Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Вдосконалення агентів LLM шляхом їх використання в якості пошукових агентів
(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-24-1
Олексій СТРЕЛЬЦОВ
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник доц. Олександра ВІТЬКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Стрельцову Олексію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Вдосконалення агентів LLM шляхом їх використання в якості пошукових агентів

затверджена наказом університету від 6 квітня 20 26 р. № 269Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 19 травня 20 26 р.

3. Вихідні дані до роботи Документація мови Python 3, документація до бібліотек PyTorch, transformers, NNsight, LlamaIndex; науково-технічні публікації на теми: великі мовні моделі, інженерія контексту, масштабування під час виведення, пошукові алгоритми; набори даних для тестування та донавчання великих мовних моделей.

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі та постановка задачі

2) Практична реалізація

3) Проведення експериментів та їх аналіз

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	06.04.2026	виконано
2	Аналіз предметної галузі	09.04.2026	виконано
3	Постановка задачі	10.04.2026	виконано
4	Аналіз сучасних методів масштабування під час	17.04.2026	виконано
	виведення та їх адаптація		виконано
5	Формування сценаріїв експериментів	18.04.2026	виконано
6	Вибір та підготовка навчальних і тестових	19.04.2026	виконано
	вибірок		
7	Проведення експериментів	26.04.2026	виконано
8	Аналіз результатів	27.04.2026	виконано
9	Оформлення пояснювальної записки	01.05.2026	виконано
10	Перевірка на академічний плагіат	03.05.2026	виконано
11	Нормоконтроль	04.05.2026	виконано
12	Підготовка презентації та доповіді	06.05.2026	виконано
13	Попередній захист	07.05.2026	виконано
14	Рецензування	08.05.2026	виконано
15	Захист перед ЕК	19.05.2026	

Дата видачі завдання 6 квітня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Олександра ВІТЬКО
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 130 с., 25 рис., 4 табл., 3 дод., 30 джерел.

АГЕНТНІ СИСТЕМИ, ВЕЛИКІ МОВНІ МОДЕЛІ, ЗАДАЧІ ЗАДОВОЛЕННЯ ОБМЕЖЕНЬ, ІНЖЕНЕРІЯ КОНТЕКСТУ, МАСШТАБУВАННЯ ПІД ЧАС ВИВЕДЕННЯ, ПОШУКОВІ АГЕНТИ.

Об’єкт дослідження – великі мовні моделі.

Предмет дослідження – покращення ефективності мовних моделей через механізми масштабування під час виведення – залучення додаткових обчислень для покращення роботи моделі, на основі пошуку.

Мета роботи – дослідження відмінностей між агентами на основі мовних моделей та класичними пошуковими агентами, причин цих відмінностей та можливості подолання проблем пов'язаних з їх наявністю.

Методи дослідження – теоретичний, емпіричний.

В результаті проведених досліджень проаналізовано, які відмінності заважають великим мовним моделям відтворювати окремі властивості пошукових агентів, запропоновано стратегію подолання відповідних перешкод і методи, які нею керуються.

В якості програмної платформи використано бібліотеки машинного навчання «PyTorch», «transformers», фреймворк для управління агентами «LlamaIndex», бібліотеку для аналізу мовних моделей «NNsight», мову програмування Python 3.

Реалізовані методи є ефективним інструментом масштабування під час виведення для мовних моделей і у відповідних їм сценаріях випереджають сучасні аналоги.

ABSTRACT

Master’s thesis contains: 130 pp., 25 fig., 4 tabl., 3 ann., 30 references.

AGENTIC SYSTEMS, CONSTRAINT SATISFACTION PROBLEMS, CONTEXT ENGINEERING, LARGE LANGUAGE MODELS, SEARCH AGENTS, TEST-TIME SCALING.

Research object – large language models.

Research subject – increasing large language models performance with search-inspired approach to test-time scaling – process when model’s performance is enhanced by introducing additional computation.

Research goal – to look into the differences between language model based agents and search based agents, reasons behind those differences and the ways to bridge the two types of agents.

Research methods – theoretical, empirical.

During this research it was analyzed which differences stop language models from reproducing some capabilities of search-based agents. A strategy to overcome these limitations was proposed along with test-time scaling methods derived from it.

As the programming platform machine learning libraries «PyTorch», «transformers», agentic workflows orchestration framework «LlamaIndex», library for language model interpretability «NNsight» and programming language Python 3 were used.

Methods of test-time scaling that were designed are very effective in test-time scaling of language models and in corresponding scenarios overcome their state-of-the-art analogs.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі та постановка задачі 11
1.1 Задача масштабування під час виведення при використанні мовних моделей 11
1.2 Масштабування під час виведення на основі пошуку. Виведення за допомогою мовних моделей як процес пошуку 16
1.3 Недоліки мовної моделі як пошукового алгоритму 18
1.3.1 Порівняння процесу виведення мовної моделі з методами пошуку для задач задоволення обмежень 20
1.3.2 Порівняння процесу виведення мовної моделі з методом пошуку A* 22
1.4 Постановка задачі 25
2 Аналіз сучасних методів масштабування під час виведення та їх адаптація 26
2.1 Методи масштабування для моделей типу «чорна скриня» 26
2.1.1 Сучасні методи масштабування для моделей типу «чорна скриня» 26
2.1.2 MCTS-Rewrite 35
2.1.3 LLM-CSP 36
2.2 Методи масштабування для моделей типу «біла скриня» 37
2.2.1 Сучасні методи масштабування для моделей типу «біла скриня» 38
2.2.2 Entropy bruteforce 41
2.2.3 Entropy search 42
2.2.4 Entropy CSP 43
2.2.5 Додаткові переваги запропонованих методів 44
2.3 Особливості запропонованих методів 46

