведення викликів в консоль в коді enajx, файл Reinforce.ipynb

Сукупність цих причин спонукала припинити подальшу розробку *proxy*-перехоплювача й зосередитися саме на аспектному рішенні, що вбачається більш гнучким і перспективним для майбутніх експериментів із Gymnasium API.

РОЗДІЛ 4: Розробка серверної частини застосунку

У процесі розробки системи збору та опрацювання даних про навчання з підкріпленням (RL) було обрано фреймворк Django для створення серверної логіки. Окрім вказаних у розділі 1.1.1 серед переваг Django під час вибору конкретної технології для написання бекенду була можливість швидко налаштувати робоче середовище, скористатися автоматичною конфігурацією бази даних (SQLite) та отримати зручний інструментарій для побудови API. Основними завданнями на початковому етапі виконання курсового проєкту були:

1. Забезпечити можливість отримувати від аспектного перехоплювача «сирі» дані про виконання епізодів та дії агента.
2. Зберігати та структурувати ці дані у базі, для подальшого аналізу винагороди, кроків і відповідно вдосконалювати модель.
3. Організувати ендпоінти, які давали б можливість не тільки зберігати інформацію, а й переглядати її або згодом розширювати під більш складні сценарії.

4.1 Проєктування моделей даних та їх призначення

На перших кроках у файлі models.py було визначено базову структуру для зберігання епізодичної статистики — клас EpisodeStatistic. В ньому зберігається основна статистика про процес тренування, а саме такі метрики:

- avg_reward: середня винагорода за епізод,
- total_reward: сумарна винагорода за епізод,
- steps: кількість кроків, що були виконані агентом,
- created_at: дата і час створення запису (використовується не тільки з точки зору вкорядкування даних, а й для відслідковування кількості часу, який було затрачено на один епізод).

Спочатку в межах мого проєкту вся статистика зберігалась лише для одного агента, тож потреба в розмежуванні сесій на той час ще не була актуальною. Модель EpisodeStatistic успадковувалася від models.Model Django і мала поля з

відповідними типами (*FloatField, IntegerField* тощо). Django покриває такі моделі ORM-можливостями, що спрощує як запис, так і подальший вибір даних.

4.2 Обробка та надсилання даних з перехоплювача на сервер

Щоб відправити на сервер інформацію про те, що відбувається в середовищі навчання (конкретніше, виклики step() та reset()), було створено кілька початкових ендпоінтів у views.py. Особливу роль виконує клас EpisodeAggregatorView, де реалізовано методи для обробки запитів *GET* і *POST*.

1. Метод GET: дає змогу переглянути всі існуючі записи поепізодної статистики та є стандартним HTTP методом для отримання даних з сервера.
2. Метод POST: приймає JSON-дані про поточний виклик step() або reset(). Перехоплювач (*GymAspectInterceptor*) відправляє такі ключі, як:
    - function_name — назва викликаного методу (step, reset тощо),
    - raw_result — «сирі» дані, отримані від середовища (наприклад, список винагород або ознаки завершення епізоду),
    - session_id (пізніше стане в нагоді для відокремлення декількох незалежних сесій, але на початку всі епізоди писалися ніби в одну спільну «сесію»).

На боці сервера діє проста логіка: якщо function_name="step" і в raw_result виявлено ознаку завершення епізоду (done або truncated дорівнює True), ми обчислюємо середню винагороду avg_reward=accum_reward/steps\text{avg_reward} = \text{accum_reward} / \text{steps}avg_reward=accum_reward/steps, формуємо новий запис EpisodeStatistic і зберігаємо його в базі. Якщо ж приходить function_name="reset", це сигнал до того, щоб «завершити» попередній епізод (за наявності непросумованих даних) і підготуватися до початку нового. Таким чином, початковий варіант бекенду виконував роль «накопичувача» даних: він перекладав обчислення середньої винагороди на свій бік, щоб перехоплювач залишався якомога «тоншим» і не займався складнішими

підрахунками, тим самим погіршуючи продуктивність на client side, що могло б призвести до гіршого user experience.

4.3 Підтримка декількох сесій і структурування експериментів

Під час імплементації основного функціоналу з розшифрування та збереження даних про епізоди на стороні бекенду, постало завдання розширити можливості серверної частини, щоб дати користувачу наступні можливості:

- Порівнювати різні гіперпараметри одного агента,
- Запускати паралельні експерименти з різними моделями,
- Зберігати результати тестів, не змішуючи їх, щоб потім можна було провести порівняння який метод тренування виявився більш ефективним.

Щоб виконати цю задачу, у models.py було додано ще один клас: TrainingSession. Спершу його не було, тож EpisodeStatistic належав «умовному» єдиному агентові. Однак, після впровадження TrainingSession було додано:

1. Поле session у EpisodeStatistic, котре пов’язує запис про епізод (avg_reward, total_reward, steps) з конкретною сесією.
2. Унікальний ідентифікатор id для TrainingSession, а також додаткові поля на кшталт agent_name, start_time, end_time тощо. Їх можна використовувати для зручного відслідковування періоду тренування та того, чи досягнуто певних цільових показників (наприклад, *reward_threshold,* але про нього згодом).
3. Можливість створювати декілька таких сесій через ендпоінт training-sessions/, який реалізується класами TrainingSessionListCreateView і TrainingSessionDetailView (у файлі views.py).

Завдяки цьому будь-який запит із полем session_id тепер може бути зіставлено з конкретною сесією, що дає змогу зберігати дані від різних агентів чи від одного агента з різними налаштуваннями. Django ORM автоматично налагоджує зв’язок «один-до-багатьох» між TrainingSession та EpisodeStatistic, тому при видаленні сесії всі пов’язані з нею епізоди також очищаються.

Декілька важливих деталей реалізації

- **Серіалізатори**: у файлі serializers.py описуються два класи — *EpisodeStatisticSerializer* та *TrainingSessionSerializer*. Вони потрібні для перетворення даних моделей у зручний формат (як правило, JSON) та навпаки. Django REST Framework автоматично надає методи create(), update() тощо, щоб працювати з отриманою інформацією.
- **URL-маршрути**: в urls.py визначено, які саме маршрути пов'язано з відповідними view. Наприклад, "/training-sessions/" відповідає TrainingSessionListCreateView, а "/episode-aggregator/" — EpisodeAggregatorView, де збираються події step, reset тощо.
- **Робота з «сирими» масивами**: оскільки надходять масиви винагород, а інколи й інші дані у raw_result, у коді необхідно перевіряти цілісність цих структур і перетворювати їх в потрібний нам тип (задля обчислень середньої винагороди або її суми). У EpisodeAggregatorView використовується бібліотека NumPy для простішої перевірки масивів done_array та truncated_array.

На цьому етапі було створено стійку основу для зберігання та обробки даних, що надходять від аспектного перехоплювача. Починаючи з найпростішої моделі EpisodeStatistic (коли вся статистика належала єдиному агентові), система перейшла до повноцінної підтримки декількох паралельних сесій (клас TrainingSession), що дозволило комфортно розділяти та аналізувати результати різних експериментів. Завдяки інтеграції з Django REST Framework написані ендпоінти, за допомогою яких перехоплювач надсилає не оброблені дані про винагороду й завершення епізодів, а сервер перетворює їх на доступні для перегляду та аналізу записи в базі даних.

