МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ ЗАСОБАМИ МОДЕЛІ LORA ТА STABLE DIFFUSION МЕРЕЖІ»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
І.В. Ткаченко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.п.н. Пшенична О.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ткаченку Івану Володимировичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка програмного застосунку генерації зображень засобами моделі LoRA та Stable Diffusion мережі

керівник роботи Мухін Віталій Вікторович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та реалізація застосунку для синтезу візуального контенту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	29.03.2024	
2.	Збір вихідних даних.	26.04.2024	
3.	Обробка методичних та теоретичних	31.05.2024	
	джерел.		
4.	Розробка першого та другого розділу.	06.09.2024	
5.	Розробка третього розділу.	13.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ І.В. Ткаченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка програмного застосунку генерації зображень засобами моделі LoRA та Stable Diffusion мережі»: 55 с., 21 рис., 10 джерел, 1 додаток.

ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, AUTOMATIC 1111, CUDA, LORA, STABLE DIFFUSION.

Об’єкт дослідження – полягають у вивченні сучасних методів тренування моделей LoRA (Low-Rank Adaptation) для системи Stable Diffusion, а також у розробці ефективної методики для покращення якості генеративних моделей зображень. Основна увага приділяється адаптації великих мовних моделей до задач генерації високоякісних зображень з використанням обмежених обчислювальних ресурсів.

Мета роботи: розглянути проблему оптимізації процесу навчання моделей LoRA для Stable Diffusion та підходи до зниження обчислювальної складності без втрати якості результатів. Як результат, була розроблена та навчена оптимальна модель, що забезпечує високу якість генерації зображень, зберігаючи при цьому ефективність використання обчислювальних ресурсів.

Метод дослідження – використання методів LoRA для адаптації великих мовних моделей до нових завдань з мінімальними витратами.

SUMMARY

Master's qualifying paper "Development of a Software Application for the Image Generation Using LoRA Model and Stable Diffusion Network": 55 pages, 21 figures, 10 references, 1 supplement.

DEEP LEARNING, NEURAL NETWORK, AUTOMATIC 1111, CUDA, LORA, STABLE DIFFUSION.

Research object – focuses on studying modern methods of training LoRA (Low-Rank Adaptation) models for the Stable Diffusion system, as well as developing an effective methodology to improve the quality of generative image models. The main focus is on adapting large language models for high-quality image generation tasks using limited computational resources.

Purpose of the work: to examine the problem of optimizing the training process of LoRA models for Stable Diffusion and approaches to reducing computational complexity without losing result quality. As a result, an optimal model was developed and trained that provides high-quality image generation while maintaining efficient use of computational resources.

Research method – using LoRA methods to adapt large language models to new tasks with minimal resource consumption.

ЗМІСТ

Завдання на кваліфікаційну роботу ……..2
Реферат ……4
Summary……5
Вступ..... 8
1 Аналіз сутностей та проблеми тренування LoRA для Stable Diffusion..... 11
1.1 Основні концепції LoRA 11
1.2 Опис Stable Diffusion та його використання в генеруванні зображень 12
1.3 Проблема технічного обладнання 13
1.4 Сучасні проблеми власників відеокарт від АМД 14
1.5 Аналіз програмного забезпечення..... 17
1.5.1 GUI для sd-scripts 17
1.5.2 GUI для Stable Diffusion 17
2 Дослідження параметрів SD SCRIPTS та дослідження формулювання датасету для тренування LoRA..... 20
2.1 Підготовка датасету 20
2.1.1 Досліджені загальні рекомендації..... 20
2.1.2 Досліджена загальна структура датасету 21
2.3 Дослідження параметрів sd scripts..... 22
3 Розробка LoRA 37
3.1 Встановлення GUI для для тренування 37
3.2 Підготовка датасету та позначення виду LoRA..... 38
3.2.1 Позначення виду LoRA 38
3.2.2 Підготовка датасету..... 40
3.3 Налаштування параметрів та вирішення проблемми шедулера cosine with restarts для тренування стилю 41
3.3.1 Вирішення проблемми шедулера cosine with restarts для

тренування стилю..... 41
3.3.2 Налаштування параметрів..... 42
3.4 Тестування 44
3.4.1 Тестування та вибір епох 44
3.4.2 Тестування на практиці у самому інтерфейсі 46
Висновки 48
Перелік посилань..... 49
Додаток А Кастомний шедулер, написаний на Python……...51

ВСТУП

Актуальність теми. Останніми роками штучний інтелект та машинне навчання демонструють стрімкий розвиток, впливаючи на різноманітні сфери людської діяльності – від медицини та науки до мистецтва та розваг. Однією з передових технологій у цій сфері стала дифузійна модель Stable Diffusion, яка використовується для генерації високоякісних зображень на основі текстових описів. Однак, незважаючи на її успіхи, залишається багато питань та викликів, пов’язаних з її оптимізацією та адаптацією для різних завдань. У цьому контексті тренування Low-Rank Adaptation (LoRA) для моделей Stable Diffusion набуває особливої актуальності та значення. Генеративні моделі, такі як Stable Diffusion, представляють собою потужні інструменти, здатні створювати зображення, тексти та інші види даних з високим рівнем реалізму. Ці моделі знаходять широке застосування в різних сферах, включаючи дизайн, розробку ігор, створення контенту для соціальних мереж та багато іншого. Однак для ефективного застосування цих моделей у конкретних завданнях часто потрібне їх донавчання або адаптація. Традиційні методи донавчання можуть бути обчислювально затратними і вимагати великих обсягів даних, що робить процес оптимізації моделі складним і дорогим. Модель Stable Diffusion, як одна з передових дифузійних моделей, володіє високою точністю та якістю генерації зображень. Однак її адаптація до конкретних завдань, таких як генерація зображень у певному стилі або під конкретні вимоги замовника, може вимагати значних ресурсів. Використання LoRA для тренування Stable Diffusion дозволяє вирішити цю проблему, забезпечуючи більш гнучкий і економічний процес адаптації моделі. Однією з ключових переваг застосування LoRA для тренування моделей є економічна ефективність. Оскільки метод LoRA вимагає значно менших обчислювальних ресурсів, ніж традиційні методи донавчання, компанії та дослідницькі установи можуть суттєво скоротити свої витрати на навчання моделей. Це особливо важливо в умовах обмежених бюджетів та зростаючих потреб у використанні машинного

навчання. Крім того, зменшення обчислювальних витрат також означає скорочення часу, необхідного для адаптації моделі. У швидкозмінному світі бізнесу та технологій швидкість впровадження нових рішень відіграє критично важливу роль. Використання LoRA дозволяє швидко адаптувати моделі до нових завдань і вимог, що забезпечує конкурентні переваги та сприяє прискоренню інновацій.

Мета і завдання дослідження. Метою цього дослідження є розробка та впровадження методології тренування моделей Stable Diffusion з використанням підходу Low-Rank Adaptation (LoRA) для підвищення ефективності та економічності процесу адаптації моделей до нових завдань.

Об'єкт дослідження. Об'єктом дослідження є процес адаптації дифузійних моделей, зокрема Stable Diffusion, для виконання спеціалізованих завдань з генерації зображень. Дослідження зосереджується на використанні методу Low-Rank Adaptation (LoRA) для підвищення ефективності та зниження обчислювальних витрат при тренуванні та донавчанні цих моделей.

Предмет дослідження. Предметом дослідження є методологія застосування Low-Rank Adaptation (LoRA) для тренування та адаптації дифузійних моделей, зокрема Stable Diffusion. Увага приділяється технічним аспектам реалізації LoRA, його впливу на продуктивність моделей, а також порівнянню з традиційними методами донавчання генеративних моделей.

Методи дослідження. Для вирішення поставленої задачі використовуються наступні методи дослідження:

- аналіз літератури та огляд сучасних підходів: вивчення наукових статей, технічних звітів та іншої літератури для отримання інформації про існуючі методи адаптації генеративних моделей;
- експериментальне моделювання: проведення експериментів з використанням LoRA на різних наборах даних для перевірки його ефективності та продуктивності;
- порівняльний аналіз: порівняння результатів, отриманих за допомогою LoRA, з результатами традиційних методів донавчання моделей;

- статистичний аналіз: використання статистичних методів для оцінки значущості та надійності отриманих результатів;
- обчислювальні експерименти: проведення тестів на різних апаратних конфігураціях для визначення обчислювальних витрат і оптимальних параметрів тренування;
- візуалізація результатів: створення графіків, діаграм та інших візуальних матеріалів для наочного представлення результатів дослідження.

Наукова новизна одержаних результатів. Наукова новизна одержаних результатів полягає у розробці та впровадженні ефективної методології застосування Low-Rank Adaptation (LoRA) для адаптації дифузійних моделей Stable Diffusion. Вперше детально досліджено вплив LoRA на продуктивність та економічну ефективність процесу донавчання моделей, що дозволяє значно знизити обчислювальні витрати і підвищити гнучкість моделей у вирішенні різноманітних завдань генерації зображень. Результати дослідження також відкривають нові можливості для подальших досліджень у галузі генеративних моделей і машинного навчання.

Практичне значення одержаних результатів. Практичне значення одержаних результатів полягає у можливості значного зниження обчислювальних витрат та часу на адаптацію дифузійних моделей, зокрема Stable Diffusion, за допомогою методу Low-Rank Adaptation (LoRA). Запропонована методологія дозволяє ефективно донавчати моделі для виконання спеціалізованих завдань, що сприяє їх широкому застосуванню в різних галузях, таких як дизайн, розробка ігор, створення контенту для медіа та багато іншого. Це сприяє підвищенню продуктивності та зниженню витрат компаній і дослідницьких установ, що займаються розробкою та впровадженням генеративних моделей.

1 АНАЛІЗ СУТНОСТЕЙ ТА ПРОБЛЕМИ ТРЕНУВАННЯ LORA ДЛЯ STABLE DIFFUSION

1.1 Основні концепції LoRA

Низькорозмірні матриці. LoRA використовує низькорозмірні матриці для апроксимації великих матриць ваг у нейронних мережах. Цей підхід зменшує кількість параметрів, які потрібно оптимізувати, що сприяє швидшому та ефективнішому тренуванню [4].

Мінімізація втрат інформації. Незважаючи на зменшення розмірності, LoRA прагне мінімізувати втрати важливої інформації. Це досягається шляхом оптимізації матриць низького рангу таким чином, щоб вони максимально точно відображали початкові ваги моделі.

Модульність та гнучкість. LoRA дозволяє додавати нові матриці низького рангу до існуючих моделей без необхідності повного перенавчання. Це забезпечує гнучкість у використанні моделей для різних задач та даних.

Прискорення тренування. Завдяки зменшенню кількості параметрів, LoRA значно прискорює процес тренування, що є особливо важливим при роботі з великими моделями та великими обсягами даних.

LoRA знайшло своє застосування у різних областях машинного навчання та штучного інтелекту, включаючи обробку природної мови, комп’ютерний зір, генеративні моделі та багато інших.

Однією з головних переваг LoRA є можливість швидкого адаптування великих моделей до нових завдань без необхідності значних обчислювальних ресурсів.

1.2 Опис Stable Diffusion та його використання в генеруванні зображень

Розглянемо **основні етапи** Stable Diffusion.

Прямий процес дифузії. На початковому етапі до зображення додається випадковий шум. Цей процес триває кілька кроків, під час яких зображення стає все більш і більш шумним.

Обернений процес дифузії. У зворотному процесі модель поступово видаляє шум із зображення, відновлюючи його до початкового стану або до заданого стану. Цей процес також відбувається поступово, зменшуючи кількість шуму на кожному кроці.

Наведемо **основні компоненти** Stable Diffusion.

Модель дифузії. Це основна нейронна мережа, яка навчається прогнозувати та зменшувати шум на кожному кроці оберненого процесу. Модель дифузії тренується на великій кількості зображень для точного видалення шуму.

Шумові функції. Вибір функцій для додавання та видалення шуму є критичним для успіху дифузійної моделі. Ці функції мають бути ретельно налаштовані, щоб забезпечити стабільність і якість генерованих зображень.

Розглянемо також основні області **застосування** Stable Diffusion.

Генерація зображень. Stable Diffusion використовується для створення високоякісних зображень з випадкового шуму. Це дозволяє генерувати реалістичні зображення, які важко відрізнити від справжніх.

Редагування зображень. Моделі, побудовані на основі Stable Diffusion, можуть використовуватися для редагування існуючих зображень шляхом додавання або видалення об’єктів, зміни стилю або корекції кольорів.

Трансфер стилів. Одним із популярних застосувань є перенос стилю, де основа зображення залишається незмінною, але стиль (наприклад, живописний) може бути змінений відповідно до заданих параметрів.