4.5 Генерація порівняльної статистики

Після створення основного фундаменту для обробки й зберігання епізодичних даних виникла потреба у подальшому розширенні функціональності на бекенді, щоб надати досліднику додаткові інструменти для зручного й гнучкого аналізу. Для цього було реалізовано можливість генерувати графіки (з використанням пакету matplotlib) безпосередньо через спеціалізовані ендпоінти (див. код у файлі code\gym_rl_manager\plots\views.py). Це зроблено для тих користувачів,

які в певних випадках віддають перевагу консольній чи постман-орієнтованій роботі з даними замість фронтенду (рисунок 4.1). Таким чином, у межах запиту GET до відповідного маршруту (наприклад, /plots/<session_id>/reward/) формується PNG-зображення графіка (середньої винагороди або інших метрик) і повертається у відповідь.

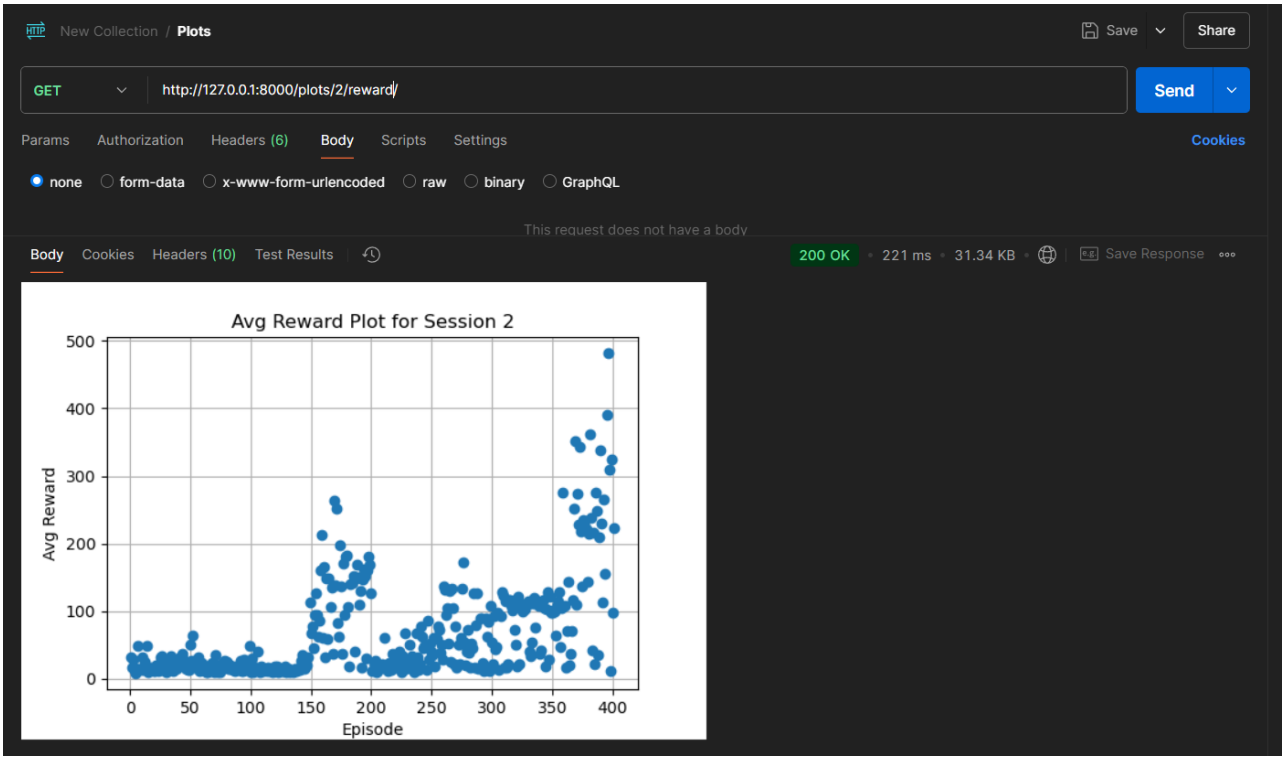


Рисунок 4.1 – Отримання графіку тренування агента з використанням Postman

Окрім можливості згенерувати та завантажити графіки безпосередньо через спеціально розроблені ендпоінти, було додано також механізм завантаження та вивантаження даних про модель у форматі JSON (див. ендпоінти /gym/upload-stats/ та /gym/export-stats/session/<session_id>/), завдяки чому дослідник отримує змогу обмінюватися результатами тренування або ж порівнювати їх з результатами інших агентів чи інших дослідників. Це доповнення надає змогу проводити більш комплексний аналіз і робить застосунок гнучким та придатним для інтеграції у будь-який R&D-процес, де потрібен швидкий обмін даними у структкрованому вигляді, в цьому випадку, в форматі JSON.
Наукова новизна даного проєкту полягає в тому, що завдяки спрощеному механізму логування, автоматичному зведенню статистики та вбудованим засобам візуалізації дослідник позбавляється потреби самостійно програмувати

інструменти для моніторингу та оцінки результатів власних алгоритмів RL. Тобто все, що тепер потрібно для оцінки ефективності агента, — це розробити (або використати наявний) Gymnasium-сумісний алгоритм і використовувати середовище, що перехоплює виклики до API. Використання наведеного в цій роботі перехоплювача унеможливлює виникнення помилок, що часто з’вляються при реалізації збору метрик, а також значно скорочує час на підготовку експерименту.

4.6 Інтеграція REST API та WebSocket

Додатково у процесі розширення функціоналу виникла необхідність оновлювати дані в реальному часі на стороні клієнта. У Django за промовчанням використовується модуль WSGI (Web Server Gateway Interface) для обробки HTTP-запитів (див. code\gym_rl_manager\gym_rl_manager\wsgi.py), однак для підтримки вебсокетів (які працюють за протоколом WS/WSS) потрібен інший серверний шар — ASGI (Asynchronous Server Gateway Interface). Він надає можливість працювати з асинхронними подіями й тим самим реалізовувати двосторонній канал зв’язку з браузером користувача. Для цього в проєкті застосовано Daphne — асинхронний сервер, сумісний з Django[19]. Перехід на Daphne вимагав внести певні зміни у конфігурацію проєкту:

- встановити й налаштувати бібліотеку channels;
- додати файли routing.py з відповідним описом websocket-маршрутів (див. code\gym_rl_manager\training\routing.py);
- визначити StatsConsumer (клас-споживач подій від websocket) у файлі code\gym_rl_manager\training\consumers.py .

Після цього у коді (див. метод broadcast_stats_update у файлі views.py) під час успішного завершення епізоду виконується розсилання повідомлення через get_channel_layer, що в підсумку призводить до оновлення UI на фронтенді або в будь-якій іншій клієнтській частині, де є підписка на відповідний канал вебсокета. У результаті дослідник може в реальному часі спостерігати динаміку зміни середньої винагороди чи інших метрик без потреби постійно оновлювати сторінку вручну.

4.6.1 Консольний та UI-підходи використання

Щоб надати максимальну гнучкість, було реалізовано два шляхи взаємодії з даними:

1. Графічний інтерфейс користувача (UI): забезпечує візуально зрозумілу панель для перегляду активних сесій, аналізу винагород та інших метрик у реальному часі.
2. Консольні скрипти: у каталозі code\scripts\ містяться cli_upload.py та cli_generate.py, які дають змогу з термінала вивантажити JSON-статистику чи навпаки завантажити її до бекенду. Виконуючи:

```
python cli_upload.py my_stats.json --session-id 2
```

можна моментально створити або перезаписати дані сесії №2, не заходячи у вебзастосунок. А команда на кшталт:

```
python cli_generate.py --session-id 2
```

генерує JSON-файл зі зведенням результатів для відповідної сесії.

Завдяки такому підходу дослідник може швидко запускати нові експерименти або інтегрувати ці скрипти у власні пайплайни CI/CD, якщо це потрібно для автоматизації процесу підготовки та аналізу результатів (джерело: code\scripts\cli_upload.py, code\scripts\cli_generate.py).

Переваги розширеного бекенду для дослідника

1. **Генерація графіків «на льоту»**: не потрібне встановлення додаткових бібліотек або запуск окремого застосунку для візуалізації. Достатньо лише відправити HTTP-запит на відповідний ендпоінт (див. код RewardPlotView та TimePlotView).
2. **Легкість обміну даними**: можна зберегти власні результати у JSON та передати їх колезі чи завантажити у власне локальне середовище. Все це

дає змогу миттєво порівнювати результати, не заглиблюючись у деталі реалізації бази даних.

3. **Підтримка вебсокетів**: надає змогу отримувати оновлення в реальному часі, покращуючи user experience.
4. **Гнучкий вибір інтерфейсу**: UI-застосунок або робота з консолі. Завдяки цьому проєкт універсальний і може бути швидко інтегрований у наявні екосистеми досліджень із підкріпленням.

4.7 Висновки щодо реалізації серверної частини

Отже нові можливості забезпечують повноцінний «цикл» роботи дослідника:

1. Запуск експерименту з Gymnasium, інтегрований із аспектним перехоплювачем,
2. Збір та зберігання статистики на бекенді,
3. візуалізація результатів та/або реальний моніторинг метрик через вебсокетне підключення,
4. Можливість експорту та імпорту результатів, що спрощує повторюваність експериментів, порівняння алгоритмів та спільну роботу з іншими фахівцями.

Така сумісність, а також динамічне налаштування каналу зв’язку (HTTP/WS/CLI) характеризують цей проєкт як зручну платформу для досліджень. Це робить роботу інноваційною з погляду реального проведення RL-експериментів: замість того щоб писати з нуля власні системи логування та графічного відображення, достатньо використати наявні модулі (джерело: згаданий функціонал у code\gym_rl_manager\plots\views.py, code\scripts\cli_upload.py тощо), отримавши швидкий результат і уніфікований формат даних.

Таким чином, бекенд-частина проєкту еволюціонувала від простого накопичувача статистики до цілісного вебсервісу, що пропонує:

- Багаторівневу взаємодію (через REST, WebSocket, консоль);

- вбудовані інструменти аналізу (генерація та візуалізація графіків, порівняльний аналіз різних сесій);
- Зручний механізм імпорту/експорту результатів тренування.

Таке комплексне рішення позбавляє користувача рутинних завдань і дає йому змогу сконцентруватися на основних наукових чи дослідницьких аспектах, підтверджуючи важливу роль розробленого програмного забезпечення в актуальній сфері підкріплювального навчання.

РОЗДІЛ 5: Розробка клієнтської частини застосунку

У цьому розділі йтиметься про особливості фронтенд-частини застосунку, яка реалізована на React та повинна забезпечити зручну візуалізацію отриманих від бекенду статистичних даних, можливість їх порівняння та runtime оновлення графіків через вебсокети. Оскільки основою фронтенду є JavaScript-бібліотека React, ми розглянемо, як саме її ключові переваги використовуються для розробки масштабованого, гнучкого та зручного в використанні інтерфейсу. Нижче описано загальну структуру, роботу з маршрутами, взаємодію з вебсокетами та способи візуалізації даних, а також наведено короткий аналіз ефективності кожного компонента з погляду гнучкості та розширюваності.

5.1 Структура інтерфейсу та загальні підходи розробки

Фронтенд-застосунок побудовано за компонентним принципом та складається з трьох ключових модулів інтерфейсу:

1. **SessionList** — компонент огляду, який відображає перелік доступних сесій навчання RL. Він отримує від бекенду базову інформацію про кожну сесію (наприклад, ідентифікатор, назву експерименту, фінальну винагороду тощо) через iREST API при завантаженні сторінки. Цей список дозволяє досліднику вибрати конкретну сесію для детального аналізу. SessionList діє як навігаційний master-component, забезпечуючи огляд експериментів та доступ до їх деталей (рисунок 5.1)
2. **SessionDetail** — компонент детального перегляду результатів окремої сесії RL. Після вибору сесії, фронтенд отримує повні дані цієї сесії з бекенду (наприклад, масив епізодів та відповідних значень винагород, показники втрат моделі, тривалість епізодів тощо) через REST API. SessionDetail відображає ці дані у вигляді інтерактивних графіків і статистичних панелей. Наприклад, типовим є графік залежності середньої винагороди від номера епізоду (крива навчання), гістограми розподілу довжин епізодів або теплові карти політики агента. Компонент забезпечує реактивне оновлення графіків: у реальному часі, по мірі надходження нових даних через WebSocket, крива навчання продовжує будуватися без

перезавантаження сторінки. SessionDetail виступає в ролі дочірнього компонента для SessionList, надаючи більш поглиблене розуміння деталей навчання конкретного агента.

3. **MultiSessionDetail** — компонент порівняння, що дає змогу одночасно аналізувати декілька вибраних сесій. Він відображає обрані метрики (наприклад, криві винагород або інші показники) для кількох сесій на спільному часовому графіку або в єдиній координатній площині. Таке порівняння «пліч-о-пліч» допомагає виявити різницю в поведінці агентів під різними налаштуваннями або алгоритмами. Аналогічний підхід використовується і в наукових інструментах для аналізу багатокомпонентних RL-сценаріїв: наприклад, система MADDPGViz дозволяє зіставляти статистики навчання одразу для декількох агентів та епізодів, щоб виявляти закономірності у їх взаємодії [20]. У нашому випадку MultiSessionDetail виконує схожу роль для кількох незалежних навчальних сесій, полегшуючи порівняльний аналіз ефективності різних підходів.

Кожен із зазначених компонентів є самодостатнім React-компонентом, що взаємодіє з бекендом та, за необхідності, відкриває WebSocket-підключення для отримання оновлень. Взаємодія між компонентами побудована через стан застосунку та властивості (props): вибір сесії в SessionList передає ідентифікатор у SessionDetail, а множинний вибір – у MultiSessionDetail. Така багатошарова архітектура фронтенду забезпечує масштабованість інтерфейсу та гнучкість у додаванні нових представлень даних.