1.3 Проблема технічного обладнання

Для тренування моделей LoRA (Low-Rank Adaptation) для Stable Diffusion (SD) 1.5 і SDXL дуже бажано мати графічну карту RTX з мінімум 8 ГБ відеопам’яті для SD 1.5 та 12-16 ГБ відеопам’яті для SDXL. Це обумовлено кількома ключовими факторами, які впливають на ефективність та швидкість тренування моделей. Давайте розглянемо ці фактори докладно.

Великий обсяг даних і складність моделей. Stable Diffusion 1.5 – це складна модель генеративного штучного інтелекту, яка використовує значний обсяг параметрів для генерації високоякісних зображень. Для її ефективного тренування потрібна достатня кількість відеопам’яті, щоб розмістити всі параметри моделі та оброблювані дані.

SDXL ще більш складна модель з більшою кількістю параметрів та шарів. Це вимагає ще більше пам’яті для розміщення вагових параметрів моделі та проміжних обчислень, що виконуються під час тренування.

Обмеження CPU та RAM. Хоча теоретично можливо тренувати моделі штучного інтелекту на центральному процесорі (CPU) з великою кількістю оперативної пам’яті (RAM), практично це буде дуже повільно і неефективно. CPU не оптимізовані для паралельних обчислень, на відміну від графічних процесорів (GPU), які мають сотні або навіть тисячі ядер, здатних одночасно обробляти велику кількість обчислень.

CUDA та оптимізації для NVIDIA. Графічні карти NVIDIA, зокрема серії RTX, мають архітектуру CUDA (Compute Unified Device Architecture), яка дозволяє значно прискорити обчислення завдяки паралельній обробці даних. Багато сучасних бібліотек машинного навчання, таких як TensorFlow та PyTorch, оптимізовані для використання CUDA, що дозволяє ефективніше використовувати ресурси GPU.

Відеопам’ять (VRAM). Для тренування Stable Diffusion 1.5 зазвичай рекомендується мінімум 8 ГБ відеопам’яті. Це пов’язано з тим, що модель та дані повинні одночасно знаходитися в пам’яті GPU. Недостатній обсяг VRAM

призведе до необхідності частої передачі даних між оперативною пам’яттю комп’ютера і відеопам’яттю, що значно уповільнює процес тренування.

Для SDXL рекомендується від 12 до 16 ГБ відеопам’яті через збільшену складність моделі та необхідність обробки більшого обсягу даних. Це дозволяє ефективно завантажувати всі необхідні параметри і виконувати обчислення без надмірного обміну даними між CPU та GPU.

Тривалість тренування. Тренування моделей LoRA та інших моделей глибокого навчання займає багато часу через велику кількість обчислень, необхідних для налаштування вагових параметрів моделі. Навіть на потужних GPU процес тренування може тривати годинами, днями або навіть тижнями, залежно від розміру моделі та обсягу тренувальних даних. Час тренування включає:

- проходження по всіх тренувальних даних: модель має переглянути кожен приклад з тренувального набору даних багато разів (епохи) для того, щоб навчитися розпізнавати закономірності та відповідності;
- обчислення градієнтів і оновлення ваг: для кожного прикладу необхідно обчислити похідні функції втрат (градієнти) і відповідно оновити ваги моделі.

Використання сучасних технологій. Сучасні GPU, такі як серії RTX від NVIDIA, підтримують технології як Tensor Cores та RTX, які оптимізовані для операцій глибокого навчання. Tensor Cores прискорюють матричні обчислення, що є основою для багатьох алгоритмів машинного навчання, а RTX-ядра допомагають у швидкому трасуванні променів для генерації зображень, що може бути корисним у моделях типу SD.

1.4 Сучасні проблеми власників відеокарт від АМД

Тренування моделей LoRA (Low-Rank Adaptation) на графічних картах AMD практично неможливе і не рекомендується з кількох важливих причин.

Ці причини включають обмежену підтримку програмного забезпечення, недостатню оптимізацію бібліотек для архітектури AMD, а також проблеми з продуктивністю і сумісністю.

Обмежена підтримка програмного забезпечення. Основна проблема полягає в тому, що більшість сучасних бібліотек машинного навчання, таких як TensorFlow та PyTorch, значною мірою оптимізовані для роботи з архітектурою CUDA від NVIDIA. CUDA (Compute Unified Device Architecture) – це власницька платформа та програмна модель паралельних обчислень, створена NVIDIA, яка забезпечує значне прискорення обчислень за рахунок використання графічних процесорів.

Графічні карти AMD використовують альтернативні платформи, такі як OpenCL (Open Computing Language) і ROCm (Radeon Open Compute). Хоча OpenCL є відкритим стандартом для паралельних обчислень, його підтримка в бібліотеках машинного навчання обмежена і часто не така оптимізована, як підтримка CUDA. ROCm, розроблений AMD, є аналогом CUDA, але він має обмежену підтримку і сумісність з популярними бібліотеками.

Оптимізація бібліотек. Більшість бібліотек машинного навчання, зокрема TensorFlow і PyTorch, мають добре оптимізовані версії для CUDA. Це означає, що розробники моделей та інженери мають доступ до набору інструментів та оптимізацій, які дозволяють ефективно використовувати ресурси GPU від NVIDIA [3].

Для AMD ситуація інша. Хоча існують версії цих бібліотек, які можуть працювати з OpenCL або ROCm, вони часто не такі ефективні. Наприклад, ROCm підтримує PyTorch, але багато користувачів стикаються з проблемами сумісності та продуктивності. Це призводить до того, що тренування моделей на картах AMD може бути значно повільнішим і менш ефективним.

Проблеми з продуктивністю. Графічні карти NVIDIA серії RTX оснащені Tensor Cores, які спеціально розроблені для прискорення операцій глибокого навчання, таких як множення матриць. Ці ядра дозволяють значно підвищити продуктивність при тренуванні моделей глибокого навчання.

Карти AMD не мають аналогічної апаратної підтримки, що призводить до меншої продуктивності при тренуванні моделей [10].

Графічні карти AMD також можуть мати проблеми з ефективним використанням відеопам’яті (VRAM) при тренуванні великих моделей. Неефективне управління пам’яттю може призводити до перевантаження пам’яті і, як наслідок, до частого обміну даними з оперативною пам’яттю, що значно уповільнює процес тренування.

Проблеми з сумісністю. Платформи ROCm та OpenCL не завжди забезпечують ту ж стабільність і підтримку, що і CUDA. Це може призвести до частих збоїв і проблем з сумісністю під час тренування моделей. Наприклад, користувачі можуть стикатися з проблемами під час налаштування середовища для тренування моделей або під час виконання складних обчислювальних завдань [5].

Більшість дослідників і інженерів зосереджені на використанні GPU від NVIDIA, оскільки вони є стандартом де-факто в індустрії машинного навчання. Це означає, що спільнота, документація та інструменти здебільшого орієнтовані на NVIDIA, що ускладнює пошук підтримки та вирішення проблем для користувачів AMD.

Альтернативні рішення. Хоча використання CPU для тренування моделей є значно повільнішим у порівнянні з GPU, для деяких завдань це може бути альтернативою використанню GPU від AMD. Сучасні процесори можуть обробляти великі обсяги даних, але час тренування буде значно більшим.

Іншим варіантом є використання хмарних обчислень, які пропонують доступ до потужних GPU від NVIDIA. Такі сервіси, як Google Cloud, AWS та Azure, надають можливість орендувати потужні обчислювальні ресурси для тренування моделей машинного навчання.

Тренування моделей LoRA на графічних картах AMD практично неможливе і не рекомендується через обмежену підтримку програмного забезпечення, недостатню оптимізацію бібліотек для архітектури AMD, проблеми з продуктивністю і сумісністю. Графічні карти NVIDIA з їх

архітектурою CUDA, оптимізованими бібліотеками та спеціалізованими ядрами для глибокого навчання забезпечують значно кращу продуктивність та ефективність. Для користувачів AMD можливі альтернативи, такі як використання CPU або хмарних обчислень, але вони також мають свої обмеження і недоліки.

1.5 Аналіз програмного забезпечення

1.5.1 GUI для sd-scripts

Easy-scripts – автоматично встановить всі необхідні залежності та налаштує оточення. Основна гілка давно не оновлювалася, але якщо перейти на дев, будуть (майже) всі останні функції.

Kohya_ss – також позбавить від потреби морочитися з установкою, але працює як WebUI через браузер. У першу чергу отримує всі останні оновлення для тренування, має буквально всі параметри скриптів, а також будь-які інструменти, необхідні для менеджменту LoRA.

OneTrainer – простий у використанні, але досить скромний вибір параметрів; краще вже покопатися в консолі при тренуванні того ж каскаду, оскільки змінити модель для нього неможливо.

LoRA-train-GUI – вимагає окремо налаштованого venv із sd-scripts. Дуже застаріла обгортка, яка вже давно не оновлювалася.

1.5.2 GUI для Stable Diffusion

Automatic1111 Web UI – один з найпопулярніших інтерфейсів для роботи зі Stable Diffusion. Він пропонує безліч функцій та інструментів для генерації, редагування та налаштування зображень.

Серед основних можливостей:

- генерація зображень на основі текстових підказок;
- інтуїтивно зрозумілий інтерфейс користувача;
- можливість збереження та завантаження налаштувань для повторного використання;
- різноманітні налаштування моделі, включаючи вибір і параметри алгоритмів;
- підтримка розширень та додаткових скриптів для розширення функціональності;
- інтеграція з іншими інструментами, такими як img2img для редагування зображень.

InvokeAI (раніше відомий як Deform) – це інший інтерфейс для Stable Diffusion, який відзначається своєю простотою та функціональністю. Основні риси:

- зручний та мінімалістичний дизайн;
- підтримка генерації зображень з текстових підказок;
- підтримка img2img та інших режимів редагування;
- можливість роботи на локальному комп’ютері або в хмарі;
- налаштування параметрів моделі для досягнення оптимальних результатів.

Stable UI – це ще один інтерфейс для роботи зі Stable Diffusion, орієнтований на користувачів, які хочуть швидко почати генерувати зображення без складних налаштувань. Основні характеристики:

- простий і зрозумілий інтерфейс;
- можливість швидкого запуску без необхідності встановлення додаткових компонентів;
- основні функції для генерації зображень на основі текстових підказок;
- обмежені, але ефективні інструменти для налаштування параметрів моделі.

Stable Horde – це сервіс, що дозволяє користувачам генерувати зображення за допомогою Stable Diffusion через розподілену мережу.

Особливості:

– використання потужності кількох комп’ютерів для швидкої генерації зображень;
– можливість працювати без необхідності встановлення локальної версії моделі;
– підтримка різних параметрів та налаштувань для персоналізації результатів;
– зручний вебінтерфейс для подання запитів та перегляду результатів.

ComfyUI – це інтерфейс, який надає можливість користувачам зосередитися на генерації зображень з високою якістю та точністю. Його особливості включають:

– орієнтація на професійних користувачів і дослідників;
– розширені налаштування параметрів моделі для досягнення найкращих результатів;
– підтримка різних режимів роботи, включаючи текстові підказки та img2img;
– інтуїтивний та гнучкий інтерфейс для налаштування процесу генерації.

Кожен з цих інтерфейсів має свої переваги та недоліки, тому вибір залежить від конкретних потреб та досвіду користувача. Automatic1111 Web UI, зокрема, є дуже популярним завдяки своїй гнучкості та великій кількості функцій, що робить його чудовим вибором як для початківців, так і для досвідчених користувачів.

2 ДОСЛІДЖЕННЯ ПАРАМЕТРІВ SD SCRIPTS ТА ДОСЛІДЖЕННЯ ФОРМУЛЮВАННЯ ДАТАСЕТУ ДЛЯ ТРЕНУВАННЯ LORA

2.1 Підготовка датасету

2.1.1 Досліджені загальні рекомендації

Датасет – це основа успішного тренування. Те, наскільки чисто оброблені зображення в ньому, впливає на отриману модель більше всього. Усі залишки чужих кінцівок, незрозумілі об’єкти на зображенні, підписи художників, посилання – все, що ви не хочете бачити у генераціях, краще обрізати або замазати у фотошопі. Роздільна здатність зображень не повинна бути нижчою за ту, з якою ви будете тренувати. Приблизна необхідна кількість зображень, під які не потрібно буде змінювати дефолтні параметри:

– 50-100 для персонажа;
– 75-200 для стилю.

Цифри приблизні і залежать від кожного конкретного датасету та також базуються на досвіді автора. Якщо зібрати мало зображень, мережа буде більш схильна до перенавчання; навпаки, якщо зібрати занадто багато, на тому ж LR мережу доведеться тренувати набагато довше.