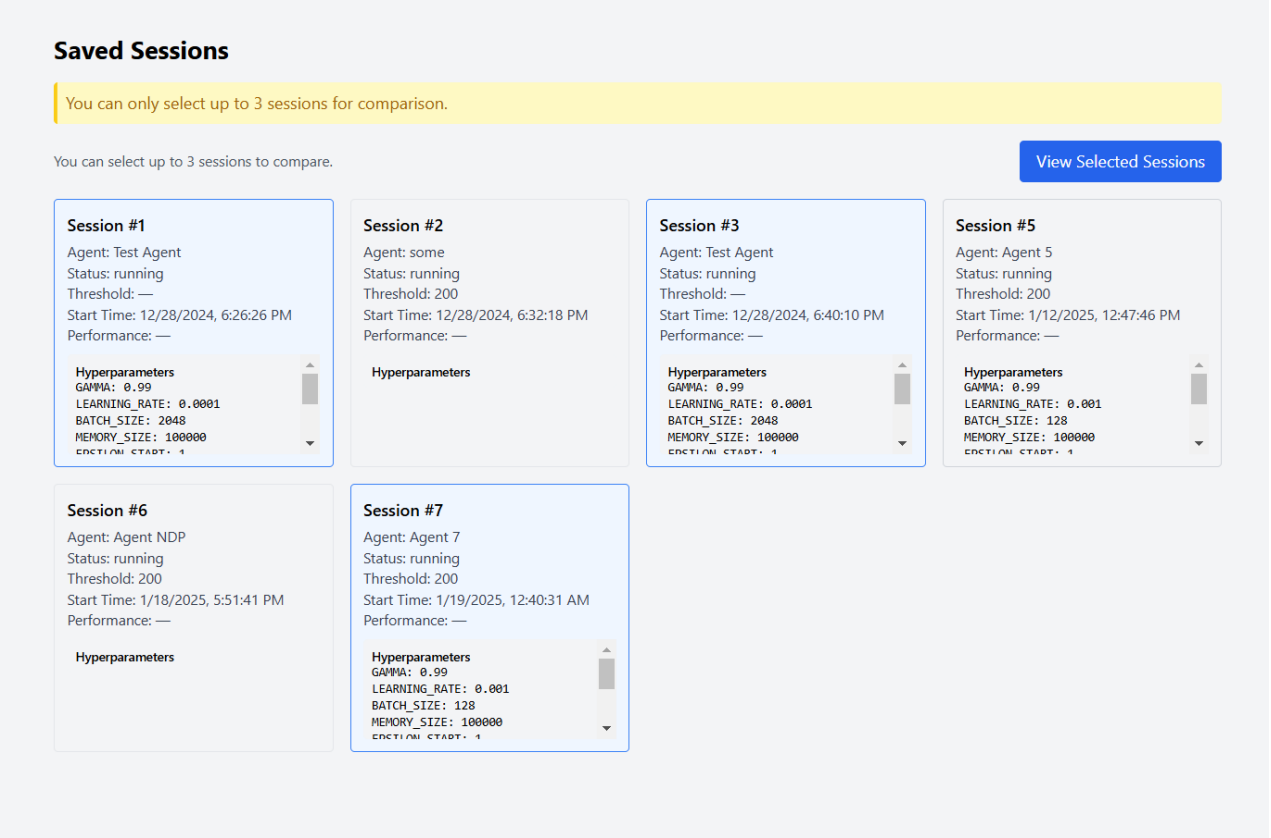


Рисунок 5.1 – Приклад використаного компоненту SessionList, для перегляду списку доступних сесій

5.2 Компонентна модель React і використання Hooks

Фронтенд реалізований на основі React, що дає можливість використовувати компонентно-орієнтовану модель розробки інтерфейсу. Кожен з вищезгаданих модулів (SessionList, SessionDetail, MultiSessionDetail) реалізований як окремий React-компонент функціонального типу. Компонентна модель React сприяє багаторазовому використанню коду та ізоляції логіки: компоненти можна розробляти і тестувати незалежно один від одного. Відмова від монолітного підходу на користь композиції дрібніших компонентів узгоджується з сучасними принципами побудови інтерфейсів, зменшуючи складність системи та покращуючи її підтримуваність.

У реалізації логіки компонентів активно застосовано React Hooks – спеціальні функції, що забезпечують управління станом і побічними ефектами в функціональних компонентах. Зокрема, useState використовується для локального стану компонентів: наприклад, SessionDetail зберігає масив даних епізодів як стан, який змінюється при надходженні нових точок даних. Хук

useState інкапсулює це значення і надає функцію для його оновлення, автоматично ініціюючи повторний рендеринг компонента при зміні стану. useEffect застосовується для керування побічними ефектами – наприклад, для виконання запиту до REST API при монтуванні компонента або для відкриття/закриття WebSocket-з'єднання. Методика використання useEffect дозволяє декларативно визначити, які дії треба виконати при появі компонента або зміні певних даних, і забезпечує очищення (cleanup) ресурсів при демонтажі компонента. Такий підхід замінює імперативні методи життєвого циклу класових компонентів, спрощуючи структуру коду та групуючи пов'язану логіку в одному місці. Крім того, використано хук **useMemo** для оптимізації продуктивності відображення складних графіків. useMemo кешує результат обчислень між повторними рендерами, виконуючи обчислення лише тоді, коли змінилися залежні дані. Це корисно, коли потрібно обробити великий масив даних або виконати складні обчислення для візуалізації: наприклад, обчислення агрегованих метрик по епізодах здійснюється лише при зміні вхідних даних, а не на кожен рендер. Використання useMemo запобігає зайвим повторним обчисленням, тим самим підвищуючи швидкодію інтерфейсу [21]. Загалом, React Hooks надали можливість повторно використовувати стан і логіку між компонентами без ускладнення ієрархії компонентів, що є одним з ключових чинників надійності та розширюваності фронтенду [21].

5.3 Візуалізація даних та інтеграція з Chart.js

Важливим аспектом фронтенду є наочне представлення даних навчання RL агентів. Для побудови інтерактивних графіків обрано бібліотеку Chart.js, яка є найпопулярнішим інструментом для візуалізації даних у веб-застосунках. Chart.js підтримує широкий спектр типів діаграм, зокрема лінійні графіки, гістограми, діаграми розсіювання, кільцеві діаграми тощо, що дозволяє обрати оптимальний спосіб представлення кожної метрики. У контексті аналізу RL агентів найчастіше використовуються лінійні графіки для відображення зміни кумулятивної винагороди чи показників похибки моделі від номера епізоду. Chart.js інтегровано в React-компоненти через створення дочірніх елементів <canvas> для рендерингу графіків. Компонент SessionDetail, наприклад,

створює instance графіка при монтуванні, передаючи йому початкові дані сесії. Бібліотека Chart.js забезпечує гладке оновлення графіка при зміні даних: коли надходять нові значення (через WebSocket або після обчислень), дані в об’єкті графіка оновлюються, і Chart.js автоматично анімує перехід до нового стану графіка. Таким чином, крива навчання на графіку плавно продовжується в реальному часі, що візуально відображає процес навчання без стрибків і перерисовок усієї сцени. Інтерактивне оновлення дозволяє досліднику буквально «спостерігати» за тим, як агент навчається з кожним кроком, що дозволяє зручніше проводити моніторинг експерименту.