Тегування датасету не менш важливе. Якщо правильно підібрати теги, які вже містять інформацію від попередніх тренувань, це значно спростить і прискорить завдання. Якщо тренувати на основі моделей для малювання, опис робити строго в стилі тегів. Наприклад: 1girl, short hair, green eyes, black hair, school uniform... Це саме приклад тих тегів, які вже використовувалися в датасеті NAI. Якщо тренуєте на основі реалізму, в описах пишіть те, що бачите на зображенні. Наприклад, якщо зображення – малюнок бородатого рибалки за роботою, пишіть pencil art of a man fishing, beard; якщо це фото друга в окулярах і червоній сорочці на фоні заходу сонця, пишіть photo of friend_name,

smiling, wearing glasses, red shirt, sunset in the background. Тегувати картинки слід завжди, незалежно від того, чи буде тренуватися енкодер чи ні. Від цього залежатиме кінцевий результат та якість тренування.

Якщо тренувати персонажа або людину, бажано, щоб датасет складався тільки з зображень, де він присутній. Також рекомендується дотримуватися різних стилів у артах, якщо це намальований персонаж. І щоб у всіх описах було ключове слово, яке описує цього персонажа або людину. Наприклад: 1.txt: photo of AndrewFriend, jacket, jeans; 2.txt: photo of AndrewFriend, shorts, t-shirt. Викликати цього персонажа у промпті ви будете відповідно через тег AndrewFriend. Те саме працює і для моделей для малювання – у цьому випадку це буде ім’я персонажа. Наприклад, maria, 1girl, short hair, green eyes тощо. Виклик через maria відповідно. Також, якщо модель вже з самого початку трохи здогадується про вашого персонажа, якого ви будете тренувати за відповідним тегом, це ще більше спростить завдання.

2.1.2 Досліджена загальна структура датасету

Для датасету необхідна певна структура папок, де n – кількість повторень даного концепту; conceptA, conceptB – імена концептів (див. рис. 2.1). Ім’я концепту може бути будь-яким, воно ніде не використовується. Між кількістю повторень і ім’ям концепту обов’язково має бути нижнє підкреслення.

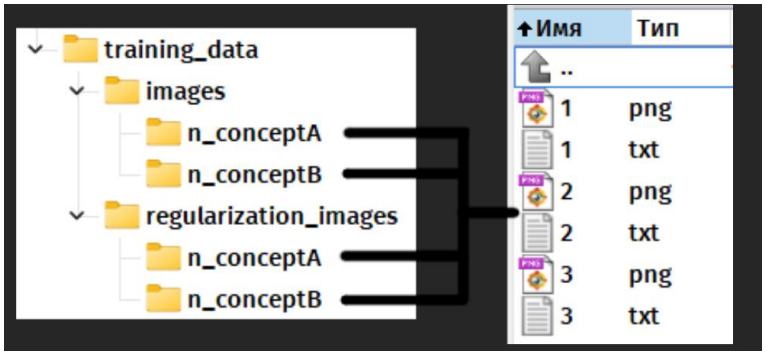


Рисунок 2.1 – Приклад структури папок

Всередині кожної папки концепту повинні бути присутні зображення разом із файлами опису у форматі *.txt, їхні імена повинні збігатися. Всередині текстових файлів має бути опис. Папок концептів може бути скільки завгодно, але хоча б одна повинна бути присутня. Обрізати зображення необов’язково.

Якщо у файлу image.png не буде відповідного image.txt, скрипт видасть йому опис у вигляді імені концепту. Наприклад, якщо папка концепту називається 6_photo, буде вважатися, що у файлу image.png опис photo. Підтримувані формати зображень: *.png, *.jpg, *.jpeg, *.webp, *.bmp.

Повторення потрібні, щоб контролювати кількість часу, витраченого на тренування концептів відносно всього датасету. У випадку з однією папкою достатньо буде просто підігнати кількість повторень під рекомендовані, від 200 до 600 ітерацій на епоху. У випадку з кількома краще утримувати баланс тренованих частин, виставивши повторення відповідним чином. Наприклад, у мене є папка 1_mygirl з 76 зображеннями всередині і папка 6_girloutfits з 16 зображеннями всередині. Сумарно це дасть 172 (1 * 76 + 6 * 16) ітерацій на епоху з цих папок, і нейронна мережа буде навчатися рівномірно кільком концептам.

2.2 Дослідження параметрів sd scripts

Параметр: ***--learning_rate***.

Швидкість навчання. Основний важіль керування навчанням: якщо встановити занадто велике значення, мережа буде навчатися занадто інтенсивно, і результат буде відповідно «пережареним»; якщо встановити занадто мале значення, мережа буде навчатися занадто повільно і довго, що зовсім не підходить для такого швидкого методу навчання, як LoRA. Цей параметр повністю залежить від інших параметрів, а також від кількості зображень у датасеті.

Для оптимайзера AdamW8bit хорошим стандартним значенням є 1e-4 (0.0001) з network_dim=network_alpha=64-128 для 1.5 чекпоїнтів і від 4 до 8e-4

для XL, якщо альфа не сильно нижча за розмірність.

Для адаптивних оптимайзерів, таких як DAdaptation, Prodigy, значення цього параметра необхідно встановлювати в діапазоні на кілька порядків вище, 0.5-2.0, але хорошим дефолтним значенням можна вважати 1.0. Ці оптимайзери за задумом створені так, щоб скасувати цей параметр: з ними він не має такої сили, як з класичними, і керувати швидкістю навчання краще за допомогою параметрів самого оптимайзера або інших параметрів, вплив яких описано далі. У адаптивних оптимайзерів також немає поділу цього параметра на unet_lr і text_encoder_lr, що трохи ускладнює задачу не «спалити» енкодер у процесі.

Параметр: ***--unet_lr***.

Швидкість навчання UNet. UNet – це частина в SD, яка відповідає за поетапне перетворення шуму на зображення. Це можна грубо уявити як графічну складову, яка буде запам’ятовувати нову інформацію. Саме для цього параметра підходять рекомендації, описані в learning_rate.

Параметр: ***--text_encoder_lr***.

Швидкість навчання TE – text encoder. Ця частина SD відповідає за перетворення тексту, який пише користувач у промпті, на математичні значення для подальшого гайдінгу UNet у перетворенні шуму на зображення. Вона менш об’ємна, і, за наявною інформацією, кількість тренованих параметрів у неї майже втричі менша за однакового рангу. Тому хорошим значенням може вважатися unet_lr/2-3, тобто 3.3e-5 – 5e-5, якщо lr UNet дорівнює 1e-4.

Роздільні швидкості навчання для unet_lr і цього параметра на адаптивних оптимайзерах неможливі, тому в такому разі значення потрібно встановлювати таким же, як і learning_rate [9].

Параметр: ***--lr_warmup_steps***.

Кроки розігріву планувальника lr. Скрипт налаштований для обчислення цих кроків у процентному співвідношенні. $lr_warmup_ratio – відсоток кроків, починаючи з першого, протягом яких швидкість навчання лінійно

збільшується від 0 до значення learning_rate. Корисний з низькими значеннями 0.05-0.1, допомагає плавно розпочати тренування з маленького lr і не перенаситити мережу занадто рано.

Параметр: ***--optimizer_type***.

Тип оптимайзера – ще один з найважливіших параметрів, що впливають на те, як доведеться підлаштовувати інші. Існує велика кількість оптимайзерів, включаючи кастомні. Ось кілька з них, взятих із допомоги SD-скриптів: AdamW, AdamW8bit, Lion8bit, Lion, SGDNesterov, SGDNesterov8bit, DAdaptation (DAdaptAdamPreprint), DAdaptAdaGrad, DAdaptAdam, DAdaptAdan, DAdaptAdanIP, DAdaptLion, DAdaptSGD, AdaFactor. Різні їх версії відрізняються різними необхідними параметрами, а також різним споживанням ресурсів GPU. Тут буде описано лише декілька з них, найкорисніших.

AdamW8bit. Король, якого складно перевершити. Практично всі існуючі гіди з тренування LoRA описують саме його параметри і як підлаштовувати все інше під нього. Має низьке споживання VRAM і 8-бітну версію, яка споживає трохи більше, але не додає значно в якості. Просто працює і вважається дефолтом, інші будуть порівнюватися саме з ним. Не потребує особливих параметрів, але якщо дуже хочеться, то такі параметри додаються до нього в Derrian-GUI: weight_decay=0.01, betas=0.9,0.999. Добре підходить для стилів. Все ще актуальний для XL, хоча з learning rate іноді доведеться повозитися і не факт, що з першої спроби LoRA вийде ідеально чистою.

DAdaptAdam. Adam без потреби регулювати learning rate. Має нижчу швидкість ітерацій тренування і підвищене споживання VRAM. Потребує аргументів decouple=True, weight_decay=0.01, betas=0.9,0.99, інакше нічого не буде працювати. З допомогою weight_decay можна регулювати швидкість його навчання: чим вище значення, тим нижча швидкість. На момент написання цього гіда цей оптимайзер є найзручнішим з усіх адаптивних. У минулому він зазнав ряд змін, і найвдаліші версії – це 3.1 і 1.5, у якій, до речі, ще була

можливість розділення learning rate, але не факт, що вона працювала коректно. У версії 1.5 він називався і викликався в скриптах просто як DAdaptation. У цього оптимайзера є деякі проблеми з тренуванням стилів, але в цілому він хороший і використовувати його можна з таким же успіхом, як і звичайний Adam. Він також є недетермінованим, що ускладнює порівняння тренувань з іншими параметрами на ньому. Є думка, що з низьким значенням network_alpha динамічний learning rate з цим оптимайзером може бути надто великим.

Prodigy. Продовження DAdaptAdam, також є адаптивним оптимайзером. Не найвдаліше рішення для 1.5, складно контрольований, потребує тонкого налаштування своїх параметрів, інакше «спалить» все швидше, ніж встигнете подивитися на TensorBoard: decouple=True, weight_decay=0.01, d_coef=1, use_bias_correction=True, safeguard_warmup=True. Все, що актуально для DAdaptAdam, актуально і для нього. Більше схильний до «пережарювання» моделі, ніж його попередник. d_coef – дуже важливий параметр: якщо поставити його занадто низьким, learning rate почне стрибати вище початкового значення посередині тренування. safeguard_warmup можна використовувати разом з warmup, але це не дуже корисно, враховуючи, що і він, і DAdaptAdam мають свій warmup. Для тренувань на XL, особливо тільки UNet, з якими не хочеться морочитися, найкраще вибрати саме цей оптимайзер, але в разі з XL просадка швидкості тренування буде ще сильніше відчуватися, так що тут знову залишається лише вибрати менше зло. Дуже хитро буде запустити тренування з Prodigy, почекати, поки learning rate устаканиться на певному значенні в TensorBoard, знизити його на 15% і запустити тренування вже на Adam зі всіма його бонусами у вигляді швидкості і можливості розділення learning rate для UNet і text encoder.

Lion. За твердженням розробників, заміна AdamW. Насправді дає трохи гіршу якість з параметрами, рекомендованими самими розробниками. Відомо, що він з іншими однаковими параметрами вимагає learning rate у 10-15 разів

нижче, ніж у Adam, тобто 1e-5 – 2e-6. В цілому, оптимайзер для експериментів. Має 8-бітну версію, але для неї потрібні більш свіжі бібліотеки bitsandbytes.

Параметр: ***--optimizer_args***.

Власне ті самі аргументи для оптимайзера: decouple=True, weight_decay=0.01, betas=0.9,0.99.

Параметр: ***--min_snr_gamma***.

Тепер трохи пояснень про timesteps і outliers. Таймстеп – це число, що показує рівень зашумленості зображення, де 0 – повний шум, а 1000 – повністю очищене від шуму зображення. Під час тренування випадковим чином, рівномірно розподілено, обирається число від 0 до 1000, на якому проходить цей конкретний крок тренування для обчислення розбіжностей у передбаченні.

Коли обирається дуже низький таймстеп, нижче 100-150, передбачення виявляються гіршими, а лосс зі стандартною функцією його обчислення вищим, що може направити процес навчання в неправильному напрямку [1]. Сильне і різке відхилення в процесі навчання, яким є більш провальне передбачення на низьких таймстепах, є аутлаєром і може призвести до провального тренування (див. рис. 2.2).