Окрім динамічного оновлення, Chart.js надає можливості інтерактивної взаємодії з графіками – такі як наведення курсору для відображення детальних значень точок (tooltips), легенди для фільтрації серій даних, масштабування та прокрутка (рисунок 5.2). У компоненті MultiSessionDetail ці можливості особливо корисні: користувач може ввімкнути/вимкнути відображення окремих кривих для різних сесій, аби зосередитися на одній з них, або порівняти значення винагород в конкретному епізоді, навівши курсор на перетин відповідних кривих. Chart.js як open source бібліотека також забезпечує адаптивність графіків: при зміні розміру вікна або компонуванні панелей, графіки автоматично підлаштовуються, зберігаючи читабельність.

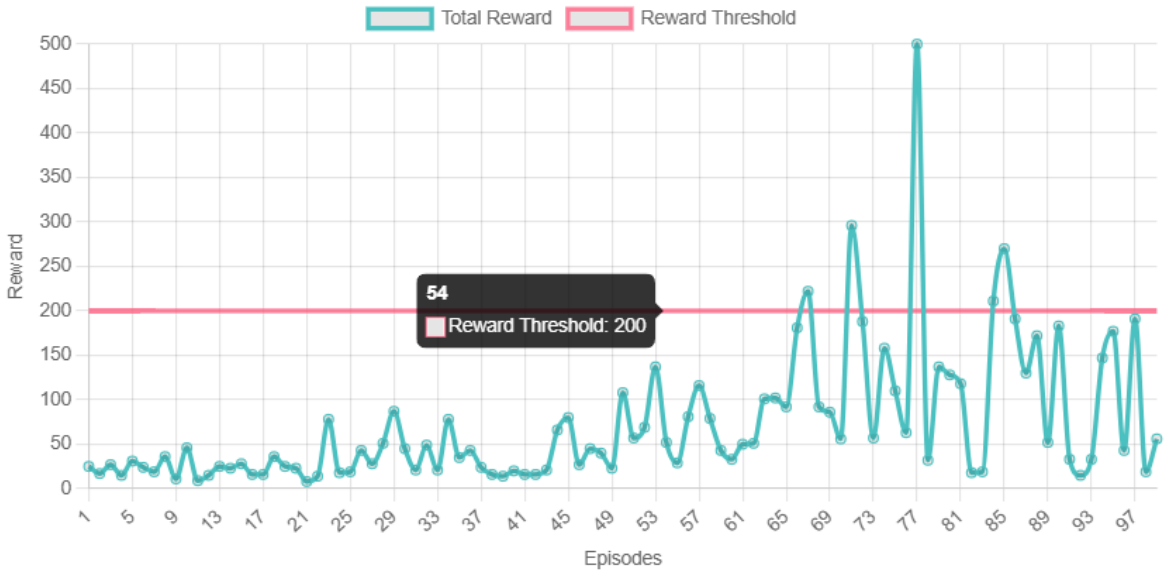


Рисунок 5.2 – приклад графіку тренування агента з використанням Chart.js

вибір Chart.js обґрунтовується його збалансованістю між функціональністю та простотою інтеграції. Існують інші інструменти, наприклад D3.js, що дають ще більшу гнучкість у візуалізації, але потребують більше коду для типової побудови графіка. Chart.js надає готові компоненти, оптимізовані для поширених завдань (лінійні графіки, гістограми розподілів тощо), що дозволило зосередитись на науковій стороні аналізу RL, а не на ручній реалізації графічних примітивів. Підсумовуючи, використання Chart.js дало змогу створити інтуїтивно зрозумілі та естетично привабливі візуалізації, які оновлюються в реальному часі синхронно з ходом експерименту.

5.4 Реалізація оновлення даних в реальному часі через WebSocket

Оскільки RL тривалим є процесом, що відбувається протягом тривалого часу (агент навчається поступово, епізод за епізодом), важливою є передача даних про перебіг навчання в реальному часі від бекенду до фронтенду. Для цього застосовано протокол WebSocket, який дозволяє встановити постійне двонапрямне з’єднання між браузером і сервером. На відміну від традиційного підходу polling або періодичного запиту даних, WebSocket забезпечує миттєве надходження нових даних щойно вони доступні на сервері, без зайвих затримок і накладних витрат на HTTP-запити.

У контексті нашого застосунку, WebSocket-з’єднання відкривається з компонента SessionDetail (або MultiSessionDetail) при його ініціалізації. Це реалізовано через useEffect-хук: при монтуванні компонента виконується встановлення з’єднання new WebSocket(url), де url вказує на спеціальну кінцеву точку (endpoint) бекенду, пов’язану з поточною сесією навчання. Після встановлення з’єднання компонент підписується на події повідомлень (onmessage). Сервер (бекенд системи RL) надсилає повідомлення у форматі JSON щоразу, коли агент завершує черговий епізод або коли оновлюються якісь метрики (наприклад, поточне значення винагороди, прогрес епізоду, стан середовища). Отримавши повідомлення, фронтенд використовує вбудовані механізми React для оновлення стану: наприклад, викликається відповідний setState, що додає нову точку даних до масиву епізодів. React автоматично

перерендерює графік з врахуванням нових даних, про що вже згадувалося в розділі про Chart.js.

Для забезпечення коректної роботи в реальному часі, важливим є належне управління життєвим циклом WebSocket. Компонент при демонтуванні (unmount) має закрити з'єднання (виклик socket.close()), щоб запобігти витоку ресурсів або спробам оновлення стану неіснуючого компонента. Ця логіка також реалізована всередині useEffect – функція очищення закриває з'єднання при зміні сесії або закритті компонента SessionDetail.

Використання WebSocket виправдане з точки зору продуктивності та мережевої ефективності. Дослідження показують, що в задачах моніторингу в реальному часі WebSocket перевершує техніку частих AJAX-запитів за рахунок меншого навантаження та більшої пропускної здатності [22]. Зокрема, підтримуючи постійний канал зв'язку, WebSocket уникає накладних витрат на установлення HTTP-з'єднання для кожного оновлення і може передавати дані частіше та з меншими затримками (рисунок 5.3). В експериментальному порівнянні з технологією AJAX було встановлено, що WebSocket-системи здатні передавати суттєво більше повідомлень за той самий час і споживати при цьому менше пропускної здатності мережі [22]. Ці властивості особливо актуальні для інтерактивного RL-застосунку, де оновлення можуть надходити дуже часто (наприклад, десятки разів за секунду при швидкому навчанні на ранніх епізодах агента).

Практична реалізація WebSocket-інтеграції забезпечує практично миттєве відображення подій навчання. Дослідник бачить, як після кожного епізоду точка на графіку негайно з'являється, і може взаємодіяти зі системою без очікування завершення всього експерименту. Наприклад, якщо виявлено, що криві навчання двох агентів у MultiSessionDetail починають розходитися, дослідник може в режимі реального часу прийняти рішення про коригування гіперпараметрів або дострокове завершення неефективного експерименту. Таким чином, WebSocket-підсистема не лише підвищує зручність, але й слугує інструментом прискорення наукового експерименту шляхом швидкого зворотного зв'язку.