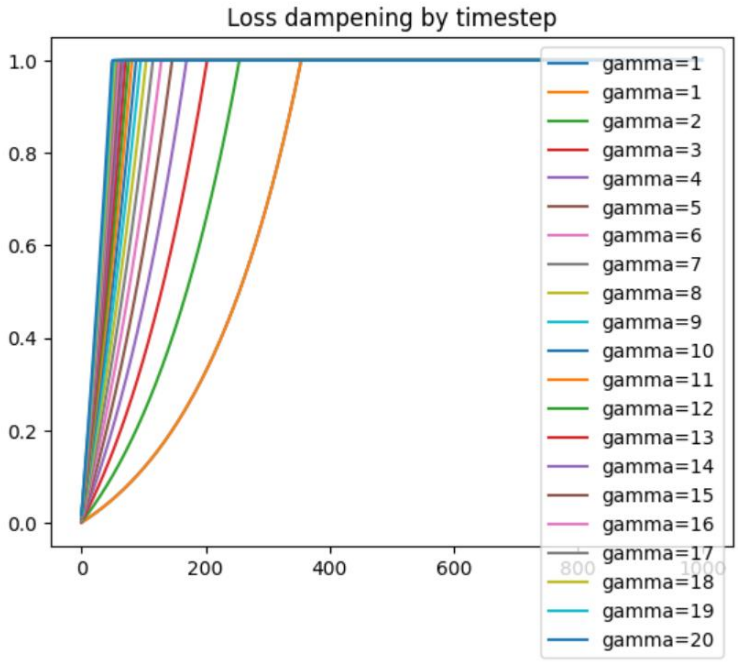


Рисунок 2.2 – Графік дампенінгу впливу нижніх таймстепів

Цей параметр було створено саме для того, щоб надати менше ваги раннім таймстепам. На графіках видно вплив цього параметра на таймстепи, рекомендоване значення в районі 5. Це також є невеликим демпфером і запобіжним заходом для тренування (див. рис. 2.3).

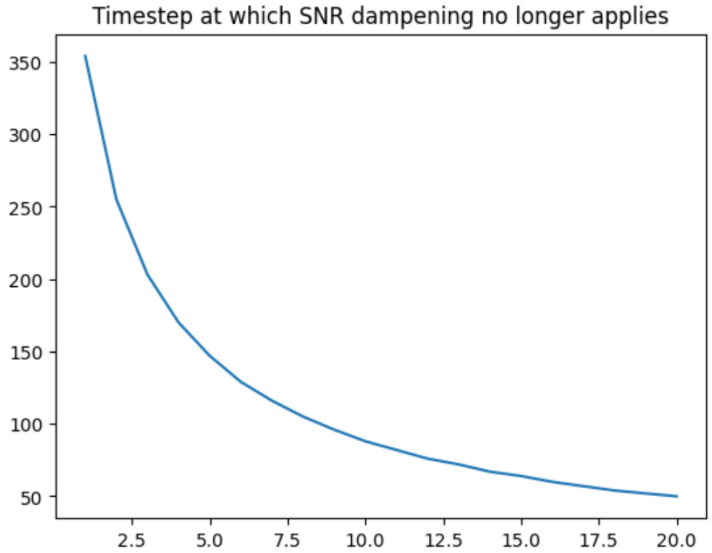


Рисунок 2.3 – Графік, на якому показано на якому таймстепі, дампенінг більше не діє, залежно від значення min_snr_gamma

Параметр: ***--min_snr_gamma***.

Розвиваючи думку про запобігання впливу аутлаєрів на тренування, було додано різні типи передбачення втрат. Відповідно, цей параметр може приймати значення: huber, pseudo-huber, smooth l1 або класичний l2 mse:

– huber добре впливає на загальну композицію, виправляючи вплив нижніх таймстепів, але погано впливає на верхні таймстепи, що викликає погану деталізацію дрібних деталей;
– pseudo-huber пропонує подібну функціональність, але може бути трохи більш гнучким у певних ситуаціях;
– smooth l1 надає компроміс між l1 і l2, забезпечуючи плавніші зміни у втраті, що може бути корисним для стабілізації тренування;
– l2 MSE (mean squared error) є класичним підходом, який добре підходить для більшості ситуацій, але може бути чутливим до аутлаєрів.

Вибір правильного типу передбачення втрат залежить від конкретних вимог до тренування і від того, як ці параметри впливають на різні аспекти моделі.

Параметр: ***--huber_schedule***.

Планувальник втрат актуальний для типів втрат huber і pseudo-huber. Він може приймати значення exponential, constant, snr. За замовчуванням встановлено snr, і найкраще залишити це значення. Це означає, що функція втрат буде контролюватися залежно від співвідношення сигнал-шум (signal-to-noise ratio) або таймстепу.

Параметр: ***--huber_c***.

Параметр, що контролює етап зміни шедулінгу smooth_l1, розробники рекомендують значення 0.1.

Параметр: ***--scale_weight_norms***.

Ще один запобіжний захід, який пропонується використовувати у зв’язці з network_dropout. Він масштабує ключі, які мають занадто великі значення для ваг у них. У консолі виводиться значення підрахунків Average key norm=tensor(0.5194, device='cuda:0') ключів і кількість відмасштабованих Keys Scaled=0. Це може сигналізувати про «пережарку» UNet, якщо ключі починають часто масштабуватися і лічильник Keys scaled зростає. Є думка, що цей параметр додасть LoRA більшу гнучкість для комбінації з іншими параметрами під час інференсу.

Що я помітив особисто, так це те, що використання цього параметра зі значенням 0.9-1.0 самостійно, без dropout, додає ще більше плавності до тренування. Краще не перестаратися зі значенням, інакше мережа нічому не навчиться.

Параметр: ***--network_dropout.***

Викидання випадкових ваг із розрахунків у процесі тренування стимулює треновану мережу створювати нові зв’язки, а не акцентуватися сильно на одних і тих самих значеннях. Це є демпфером і підвищує адаптивність LoRA. Рекомендоване значення – 0.1.

Параметр: ***--lr_scheduler***.

Планувальник кривої learning rate (швидкості навчання). За замовчуванням планувальники з падаючою швидкістю навчання не сильно відрізняються один від одного, але і тут є пара нюансів. Для довгих тренувань стилів найкраще підходять різноманітні косинуси з рестартами/підогрівами. Константний шедулер рекомендується використовувати разом з адаптивними оптимізаторами, тоді як звичайний Адам погано його переварює і потребує постійно падаючого learning rate.

Розглянемо шедулери, які постачаються за замовчуванням з sd-scripts.

Constant: швидкість навчання не змінюється з початку і до кінця навчання. Єдиний планувальник, який не використовує змінну lr_warmup_ratio (див. рис. 2.4).

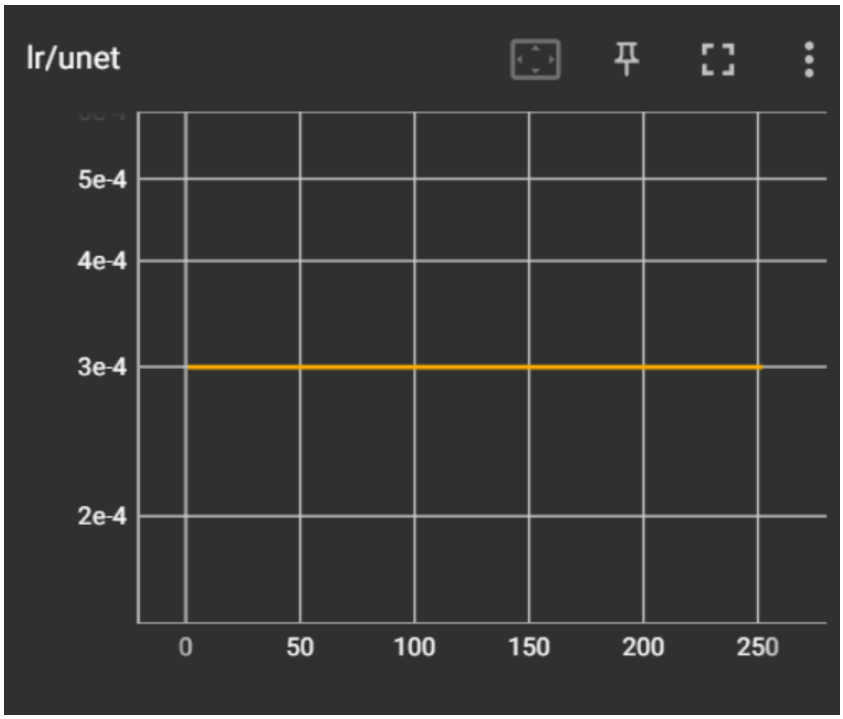


Рисунок 2.4 – Constant

Всі інші планувальники використовують змінну lr_warmup_ratio і мають наступні формулювання.

Після періоду розігріву, що триває lr_warmup_ratio% кроків, протягом якого швидкість навчання лінійно збільшується між 0 і learning_rate.

Polynomial: швидкість навчання зменшується як поліноміальний розпад від значення learning_rate до значення 1e-7 (див. рис. 2.5).

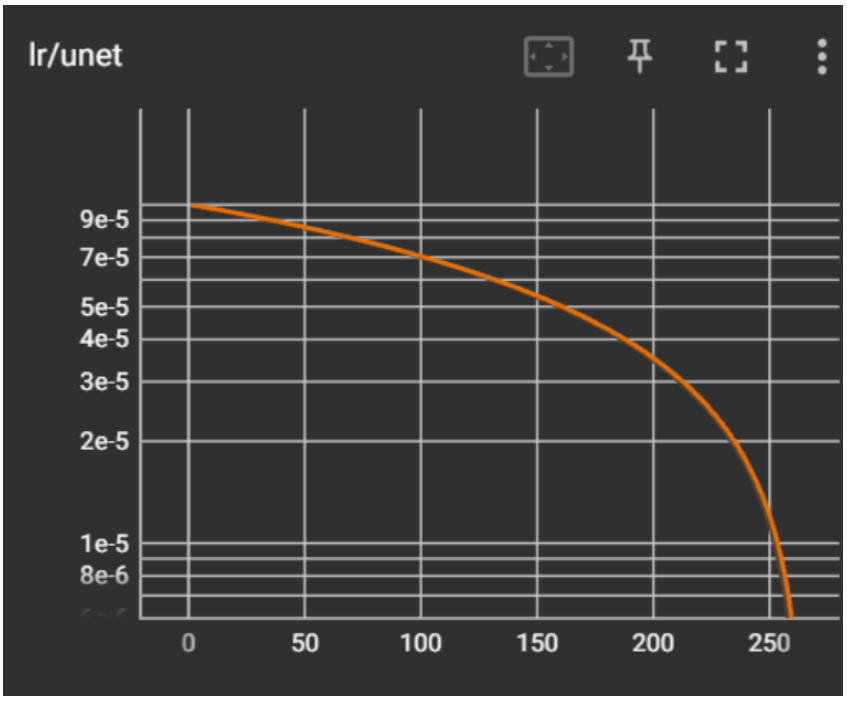


Рисунок 2.5 – Polynomial 0.75 power

Cosine: швидкість навчання зменшується відповідно до значення косинусної функції між learning_rate і 0 (див. рис. 2.6).

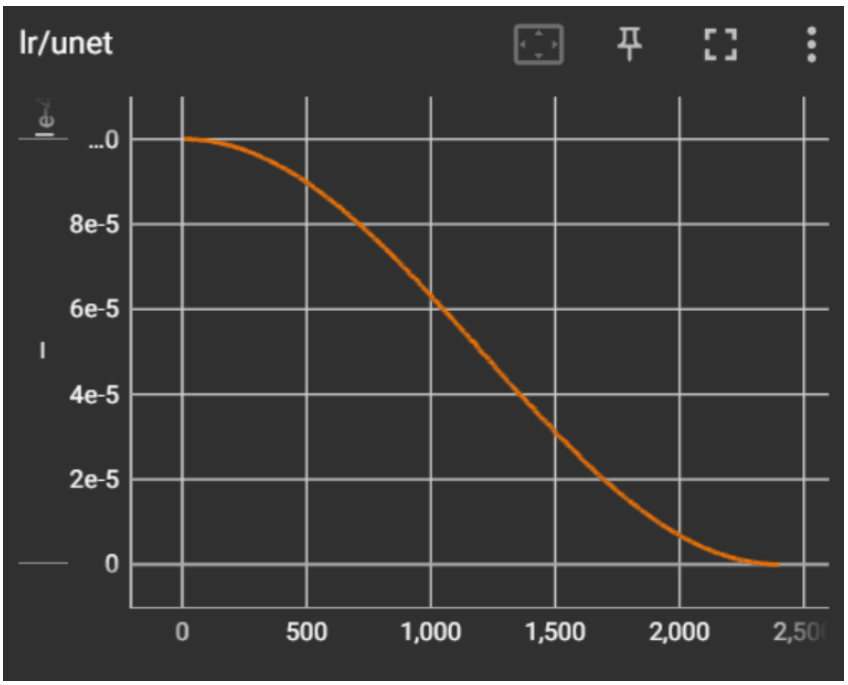


Рисунок 2.6 – Cosine

Linear: швидкість навчання лінійно зменшується від значення learning_rate до 0 (див. рис. 2.7).