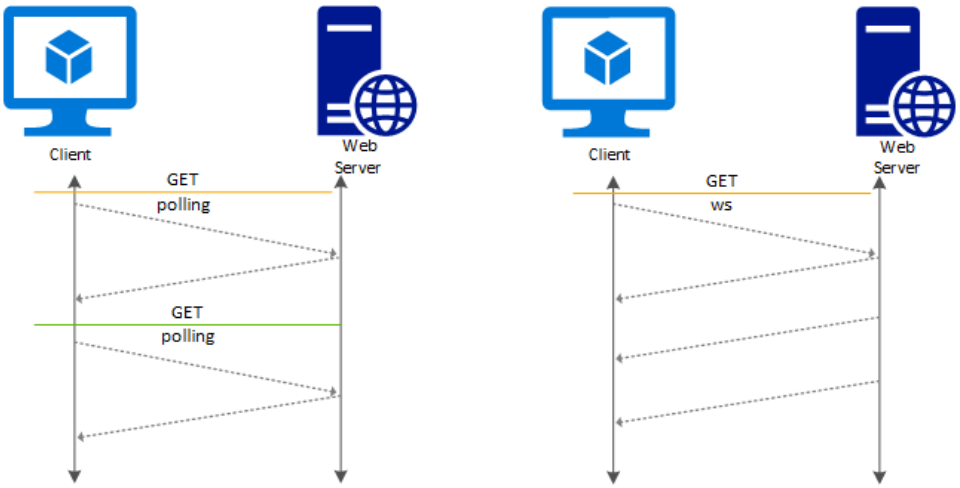


Рисунок 5.3 – Схема роботи web socket, image credit: Alessandro Duarte

5.5 Взаємодія з REST API бекенду

Фронтенд-застосунок спілкується з бекенд-сервером не лише через WebSocket, а й за допомогою стандартного REST API. WebSocket забезпечує потокове надходження даних, проте початкове завантаження інформації про сесії та отримання агрегованих статистик зручніше виконати через одноразові HTTP-запити до REST API. Архітектурний стиль REST (Representational State Transfer) передбачає поділ клієнта і сервера та використання стандартних методів HTTP (GET, POST, PUT, DELETE) для маніпуляції ресурсами[22]. У нашому застосунку бекенд надає REST-інтерфейси для ресурсів Session (окрема сесія навчання) та SessionCollection (список усіх сесій).

При завантаженні компонента SessionList виконується запит GET /sessions до сервера, у відповідь на який надходить JSON-список всіх сесій з їх метаданими. Ці дані використовуються для відображення списку експериментів у таблиці або переліку. Вибір користувачем конкретної сесії призводить до навігації (наприклад, зміни маршруту у SPA) та завантаження SessionDetail, який у свою чергу здійснює запит GET /sessions/{id} для отримання детальної інформації про вибрану сесію (масив історії винагород, параметри середовища, конфігурація агента тощо). Аналогічно, MultiSessionDetail може використовувати спеціальний ендпойнт GET /sessions?ids={id1,id2,...}, який повертає згруповані дані для кількох сесій одночасно, що оптимізує кількість

запитів. Отримані через REST API дані зберігаються у стані відповідних компонентів і візуалізуються Chart.js, як описано вище.

REST API також використовується для керування експериментами. Хоча основна мета фронтенду – візуалізація, інтерфейс може надавати можливості взаємодії, такі як запуск нової сесії навчання, зупинка поточної або зміна параметрів. Такі дії реалізуються через відповідні HTTP-запити (наприклад, POST /sessions для старту нового експерименту, або PATCH /sessions/{id} для зміни параметрів). Завдяки принципу безстанності (statelessness) REST, кожен запит містить всю необхідну інформацію, а сервер не зберігає стану між запитами клієнта. Це спрощує масштабування бекенду і полегшує відновлення з'єднання WebSocket при перезавантаженні сторінки, оскільки фронтенд завжди може повторно отримати актуальний стан сесії через REST. використання REST для початкового завантаження даних і керування ресурсами доповнює WebSocket-потоки, утворюючи збалансовану систему. Такий підхід відповідає загальноприйнятим архітектурним стилям веб-сервісів [22] і відокремлює відповідальність: WebSocket – для push-оновлень і стрімінгу, REST – для запиту стану «на вимогу» та виконання команд. З точки зору надійності, фронтенд передбачає обробку можливих збоїв: якщо WebSocket-з'єднання переривається, застосунок може періодично виконувати запити на REST API як запасний механізм, або спробувати перепідключення. Завдяки стандартності REST-інтерфейсів, інтеграція з бекендом є прозорою і відповідає принципам, закладеним у специфікації REST. Це забезпечує сумісність і можливість уніфікованого доступу до даних як для нашого фронтенду, так і для потенційних інших клієнтів (наприклад, аналітичних скриптів або зовнішніх інструментів).

5.6 Роль фронтенду в наукових дослідженнях

Розроблений фронтенд-інтерфейс виконує не лише утилітарну функцію відображення даних, але й виступає як інструмент наукового аналізу, що підвищує ефективність досліджень у галузі RL. Інтерактивна візуалізація спрощує інтерпретацію поведінки агентів: дослідник може в режимі реального часу спостерігати, як змінюються показники навчання при різних

гіперпараметрах або алгоритмах. Наприклад, порівнюючи на одному екрані (через MultiSessionDetail) графіки навчання для кількох варіантів алгоритму, можна швидко виявити, який з них збігається швидше або досягає вищої винагороди. Це дозволяє оперативніше сформулювати гіпотези і перевірити їх у наступних експериментах. Замість трудомісткого аналізу лог-файлів або статичних графіків після завершення навчання, дослідник отримує інтерактивний та готовий до роботи інтерфейс (рисунок 5.4).

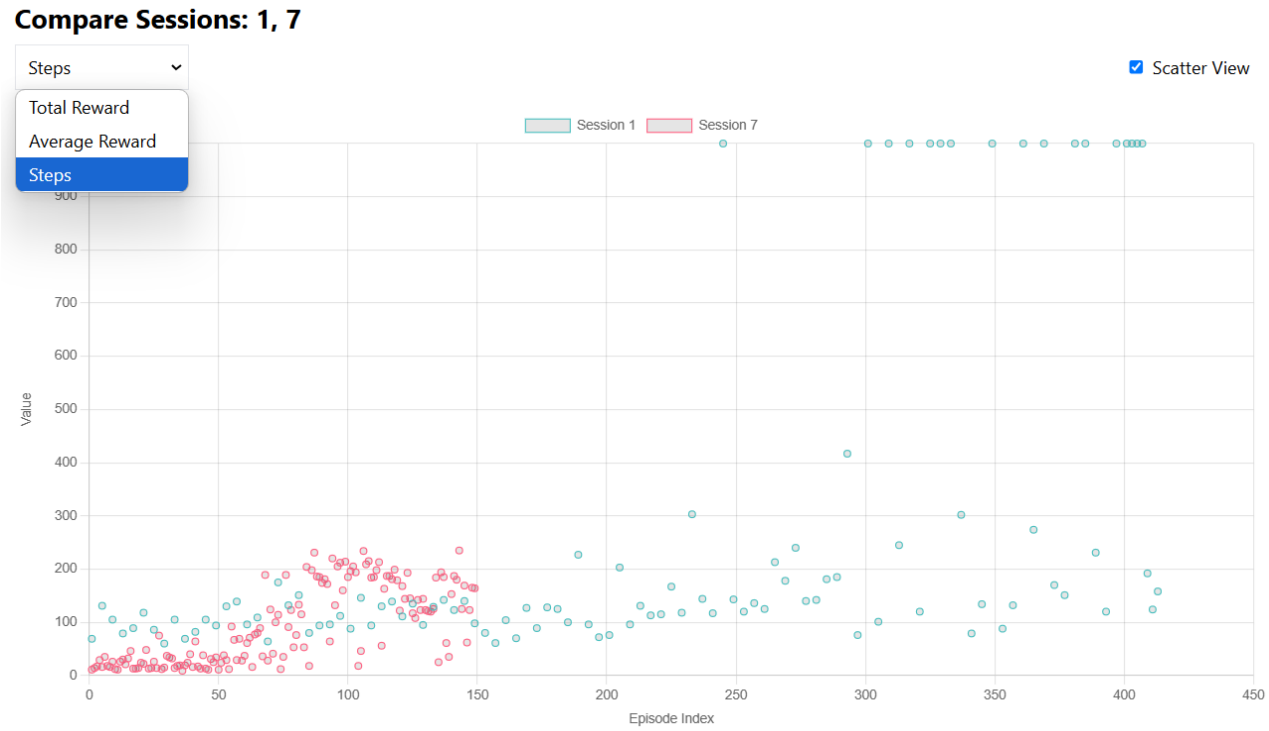


Рисунок 5.4 – Порівняння кількох аентів

Наукова новизна та цінність такого підходу підтверджується сучасними тенденціями у візуальній аналітиці для штучного інтелекту. Нещодавні роботи підкреслюють, що інтерактивні інтерфейси здатні покращити розуміння складних моделей і сприяти генерації нових інсайтів [4] [7]. Зокрема, платформа AutoRL X, представлена IBM Research, продемонструвала, що веб-орієнтований інтерфейс з інтерактивною візуалізацією в режимі реального часу підсилює розуміння користувачами поведінки RL-агентів і полегшує прийняття рішень щодо налаштування алгоритмів. Аналогічно, у роботі Neuwirth & Riley [7] створено інтерактивний дисплей для візуалізації навчання агента у грі Pong,

що допоміг користувачам інтуїтивно осягнути механіку роботи алгоритму і краще налаштувати процес тренування [7]. Наш фронтенд виконує схожу роль: він стає складовою частиною наукового експерименту, надаючи зручний інтерфейс між дослідником і складною RL-системою.

Крім того, наявність такого інструменту підвищує відтворюваність (reproducibility) досліджень. Візуальний огляд кількох сесій допомагає переконатися, що результати стабільні та не є випадковістю. Дослідник може зберігати знімки стану графіків або експортувати дані, на основі яких будується графік, для включення до звітів чи публікацій. Фронтенд також можна розширити модулем для автоматичного генерування звітів (наприклад, у вигляді PDF із графіками та ключовими метриками), що пришвидшує підготовку наукових статей.

Також варто відзначити, що інтерактивний фронтенд сприяє “демократизації” RL-досліджень. Зручний веб-інтерфейс дозволяє навіть менш досвідченим користувачам експериментувати з агентами, спостерігати за їх навчанням і робити висновки, не заглиблюючись у програмний код. Це розширює спільноту дослідників та інженерів, залучених до аналізу RL, і стимулює міждисциплінарні підходи.

5.7 Висновки щодо реалізації клієнтської частини

Розглянутий фронтенд-застосунок для візуалізації та порівняння результатів підкріплювального навчання поєднує в собі сучасні веб-технології та науково обґрунтовані рішення. Компонентна архітектура на базі React (SessionList/SessionDetail/MultiSessionDetail) забезпечує модульність і масштабованість інтерфейсу, а використання Hooks (useState, useEffect, useMemo) спрощує управління станом та побічними ефектами. Інтеграція з Chart.js надає гарні можливості візуалізації даних: різноманітні типи графіків з інтерактивним оновленням в реальному часі допомагають глибоко аналізувати процес навчання RL агентів. Механізм WebSocket дозволив реалізувати низьколатентні оновлення даних, що є критичним для моніторингу швидкоплинних процесів, тоді як REST API забезпечив надійну взаємодію та керування експериментами. Наукова цінність фронтенду проявляється у

прискоренні циклу досліджень та підвищенні наочності: дослідники отримали інструмент, який підсилює інтуїтивне розуміння та сприяє генерації нових гіпотез щодо поведінки агентів RL. Таким чином, поєднання інженерних рішень і наукового підходу при розробці цього інтерфейсу робить його важливим компонентом в екосистемі дослідницьких засобів для RL навчання.

Висновки

У даній курсовій роботі було розроблено комплексну систему для підтримки досліджень у сфері RL на базі Gymnasium, що об’єднує серверну і клієнтську частини у єдину платформу для моніторингу експериментів. Реалізація серверної частини, яка включає запуск експерименту з інтегрованим аспектним перехоплювачем, автоматичний збір статистичних даних, їх збереження та можливість експорту/імпорту результатів, дозволяє створити повноцінний цикл роботи дослідника. Розроблений бекенд забезпечує багаторівневу взаємодію – через REST API, WebSocket та консоль – і надає вбудовані засоби аналізу, зокрема генерацію та візуалізацію графіків, що спрощує порівняльний аналіз сесій.

Клієнтська частина, розроблена з використанням сучасних веб-технологій на базі React, забезпечує інтуїтивно зрозуміле представлення результатів досліджень. Завдяки компонентній архітектурі, використанню Hooks та інтеграції з Chart.js, користувач отримує можливість оперативного аналізу даних через різні типи графічних відображень, а також оновлення інформації через WebSocket. Це сприяє глибшому розумінню динаміки навчання агентів, підвищує прозорість моделей і дозволяє оперативно виявляти проблеми у роботі алгоритмів.

Сумісність розроблених компонентів, їх модульність і можливість динамічного налаштування каналу зв’язку забезпечують уніфікацію процесу проведення експериментів, що є ключовою перевагою даного підходу. Система не лише зменшує втручання дослідника у процес збору даних, але й створює базу для подальшої автоматизації аналізу і оптимізації гіперпараметрів, що значно полегшує цикл «гіпотеза – експеримент – аналіз» у Reinforcement Learning.

Подальша робота

Подальший розвиток проєкту може йти декількома напрямами. Розширення функціональності системи полягає у підтримці ширшого спектра сценаріїв і метрик: наприклад, додавання аналізу для безперервних просторів дій або вбудованих метрик збіжності алгоритму. Перехоплювач можна удосконалити для роботи в багатопоточному та асинхронному середовищі, що дозволить застосовувати його в більш складних експериментах. Крім того, на базі розробленої системи можна створити багатокористувацьку платформу: додати аутентифікацію, розмежування доступу та засоби спільної роботи, що перетворить рішення на повноцінний веб-застосунок для колективного ведення ML-експериментів. Іншою перспективою є інтеграція з хмарними сервісами та платформи Container-orchestration (Docker, Kubernetes) для автоматичного розгортання середовищ навчання та масштабування обчислень. Такі кроки зроблять систему привабливішою для широкого кола користувачів і сприятимуть її еволюції до рівня професійного інструментарію дослідника. Науковий розвиток проєкту може включати поглиблений аналіз зібраних даних: наприклад, додавання модулів для автоматичного виявлення аномалій у поведінці агента чи побудови узагальнених моделей на основі статистики кількох запусків. Запропонована система закладає підґрунтя для таких досліджень, адже забезпечує зручний доступ до структурованих даних експериментів. Загалом, результати роботи мають значний потенціал для подальшого вдосконалення, а їх впровадження та масштабування можуть істотно вплинути на підхід до проведення експериментів у галузі підкріплювального навчання.