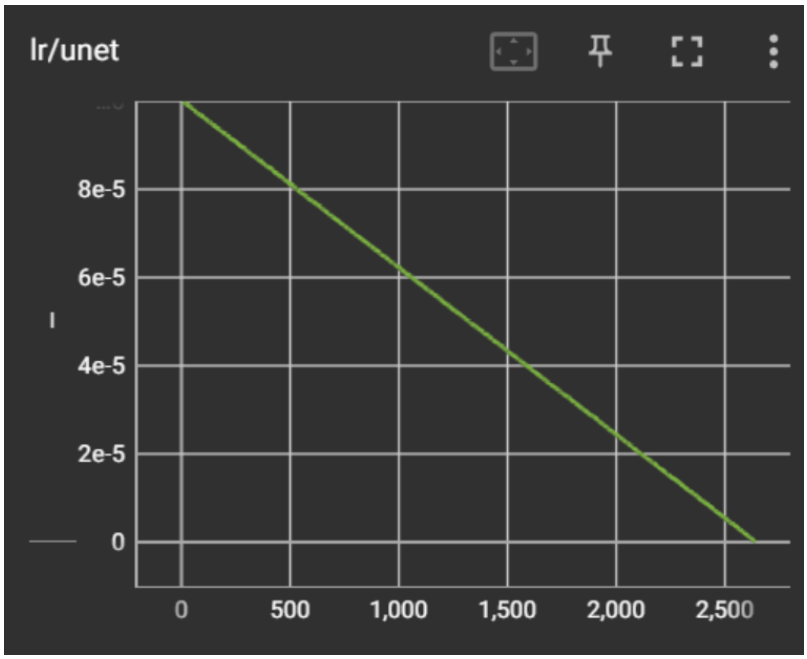


Рисунок 2.7 – Linear

Cosine_with_restarts: швидкість навчання зменшується відповідно до значення косинусної функції між learning_rate і 0, з кількома жорсткими перезапусками (див. рис. 2.8).

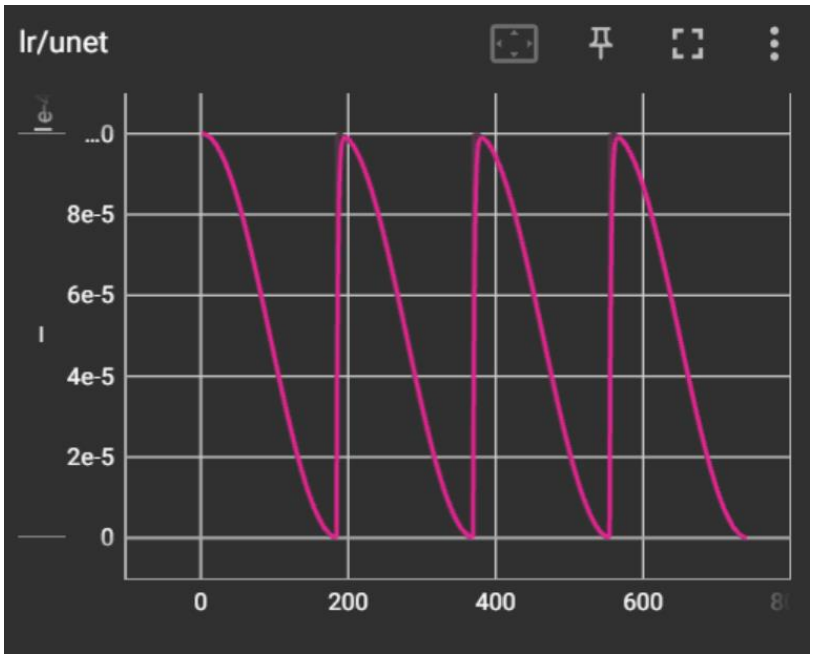


Рисунок 2.8 – Cosine with 4 restarts

Параметр: ***--lr_scheduler_num_cycles***.

Кількість рестартів планувальника cosine_with_restarts. Приклад застосування: --lr_scheduler_num_cycles=4 дасть три рестарти, оскільки саме початок тренування також вважається рестартом.

Параметр: ***--lr_scheduler_power***.

Сила спаду планувальника Polynomial. Чим ближче значення до 0, тим раніше і сильніше почнеться спад.

Параметр: ***--lr_scheduler_type***.

Можливість додати кастомний шедулер прямо через аргумент, наприклад: lr_scheduler_type=cosine_annealing_warmup.CosineAnnealingWarmupRestarts.

Параметр: ***--lr_scheduler_args***.

Аргументи кастомного шедулера. Наприклад: --lr_scheduler_args "T_0=549" "gamma_min_lr=0.99915" "decay=1" "down_factor=0.5" "warmup_steps=100" "cycle_warmup=50" "init_lr_ground=True".

Параметр: ***--network_dim***.

Розмір мережі. Параметр, що визначає кількість тренованих параметрів UNet і TE: чим вище значення, тим більше параметрів буде оновлюватися на кожному проході. Є одним із параметрів, що впливають на швидкість навчання. При низьких значеннях він її уповільнює, але не так значно, як інші.

Також впливає на розмір вихідного файлу та обчислюється за формулою:

– network_dim x 1,125 при половинній точності (FP16, BF16) для лінійних і конволюшен шарів 1.5 чекпоінтів;
– network_dim x ~6,96 для лінійних шарів XL.

Впливає на споживання VRAM під час тренування: чим менше значення, тим менше споживання. Значення вище 128 є надлишковими; достатнім значенням буде 64 для моделей 1.5 або 16 для XL.

Параметр: ***--network_alpha***.

Параметр, спочатку введений для боротьби з андерфлоу вихідних ваг, коли ваги, близькі до нуля, округлялися до останнього і записувалися в тензор.

Працює через масштабування за формулою network_alpha / network_dim. Для тих, хто може використовувати переваги BF16 точності, цей параметр потрібен насамперед тільки для уповільнення тренування. Є другим за силою уповільнювачем тренування і погано впливає на навчання стилів. У разі з FP16 не повністю прибирає нульові значення, а лише знижує їх кількість: чим більше співвідношення, тим менше буде нульових тензорів.

При зниженні цього параметра потрібно компенсувати швидкість навчання підвищенням learning rate у разі з не адаптивними оптимайзерами. Приблизне рекомендоване значення для alpha=1, розмірності мережі=64 – це підвищення lr UNet у 10 разів, до 1e-3.

Хороші значення:

– від 1/4 до 1/1 від розмірності мережі – для тренування стилів;
– від 1/128 до 1/16 – для тренування простих персонажів.

У разі тренувань LoRA на SDXL цей параметр можна знижувати нижче одиниці. Наприклад, для розмірності мережі 8 співвідношення 1/128 для персонажів стане 0.0625 результатом для цього параметра.

Параметр: ***--mixed_precision***.

Вибір цього параметра залежить від наявної GPU для тренування. Якщо у вас є RTX 3000 і вище, обидва значення краще ставити в BF16. FP16 підійде для власників інших карт, але цей формат має занадто мало біт на експоненту, і через це в кінцевій LoRA може вийти дуже багато нульових тензорів.

Параметр: ***--fp8_base***.

Опція, що заморожує ваги основної моделі з точністю FP8 (формат e4m3) на час тренування, значно знижуючи споживання VRAM. Корисна лише в одному винятковому випадку: коли потрібно натренувати XL LoRA, але є лише 8 ГБ VRAM або менше. Не рекомендується використовувати цей параметр в інших випадках, оскільки він зануляє деяку частину значень ще на етапі заморожування ваг.

Параметр: ***--full_bf16.***

Опція для зниження споживання VRAM. Розробники рекомендують використовувати її на свій страх і ризик, оскільки вона була недостатньо

протестована. Вона працюватиме тільки з новими версіями bitsandbytes (0.41.1 і вище), що дещо ускладнює її використання в старих середовищах, особливо враховуючи, що версії між 0.35 і 0.41.1 мають проблеми при використанні на Windows. Також ця опція вимагає встановлення значення bf16 для параметра --mixed_precision.

Параметр: ***--resolution***.

Роздільна здатність тренування. Найголовніший підвищувач якості тренування. Кожні +64 пікселі дають великий приріст дрібних деталей і переносимість стилю. Підвищувати слід насамперед, враховуючи свої можливості VRAM, тому що цей параметр водночас є найсильнішим уповільнювачем тренування.

По-перше, більше роздільна здатність знижує швидкість ітерацій тренування. По-друге, прояв результатів тренування на однакових кроках буде набагато слабкішим. При підвищенні з 512 до 768 потрібне мінімум удвічі більше кроків. По-третє, через більше споживання VRAM доводиться використовувати набагато менший batch size, що ще більше уповільнює тренування.

Рекомендовані значення:

– 768 для тренувань на моделях з 1.5 чекпоінтами;
– 1024 для XL.

Не рекомендується знижувати роздільну здатність для XL, оскільки деградація якості буде дуже помітною, навіть більшою, ніж при тренуванні з вагами FP8. Для XL це також не є демпфером, оскільки базовий чекпоінт уже був натренований з такою роздільною здатністю.

Параметр: ***--train_batch_size***.

Кількість зображень, з яких одночасно буде обчислюватися середнє значення для оновлення градієнта і ваг. Рекомендується ставити максимально можливе значення, яке дозволяє VRAM, оскільки кожне додаткове зображення суттєво збільшує її споживання. Для карт з низьким VRAM значення вище 1-2 недосяжні, тому кожне зображення датасету буде впливати на результат сильніше, ніж одразу кілька семплів, як у випадку з картами на

24 ГБ. Це не є чимось хорошим, але якщо датасет добре підготовлений, то це не страшно. Є помилкова думка, що цей параметр вимагає підвищення learning rate для компенсації, але це не так. Достатньо лише не ставити його занадто великим, якщо датасет недостатньо великий і різноманітний. Наприклад, batch size 5 і 15 зображень буде поганим вибором.

Для XL достатньо і batch size у 2, оскільки більші batch size будуть тренуватися занадто довго для такого ж кількості оновлень ваг.

Параметр: ***--gradient_accumulation_steps***.

Імітатор batch size для тих, у кого недостатньо VRAM для повноцінного batch size. Об'єднується з batch size наступним чином: bs x ga. Загальна формула для оновлень ваг таким чином перетворюється на: (кількість зображень×повтори×епохи)/(bs×ga);

Це дозволяє використовувати імітацію більшого batch size без необхідності додаткової VRAM.

Параметр: ***--noise_offset***.

Жахливий костиль, за допомогою якого можна вирішити проблему дистрибуції шуму для генерації дуже темних або дуже світлих зображень за прийнятну кількість кроків семплера. Жахливий він тому, що при комбінації з іншою моделлю або LoRA, натренованою таким способом, зображення може повністю «згоріти».

Параметр: ***--pretrained_model_name_or_path***. Шлях до чекпоїнту з якого відбуватиметься тренування.

Параметр: ***--output_dir***. Шлях яким зберігатимуться натрені лори.

Параметр: ***--output_name***. Ім'я збереження файлів лор.

Параметр: ***--train_data_dir***. Шлях до датасету.

Параметр: ***--max_token_length***. Максимальна довжина для токенів в описах до зображень. Достатньо просто одразу встановити значення 225 і більше не турбуватися про цей параметр.

Параметр: ***--max_train_epochs***. Кількість епох (повторів усього датасету), протягом яких буде тренуватися мережа.

Параметр: ***--max_train_steps***.

Кількість кроків, протягом яких буде тренуватися мережа. Виставляти потрібно або це, або епохи. Епохи, тим не менш, все одно будуть зберігатися після кожного повного проходу по датасету.

Рекомендації щодо граничної кількості ітерацій на зображення до поділу цього числа на batch size, з урахуванням усіх увімкнених поліпшувачів тренування та рекомендації щодо кількості зображень:

– від 8000 до 12000 для стилів. Від 400 до 600 ітерацій на епоху;
– від 2000 до 7000 для персонажів, залежно від складності персонажа, грамотного підбору тегів і обізнаності моделі про нього. Від 300 до 500 ітерацій на епоху;
– від 1500 до 6000 кроків, залежно від інших параметрів при тренуванні XL у низьких рангах.

Параметр: ***--save_every_n_epochs***. Як часто зберігати епохи? Краще встановити 1, щоб зберігати кожну.

Параметр: ***--save_last_n_epochs***.

Кількість останніх епох, які зберігатимуться. Щоб раптом нічого не вдалося краще ставити значення більше, ніж буде всього натренованих епох.

Параметр: ***--cache_latents*** та ***--cache_latents_to_disk***.

Можуть допомогти зберегти ще трохи VRAM, особливо на початку тренування. Але якщо кешувати на диск, можуть виникнути різного роду конфлікти в подальшому. Тому, якщо бачите якісь дивні помилки під час тренування, перш за все краще видалити ці .npz файли з датасету.

Параметр: ***--enable_bucket, --min_bucket_reso, --max_bucket_reso***.

Автоматичний кроп під обране роздільне здатність тренування зображень і розподілення їх у «бакети». Горизонтальні зображення підуть у 896x640 та подібні, якщо вибрано роздільну здатність тренування 768, вертикальні – навпаки, у 640x896 та подібні. Краще встановити великий діапазон, наприклад, --min_bucket_reso=256 --max_bucket_reso=2048. Якщо в датасеті будуть переважати, наприклад, вертикальні зображення, це може позначитися на кінцевій якості моделі під час генерації горизонтальних зображень.

3 РОЗРОБКА LORA

3.1 Встановлення GUI для для тренування

Як GUI я буду використовувати LoRA Easy Training Scripts.

LoRA Easy Training Scripts – це набір навчальних скриптів, написаних на Python для використання в SD-скриптах. Він має інтерфейс користувача, написаний на pyside6, що допомагає спростити процес навчання моделей (див. рис. 3.1).

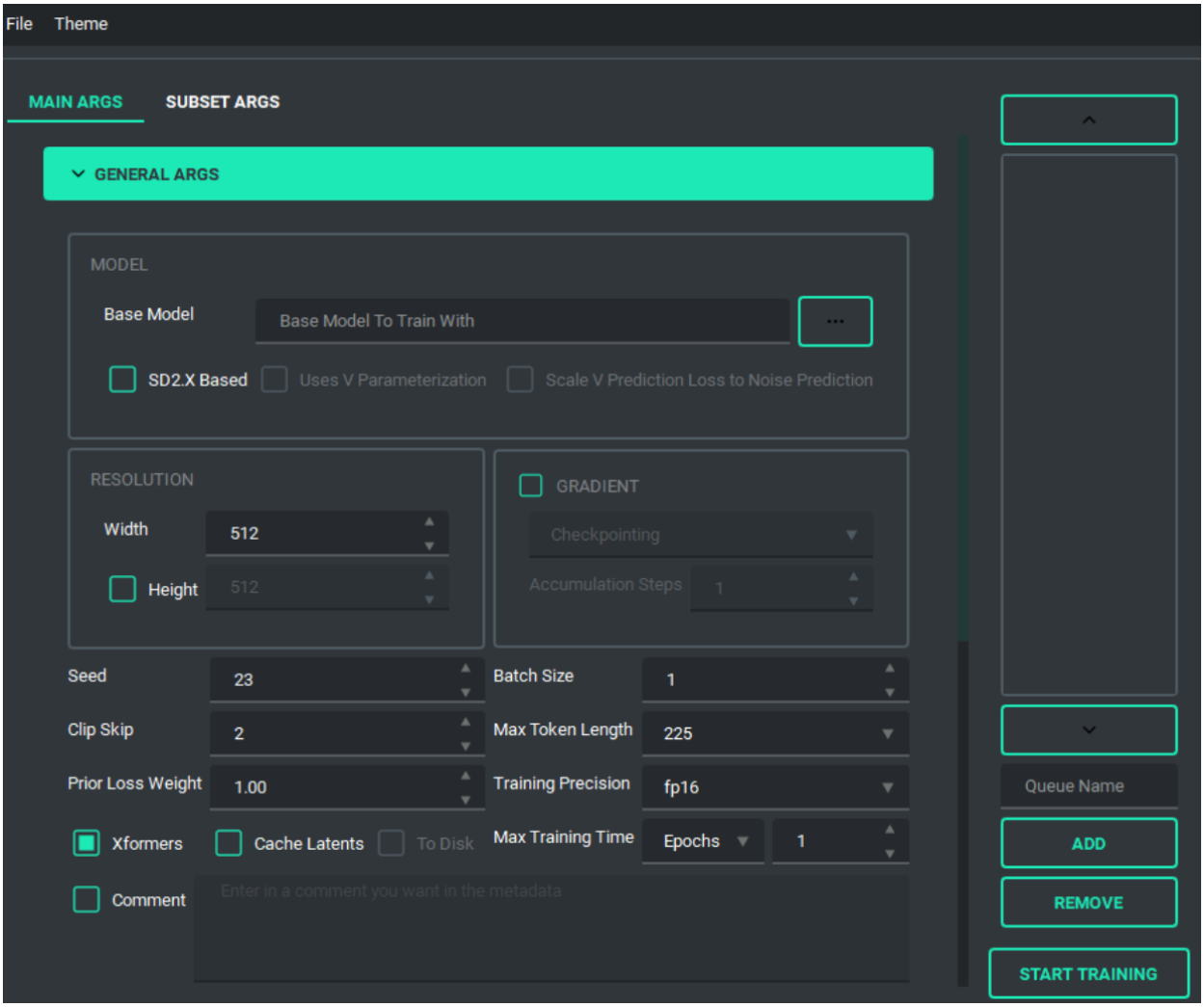


Рисунок 3.1 – GUI LoRA Easy Training Scripts

3.2 Підготовка датасету та позначення виду LoRA

3.2.1 Позначення виду LoRA

Існує кілька типів лор:

– концепт, наприклад навчання на певну позу чи рух;
– стиль, наприклад навчання на імітацію стилю малювання певного художника;
– об’єкт, наприклад навчання на певну людину або марку автомобіля.

Я навчатиму лору на стиль мого знайомого художника. Оскільки тренування LoRA (Low-Rank Adaptation) для стилю художника є однією з найскладніших задач у сфері штучного інтелекту та машинного навчання через кілька важливих причин, які об’єднуються у складний процес, що потребує великих обсягів даних, спеціалізованих знань та значних обчислювальних ресурсів.

Унікальність художнього стилю. Кожен художник має свій унікальний стиль, який може включати різні елементи, такі як:

– колірна палітра: використання специфічних кольорів і їх комбінацій;
– техніка мазків: як художник наносить фарбу на полотно (розмашисті мазки, тонкі лінії тощо);
– теми та мотиви: часто повторювані образи, сюжети або символи;
– текстура та об’єм: особливості передачі текстури і об’єму на двовимірному полотні.

Ці особливості важко формалізувати у вигляді математичних моделей, оскільки стиль – це не просто набір фіксованих правил, а скоріше сукупність тонких і часто інтуїтивних рішень, прийнятих художником. Тому для успішного навчання моделі потрібно дуже глибоке розуміння цих тонкощів.

Великий обсяг даних. Для тренування LoRA потрібен великий обсяг високоякісних даних, які б точно відображали стиль художника. Це включає:

– зображення в високій роздільній здатності: щоб захопити всі деталі та нюанси стилю;

– різноманітні приклади: роботи художника повинні охоплювати широкий спектр його творчості, щоб модель змогла навчитися універсальним характеристикам стилю, а не конкретним зображенням.

Збір таких даних може бути дуже трудомістким і дорогим процесом, особливо якщо мова йде про рідкісні або мало відомі роботи.

Спеціалізовані знання. Розробка моделей машинного навчання для художніх стилів потребує не лише знань у сфері програмування та математики, але й розуміння мистецтва:

– аналіз стилю: вміння розпізнавати та описувати художні особливості, які необхідно перенести на модель;
– мистецька критика: розуміння культурного і історичного контексту робіт, щоб адекватно оцінити і врахувати їх у моделі.

Ці знання рідко поєднуються в одній особі, тому команда, яка працює над проєктом, повинна бути міждисциплінарною.

Обчислювальні ресурси. Тренування моделей LoRA є обчислювально інтенсивним процесом, особливо коли йдеться про високоякісні зображення:

– висока роздільна здатність: чим більше деталей в зображенні, тим більше обчислювальних ресурсів потрібно для обробки;
– складність моделей: щоб адекватно захопити всі нюанси стилю, часто потрібні складні архітектури нейронних мереж [6, 7].

Це потребує значних фінансових вкладень в потужні комп’ютери або оренду обчислювальних ресурсів у хмарі.

Перенавчання та узагальнення. Однією з найважливіших проблем у тренуванні стилей є перенавчання:

– перенавчання: модель може занадто добре запам’ятати конкретні приклади з навчальної вибірки, втративши здатність узагальнювати стиль на нові зображення;
– балансування: необхідно знайти баланс між точністю і здатністю моделі узагальнювати стиль на нові роботи, що потребує ретельного налаштування гіперпараметрів і вибору архітектури.

3.2.2 Підготовка датасету

Насамперед, я зібраю достатню кількість зображень у моєму випадку це буде 101 картинка. Далі я створюю текстові файли з текстовими підказками.

Для того щоб спростити собі завдання для створення текстових підказок я використовував нейромережу для розпізнавання зображень DeepDanbooru.

DeepDanbooru – це нейронна мережа для автоматичної класифікації тегів на зображеннях у стилі аніме. Вона використовує глибоке навчання для розпізнавання та призначення тегів до зображень, що дозволяє автоматизувати процес маркування великої кількості картинок.

Проєкт заснований на глибоких згорткових нейронних мережах, які добре підходять для обробки зображень (див. рис. 3.2).

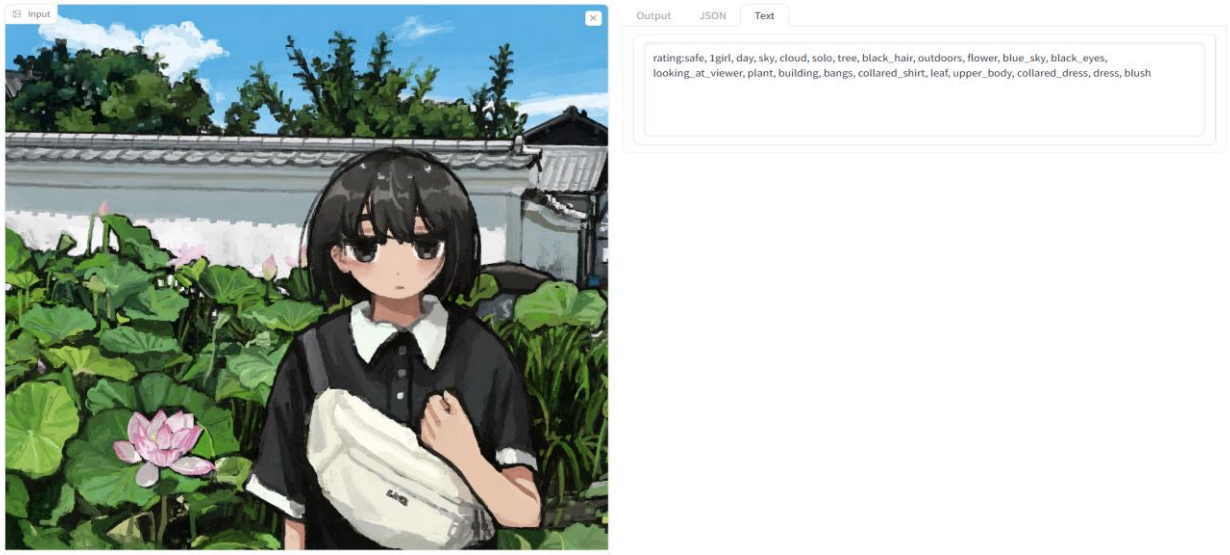


Рисунок 3.2 – Приклад зображення з датасету та розпізнавання DeepDanbooru

Таким чином, я автоматизував процес і дуже швидко склав датасет (див. рис. 3.3 – 3.4).