Список Літератури

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement Learning: A Survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996, doi: 10.1613/jair.301.

[2] M. Towers *et al.*, “Gymnasium: A Standard Interface for Reinforcement Learning Environments,” Nov. 08, 2024, *arXiv*: arXiv:2407.17032. doi: 10.48550/arXiv.2407.17032.

[3] C. Bamford, M. Jiang, M. Samvelyan, and T. Rocktäschel, “GriddlyJS: A Web IDE for Reinforcement Learning,” Oct. 12, 2022, *arXiv*: arXiv:2207.06105. doi: 10.48550/arXiv.2207.06105.

[4] L. Franke, D. K. I. Weidele, N. Dehmamy, L. Ning, and D. Haehn, “AutoRL X: Automated Reinforcement Learning on the Web,” *ACM Trans Interact Intell Syst*, vol. 14, no. 4, p. 26:1-26:30, Dec. 2024, doi: 10.1145/3670692.

[5] F. Pardo, “Tonic: A Deep Reinforcement Learning Library for Fast Prototyping and Benchmarking,” May 19, 2021, *arXiv*: arXiv:2011.07537. doi: 10.48550/arXiv.2011.07537.

[6] M. W. Hoffman *et al.*, “Acme: A Research Framework for Distributed Reinforcement Learning,” Sep. 20, 2022, *arXiv*: arXiv:2006.00979. doi: 10.48550/arXiv.2006.00979.

[7] A. Neuwirth and D. Riley, “Architecting and Visualizing Deep Reinforcement Learning Models,” Dec. 02, 2021, *arXiv*: arXiv:2112.01451. doi: 10.48550/arXiv.2112.01451.

[8] D. Ghimire, “Comparative study on Python web frameworks: Flask and Django.” Accessed: Mar. 15, 2025. [Online]. Available: http://www.theseus.fi/handle/10024/339796

[9] “Django vs. Flask: Which Is the Best Python Web Framework? | The PyCharm Blog,” The JetBrains Blog. Accessed: Mar. 15, 2025. [Online]. Available: https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/

[10] “Python Programming - The State of Developer Ecosystem in 2023 Infographic,” JetBrains: Developer Tools for Professionals and Teams. Accessed: Mar. 15, 2025. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2023

[11] L. Django, “Flask vs Django in 2024: A Comprehensive Comparison of Python Web Frameworks.” Accessed: Mar. 15, 2025. [Online]. Available: https://learndjango.com/tutorials/flask-vs-django

[12] P. Dymora, M. Mazurek, and M. Nycz, “Comparison of Angular, React, and Vue Technologies in the Process of Creating Web Applications on the User Interface Side,” *J. Educ. Technol. Comput. Sci.*, vol. 34, no. 4, Art. no. 4, Dec. 2023, doi: 10.15584/jetacomps.2023.4.21.

[13] R. Mohammadi, “Performance, and Efficiency based Comparison of Angular and React in a case study of Single page application (SPA)”.

[14] M. Levlin, “DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte.” Accessed: Mar. 15, 2025. [Online]. Available: https://www.doria.fi/handle/10024/177433

[15] "4. Aspect Oriented Programming — Spring Python v1.2.1.FINAL documentation." Accessed: Mar. 15, 2025. [Online]. Available: https://docs.spring.io/spring-python/1.2.x/sphinx/html/aop.html
[16] C. Patel, "Aspect-Oriented Programming (AOP) and Object-Oriented Programming (OOP) concepts," Nov. 09, 2024, *Social Science Research Network, Rochester, NY*: 5015146. doi: 10.2139/ssrn.5015146.
[17] M. Matusiak, "Strategies for aspect oriented programming in Python".
[18] M. Wunder, M. Littman, and M. Babes, "Classes of Multiagent Q-learning Dynamics with -greedy Exploration".
[19] "Django Channels — Channels 4.2.0 documentation." Accessed: May 04, 2025. [Online]. Available: https://channels.readthedocs.io/en/latest/
[20] X. Shi, J. Zhang, Z. Liang, and D. Seng, "MADDPGViz: a visual analytics approach to understand multi-agent deep reinforcement learning," *J. Vis.*, vol. 26, no. 5, pp. 1189–1205, Oct. 2023, doi: 10.1007/s12650-023-00928-0.
[21] "Introducing Hooks – React." Accessed: Mar. 23, 2025. [Online]. Available: https://legacy.reactjs.org/docs/hooks-intro.html
[22] D. G. Puranik, D. C. Feiock, and J. H. Hill, "Real-Time Monitoring using AJAX and WebSockets," in *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, Apr. 2013, pp. 110–118. doi: 10.1109/ECBS.2013.10.
[23] J. D. Co-Reyes *et al.*, "Evolving Reinforcement Learning Algorithms," Nov. 10, 2022, *arXiv*: arXiv:2101.03958. doi: 10.48550/arXiv.2101.03958.
[24] M. Chevalier-Boisvert *et al.*, "Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks," *Adv. Neural Inf. Process. Syst.*, vol. 36, pp. 73383–73394, Dec. 2023.
[25] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," in *Proceedings of The 33rd International Conference on Machine Learning*, PMLR, Jun. 2016, pp. 1329–1338. Accessed: Feb. 22, 2025. [Online]. Available: https://proceedings.mlr.press/v48/duan16.html
[26] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A Research Framework for Deep Reinforcement Learning," Dec. 14, 2018, *arXiv*: arXiv:1812.06110. doi: 10.48550/arXiv.1812.06110.
[27] F. E. F. Junior and A. Oulasvirta, "AgentForge: A Flexible Low-Code Platform for Reinforcement Learning Agent Design," Jan. 09, 2025, *arXiv*: arXiv:2410.19528. doi: 10.48550/arXiv.2410.19528.
[28] S. Huang *et al.*, "Open RL Benchmark: Comprehensive Tracked Experiments for Reinforcement Learning," Feb. 05, 2024, *arXiv*: arXiv:2402.03046. doi: 10.48550/arXiv.2402.03046.
[29] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.
[30] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," Feb. 15, 2017, *arXiv*: arXiv:1611.01578. doi: 10.48550/arXiv.1611.01578.
[31] R. de Kock *et al.*, "Mava: a research library for distributed multi-agent reinforcement learning in JAX," Dec. 15, 2023, *arXiv*: arXiv:2107.01460. doi: 10.48550/arXiv.2107.01460.

[32] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman, “Data-Efficient Reinforcement Learning with Self-Predictive Representations,” May 20, 2021, *arXiv*: arXiv:2007.05929. doi: 10.48550/arXiv.2007.05929.