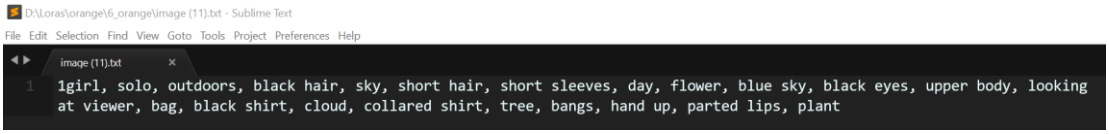


Рисунок 3.3 – Приклад файлу текстової підказки

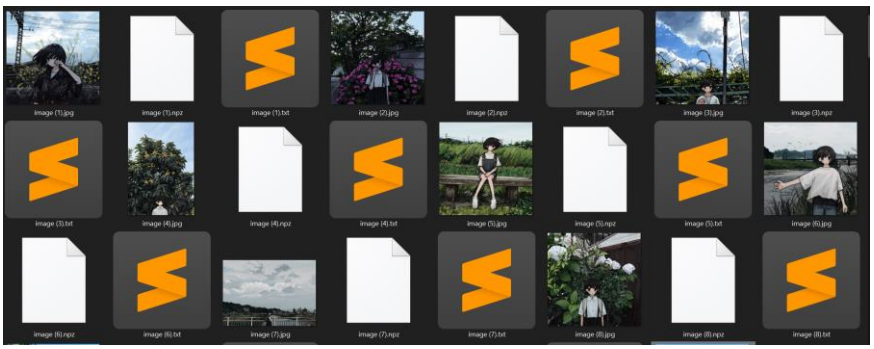


Рисунок 3.4 – Приклад датасету

3.3 Налаштування параметрів та вирішення проблеми шедулера cosine with restarts для тренування стилю

3.3.1 Вирішення проблеми шедулера cosine with restarts для тренування стилю

Стилі за допомогою LORA тренуються найдовше, і так вийшло, що найкращим шедулером для цього є cosine_with_restarts, але у нього є один недолік. Моментальне підвищення learning rate за один крок з 0 до значення, вказаного в learning_rate. Виходило так, що протягом однієї епохи learning rate буває занадто маленьким (нижче 1e-6), щоб мережа могла чогось навчитися, а після цього різке підвищення, що впливає на плавність тренування [2]. Для вирішення цієї проблеми я буду використовувати кастомний шедулер (див. Додаток А).

Спочатку потрібно встановити оригінальний шедулер всередині venv папки за допомогою sd-scripts команд:

– .\venv\Scripts\Activate.ps1;
– pip install 'git+https://github.com/katsura-jp/pytorch-cosine-annealing-with-warmup'.

Пройти шляхом sd-scripts\venv\Lib\site-packages\cosine_annealing_warmup і змінити назву файлу scheduler.py на scheduler.py.bckp.

Завантажити кастомний шедулер у цю папку з назвою scheduler.py.

T_0 слід обчислювати так: визначити загальну кількість кроків, яке буде зроблено під час тренування, наприклад 1600, і поділити його на бажану кількість рестартів. Це число, в даному випадку 400, потрібно вписувати в T_0, в цей момент буде відбуватися рестарт з підогрівом. За допомогою gamma_min_lr варто регулювати швидкість спаду learning rate, чим нижче число, тим швидше він буде знижуватися (див. рис. 3.5).

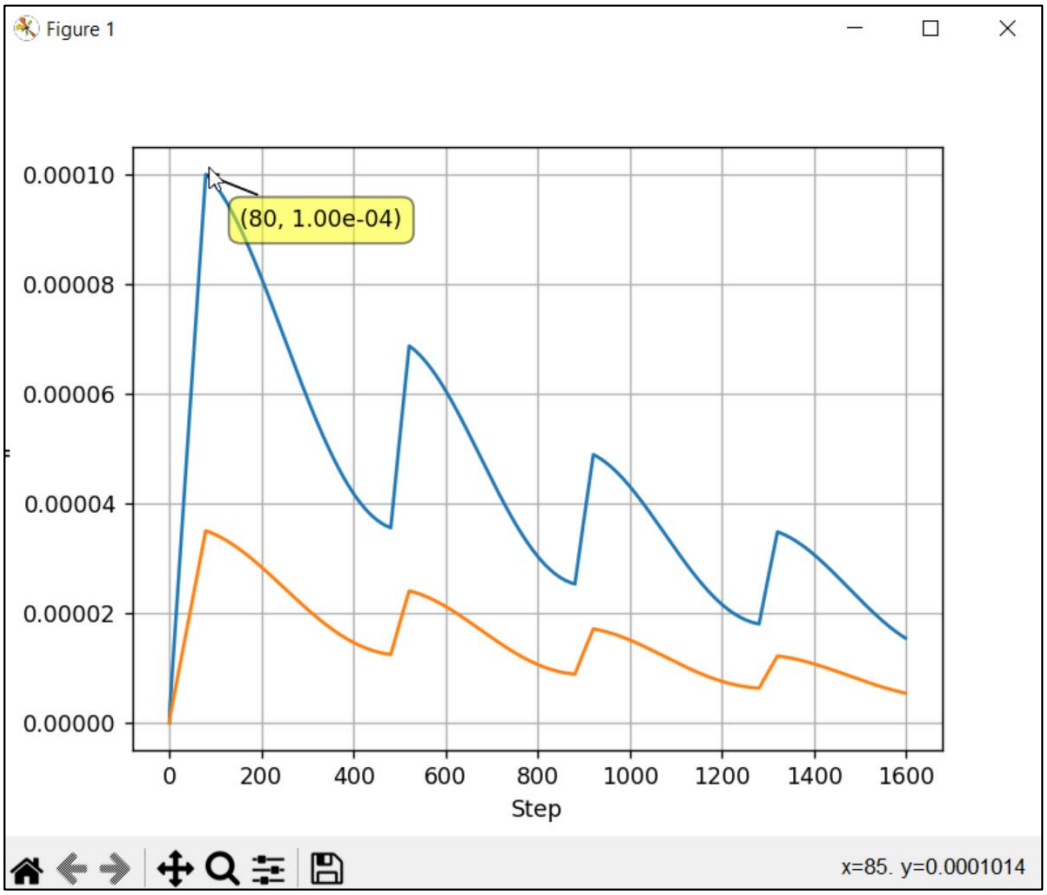


Рисунок 3.5 – Візуалізації роботи кастомного шедулера

3.3.2 Налаштування параметрів

Суть будь-якого тренування LORA полягає в потраплянні в свитспот одразу двох складових частин моделі Stable Diffusion, Unet і Text Encoder (TE) (див. рис. 3.6).

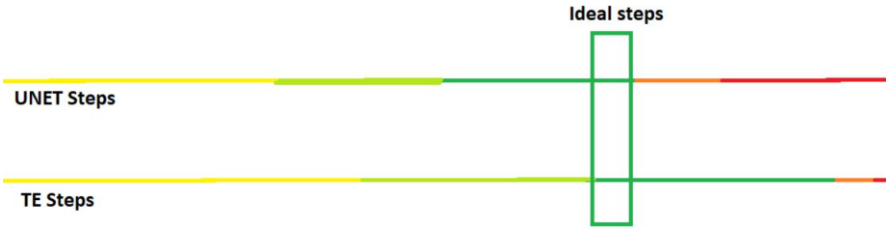


Рисунок 3.6 – Приклад як це має виглядати

Тому, на основі всіх своїх знань, я підібрав ідеальні параметри конкретно для свого випадку тренування стилю (див. рис. 3.7). Після цього можна розпочинати тренування LoRa.

```
[[subsets]]
num_repeats = 6
caption_extension = ".txt"
shuffle_caption = false
flip_aug = false
is_reg = false
image_dir = "C:/Users/user/Desktop/my lora/6_orange"
keep_tokens = 0

[noise_args]

[sample_args]

[general_args.args]
pretrained_model_name_or_path = "C:/Users/user/Desktop/my lora/anything-v3-full.safetensors"
mixed_precision = "fp16"
seed = 42
max_data_loader_n_workers = 1
persistent_data_loader_workers = true
max_token_length = 225
prior_loss_weight = 1.0
clip_skip = 2
xformers = true
cache_latents = true
cache_latents_to_disk = true
max_train_epochs = 15
full_bf16 = false
full_fp16 = true
vae = "E:/stable_df/stable-diffusion-webui/models/VAE/vae-ft-mse-840000-ema-pruned.ckpt"

[general_args.dataset_args]
resolution = [ 768, 768,]
batch_size = 1

[network_args.args]
network_dim = 64
network_alpha = 64.0
min_timestep = 0
max_timestep = 1000

[optimizer_args.args]
optimizer_type = "AdamW8bit"
lr_scheduler = "cosine_with_restarts"
learning_rate = 0.0001
max_grad_norm = 1.0
lr_scheduler_num_cycles = 16
unet_lr = 0.0001
text_encoder_lr = 5e-5
warmup_ratio = 0.05
min_snr_gamma = 5
scale_weight_norms = 1.0

[saving_args.args]
output_dir = "C:/Users/user/Desktop/my lora/orange"
save_precision = "fp16"
save_model_as = "safetensors"
output_name = "orange"
save_every_n_epochs = 1

[bucket_args.dataset_args]
enable_bucket = true
min_bucket_reso = 256
max_bucket_reso = 1536
bucket_reso_steps = 64

[logging_args.args]
logging_dir = "C:/Users/user/Desktop/my lora/logi"
log_prefix = "orange"
log_tracker_name = "orange"
log_with = "tensorboard"

[network_args.args.network_args]
conv_alpha = 32.0
conv_dim = 32
algo = "locon"

[optimizer_args.args.optimizer_args]
weight_decay = "0.1"
betas = "0.9,0.99"
```

Рисунок 3.7 – Приклад конфіг файлу моїх параметрів

3.4 Тестування

3.4.1 Тестування та вибір епох

Після завершення тренування я отримав 15 епох у відповідності з моїми налаштуваннями (див. рис. 3.8).

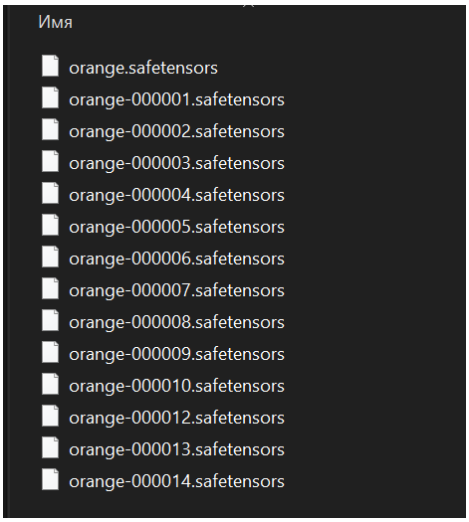


Рисунок 3.8 – Епохи lora

Для тестування кожної епохи я використовуватиму Tensorboard.

TensorBoard – це інструмент для візуалізації даних машинного навчання, створений компанією Google для використання з фреймворком TensorFlow. Він дозволяє користувачам легко візуалізувати різні показники моделі, графи обчислень, гістограми параметрів, зображення та інші дані, які можуть бути корисними для діагностики та оптимізації моделей машинного навчання.

В теорії, за правильно підібраними параметрами, графік Loss в TensorBoard повинен мати постійний спадний тренд протягом усього тренування [8]. Проблема в тому, що в найгіршому випадку на графіках буде видно лише повний провал тренування, а в найкращому, за правильно підібраними параметрами, лише незначне зниження значень.

На знімку 3.9 абсолютно нормальна lora з добре виконаною стилізацією, варто було б взяти 18 епоху, адже там найнижче значення втрат. Але якщо

поглянути на загальну картину, жодна епоха не перевищує першу за значенням, і можна спокійно вибрати будь-яку.

На знімку 3.10 навмисно встановлений x15 lr для демонстрації того, що дійсно покаже TensorBoard, такого потрібно уникати, тренд в ідеалі повинен бути або спадним, або хоча б незмінним. На рисунку 3.11 показано перетренування або недотренування.

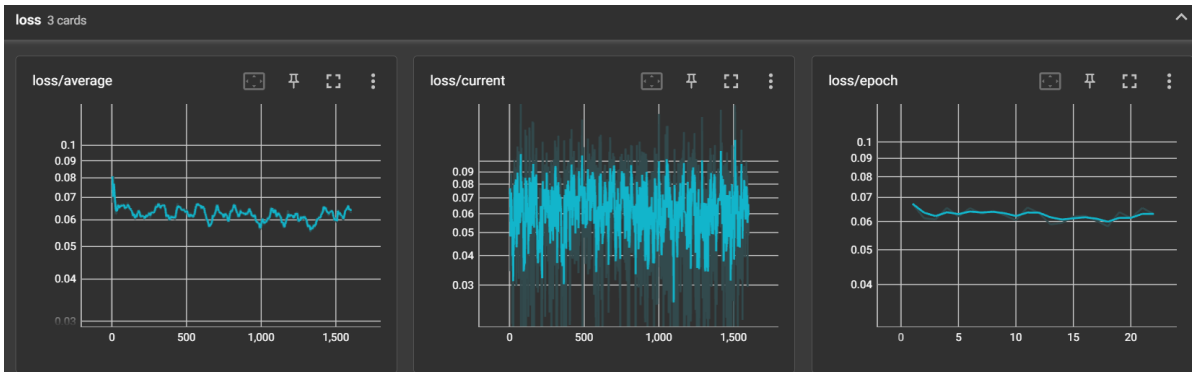


Рисунок 3.9 – Успішне тренування

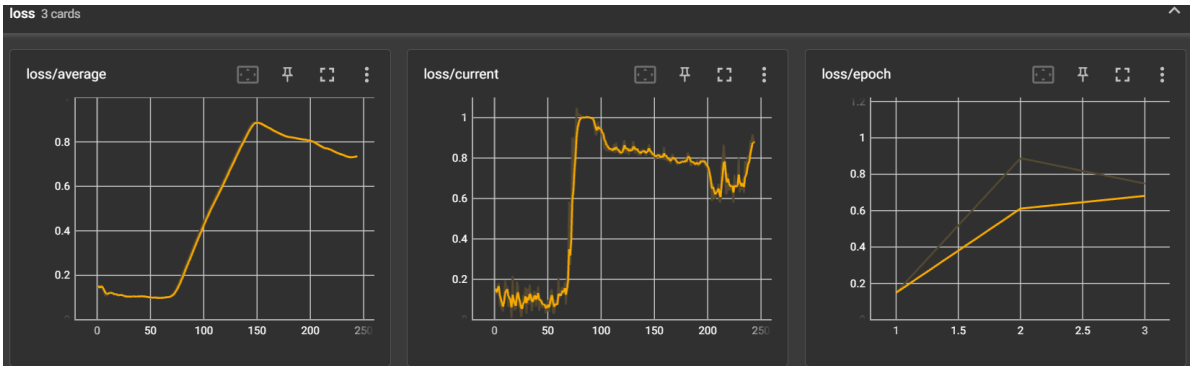


Рисунок 3.10 – Зламалось усе, що тільки могло

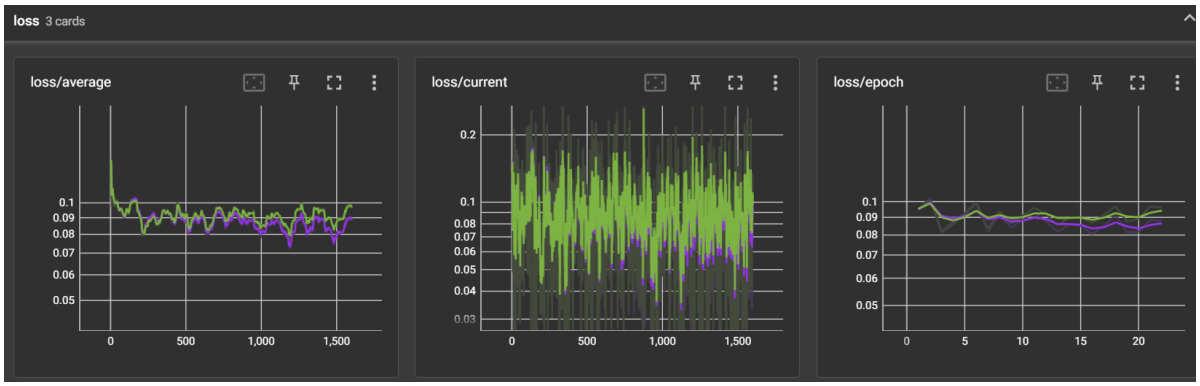


Рисунок 3.11 – Перетренування або недотренування

3.4.2 Тестування на практиці у самому інтерфейсі

Після успішного тестування та вибору цехи можна переходити до тестування у реальних умовах.

Ось які характеристики повинна мати ідеальна лора:

– гнучкість;

– повинна слідувати промту;

– сумісність з іншими lora;

– сумісність з будь-якими сеплерами.

Тому в якості прикладу я склав такий промт де буде речі яких немає в датасеті, так як лора орієнтування тільки на стиль, вона повинна буде відтворити стиль незалежно від того, що я напишу в промт (рис. 3.12).



Рисунок 3.12 – Промт: 1girl, blond hair, blue eyes, sun dress, starry sky, ocean

Як можна помітити, у дівчини біле плаття, сині очі і світле волосся, хоча в датасеті на всіх зображеннях тільки дівчата з чорним волоссям і чорними очима. При цьому на картинці було зображено все те, що я написав у промт, і ідеально відтворено стиль.

Щоб ще раз переконатися в результатах, я спробую щось складніше. Цього разу це буде м’язистий хлопець, нічого подібного в датасеті взагалі немає (рис. 3.13).



Рисунок 3.13 – Промт: 1boy, muscular, orange hair, purple eyes, ank top, clear sky, green field

Тим не менш, моя lora впоралася із завданням і відтворила стиль. Таким чином, можна вважати її успішною і завдання виконаним.

ВИСНОВКИ

Досліджено проблему тренування моделей LoRA (Low-Rank Adaptation) для використання у стабільному дифузійному процесі: розглянуто переваги та недоліки існуючих технологій і систем для її вирішення, сфери застосування таких систем та сучасні підходи до оптимізації нейронних мереж.

Досліджено засоби для вирішення поставленої проблеми: серед розглянутих архітектур нейронних мереж для створення та тренування моделей LoRA обрані найбільш ефективні.

Проведено навчання моделей LoRA для стабільного дифузійного процесу: налаштовано середовище для навчання, підготовлено зображення з відповідних датасетів, створено необхідні для тренування файли та успішно виконано навчання моделей.

Досліджено результати навчання та точність натренованих моделей LoRA: моделі, які тренувалися на датасеті по стилю, показали більшу точність та якість генерації зображень.

Поставлена проблема тренування моделей LoRA для стабільного дифузійного процесу повністю вирішена з допустимою точністю та ефективністю, що дозволяє застосовувати розроблену технологію у різноманітних сферах, потребуючих високоякісних генеративних моделей.

ПЕРЕЛІК ПОСИЛАНЬ

1. Іванов І. П., Петров О. В., Сидоров А. М. Оптимізація параметрів LoRA для моделі стабільного дифузійного процесу. *Молода наука-2023* : зб. наук. праць студентів, аспірантів і молодих вчених. Київ : КНУ, 2023. Т. 4. С. 81–89.

2. Петренко В. О., Сидоренко Н. А. Навчання LoRA з використанням стабільного дифузійного процесу у візуальному моделюванні. *Енергетика, електроніка та інформаційні технології*. Львів : ЛНУ, 2023. С. 201.

3. Brown T., Mann B., Ryder N., Subbiah M. Language Models are Few-Shot Learners. NeurIPS 2020. 2020. URL: https://doi.org/10.48550/arXiv.2005.14165 (дата звернення: 05.05.2024).

4. Attention is All You Need. Advances in Neural Information Processing Systems / A. Vaswani, N. Shazeer, N. Parmar & etc. 2017. URL: https://doi.org/10.48550/arXiv.1706.03762 (дата звернення: 10.05.2024).

5. Kingma D. P., Welling M. Auto-Encoding Variational Bayes. ICLR 2014. URL: https://doi.org/10.48550/arXiv.1312.6114 (дата звернення: 13.05.2024).

6. High-Resolution Image Synthesis with Latent Diffusion Models / R. Rombach, A. Blattmann, D. Lorenz & etc. CVPR 2022. URL: https://doi.org/10.48550/arXiv.2112.10752 (дата звернення: 15.05.2024).

7. Touvron H., Vedaldi A., Douze M., Cord M. Training data-efficient image transformers & distillation through attention. ICML 2021. URL: https://doi.org/10.48550/arXiv.2012.12877 (дата звернення: 20.05.2024).

8. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows / Z. Liu, Y. Lin, Y. Cao & etc. ICCV 2021. URL: https://doi.org/10.48550/arXiv.2103.14030 (дата звернення: 24.05.2024).

9. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. NeurIPS 2020. URL: https://doi.org/10.48550/arXiv.2006.11239 (дата звернення: 26.05.2024).

10. Röder M., Both A., Hinneburg A. Exploring the Space of Topic Coherence Measures. WSDM 2015. URL: https://dl.acm.org/doi/10.1145/2684822.2685324 (дата звернення: 30.05.2024).

ДОДАТОК А

Кастомний шедулер, написаний на Python

```
import math
import warnings
from torch.optim.lr_scheduler import LRScheduler

class CosineAnnealingWarmupRestarts(LRScheduler):

    def __init__(self,
                 optimizer,
                 T_0,
                 T_mult = 1,
                 last_epoch=-1,
                 verbose=False,
                 warmup_steps=1,
                 decay=1,
                 gamma_min_lr : float = 1.,
                 gamma_lr_update_freq: int = 1,
                 down_factor = 1,
                 cycle_warmup = 0,
                 init_lr_ground = False,
                 sin = False
                 ):

        self.count = 0
        self.sin = sin
        self.init_lr_ground = init_lr_ground
        self.cycle_warmup = cycle_warmup
```

```
            self.base_lrs_probe = [0, 0]
            self.counter_warmup = cycle_warmup
            self.gamma_lr_update_freq = gamma_lr_update_freq
            self.gamma_min_lr = gamma_min_lr

            if T_0 <= 0 or not isinstance(T_0, int):
                raise ValueError("Expected positive integer T_0, but got {}".format(T_0))
            self.T_0 = T_0
            self.T_i = T_0
            self.T_mult = T_mult
            self.T_cur = last_epoch
            super(CosineAnnealingWarmupRestarts, self).__init__(optimizer, last_epoch, verbose)

            # Decay attributes
            self.decay = decay
            self.initial_lrs = self.base_lrs

            # Warmup attributes
            self.warmup_steps = warmup_steps
            self.current_steps = 0

            self.down_factor = down_factor

        def get_lr(self):
            if not self._get_lr_called_within_step:
                warnings.warn("To get the last learning rate computed by the scheduler, "
                              "please use `get_last_lr()`.", UserWarning)
```

```
            if self.last_epoch == 1 and self.init_lr_ground:
                return [0,0]

            if not self.sin:

                if self.T_cur < self.cycle_warmup and self.count > 0:
                    # print('get g_warmup')
                    return [((base_lr * self.down_factor)) * self.T_cur / self.cycle_warmup + (base_lr * self.down_factor) for base_lr in self.base_lrs]
                else:
                    # print(self.T_cur,self.warmup_steps,self.T_i,self.current_steps,self.last_epoch)
                    return [
                        (self.current_steps / self.warmup_steps) *
                        ((base_lr * self.down_factor) + (base_lr - (base_lr * self.down_factor)) * (1 + math.cos(math.pi * (self.T_cur - self.cycle_warmup + self.counter_warmup) / (self.T_i - self.cycle_warmup + self.counter_warmup))) / 2)
                        for base_lr in self.base_lrs
                    ]
            else:
                # устарело
                return [
                    (self.current_steps / self.warmup_steps) *
                    ((base_lr * self.down_factor) + (base_lr - (base_lr * self.down_factor)) * (1 + math.cos(math.pi * (self.T_cur - self.T_0 / 2 ) / (self.T_i - self.T_0 / 2))) / 2)
                    for base_lr in self.base_lrs
                ]

        def step(self, epoch=None):
```

```
        """Step could be called after every batch update"""
        if epoch is None and self.last_epoch < 0:
            epoch = 0

        if self.count > 0:
            self.counter_warmup = 0

        if self.T_cur + 1 == self.T_i:
            if self.verbose:
                print("multiplying base_lrs by {:.4f}".format(self.decay))
            self.base_lrs = [base_lr * self.decay for base_lr in self.base_lrs]
            self.count = self.count + 1
            # print(self.count)

        if epoch is None:
            epoch = self.last_epoch + 1
            if self.last_epoch > self.warmup_steps:
                self.T_cur = self.T_cur + 1

            if self.T_i % self.gamma_lr_update_freq == 0 and self.last_epoch > self.warmup_steps:
                for i in range(len(self.base_lrs)):
                    self.base_lrs[i] *= self.gamma_min_lr

            if self.current_steps < self.warmup_steps:
                self.current_steps += 1

            if self.T_cur >= self.T_i:
                self.T_cur = self.T_cur - self.T_i
```

```
                self.T_i = int((self.T_i - self.cycle_warmup) * self.T_mult) +
self.cycle_warmup
                self.cycle_warmup = math.floor(self.cycle_warmup * self.T_mult)

            self.last_epoch = math.floor(epoch)

            class _enable_get_lr_call:

                def __init__(self, o):
                    self.o = o

                def __enter__(self):
                    self.o._get_lr_called_within_step = True
                    return self

                def __exit__(self, type, value, traceback):
                    self.o._get_lr_called_within_step = False
                    return self

            with _enable_get_lr_call(self):
                for i, data in enumerate(zip(self.optimizer.param_groups,
self.get_lr())):
                    param_group, lr = data
                    param_group['lr'] = lr
                    self.print_lr(self.verbose, i, lr, epoch)

            self._last_lr = [group['lr'] for group in self.optimizer.param_groups]
```