2.3.1 Методика застосування в складніших агентних задачах..... 47
3 Проведення експериментів та їх аналіз 49
3.1 Методика тестування 49
3.2 Тестування запропонованих методів для моделей типу «чорна скриня» 50
3.3 Тестування запропонованих методів для моделей типу «біла скриня» 56
3.3.1 Визначення гіперпараметрів для методів на основі ентропії.... 56
3.3.2 Результати експериментів 60
Висновки 63
Перелік джерел посилання 65
Додаток А Вихідний код для методів на основі інженерії контексту 69
Додаток Б Вихідний код для методів на основі семплювання..... 121
Додаток В Відомість кваліфікаційної роботи 130

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект;

CSP – Constraint Satisfaction Problem – задача задоволення обмежень;

LLM – Large Language Model – велика мовна модель;

MCTS – Monte Carlo Tree Search – дерево пошуку Монте Карло;

PRM – Process Reward Model – модель винагороди процесу;

PUCT – Predictor + Upper Confidence Threshold – верхня межа впевненості з механізмом передбачення;

UCT – Upper Confidence Threshold – верхня межа впевненості.

ВСТУП

Великі мовні моделі є потужним інструментом, що вже застосовується для автоматизації у багатьох сферах [1]. Проте, особливості задач на основі природної мови включають високу неоднозначність і, як результат, необхідність мати можливість виправляти помилки, пропонувати альтернативні рішення або застосовувати планування для тривалих задач. Такі властивості слабко проявляються у сучасних великих мовних моделях, особливо у тих, що широко використовуються в локальних сценаріях: довгі діалоги призводять до падіння точності [2], а найбільш стабільні методи масштабування моделей переважно користуються генерацією кількох варіантів відповіді для однакових вхідних даних. Це погіршує можливість до агентного застосування інструментів на основі великих мовних моделей, а такі «агенти» критикуються за те, що за окремими характеристиками гірші за класичних пошукових агентів.

Певні відповідні вади мовних моделей вважаються фундаментальними і зазвичай розглядаються лише в рамках досліджень зі змін у підході до дизайну та навчання великих мовних моделей або їх альтернатив. Це сприяє високому рівню інновацій у сфері великих мовних моделей і дозволяє створювати моделі, які здатні виконувати складніші задачі без сторонньої допомоги. Проте, такий підхід фактично залишає поза увагою вже існуючі знання та інфраструктуру для застосування великих мовних моделей, які можуть масштабувати існуючі моделі до виконання складніших задач, що зробить цей процес ефективнішим і доступнішим.

Відповідні дослідження з ефективних методів масштабування великих мовних моделей більше покладаються на відтворення класичних підходів до логічного виведення, аніж на забезпеченні функціонального паритету, який зазвичай вважають досяжним з часом. У цій роботі пропонується альтернативний погляд на проблему. Якщо бажаними для мовних моделей є властивості пошукових агентів, то варто порівняти

обидва процеси і спробувати реалізувати механізми, що долають відмінності між ними. В ході даного дослідження проводиться аналогія між процесом виведення виключно декодерної мовної моделі та ключовими алгоритмами пошуку для встановлення конкретних перешкод в досягненні еквівалентних можливостей, пропонуються методи подолання цих перешкод і порівнюються з існуючими методами масштабування під час виведення. В ході роботи розглядаються підходи як для моделей типу «чорна скриня», так і для моделей типу «біла скриня».

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Задача масштабування під час виведення при використанні мовних моделей

Мовна модель [3] – модель машинного навчання, що за наданим початком тексту надає його продовження, вирішуючи задачу відому, як next token prediction. Особливого успіху в вирішенні таких задач досягли великі мовні моделі на основі архітектури Transformer [4]. Представлений у архітектурі блок Scaled Dot-Product Attention (SDPA, механізм уваги на основі скалярного добутку та масштабування) має дуже високу ефективність у задачах, що покладаються на асоціативну пам’ять, але для того аби запам’ятати достатню для якісної роботи кількість текстів має мати достатньо прикладів та параметрів. Подібні моделі добре масштабуються, завдяки чому збільшення моделі та набору тренувальних даних відповідає кращій ефективності [5]. Відповідно до цього принципу покращення ефективності через збільшення, моделі на основі архітектури Transformer активно збільшували, що виділяло їх на фоні старих архітектур і закріпилося у назві «великі мовні моделі».

Точна мовна модель, натренована на різноманітному наборі даних, часто може мати таку емерджентну властивість як «вісь асистента» [6]. Фактично, це вивчені правила побудови діалогів, використовуючи які модель може вести діалоги, отримуючи дані у відповідному форматі. Для підтримки такого формату та заохочення до використання саме «вісі асистента» мовні моделі додатково тренуються. Отримані асистенти мають дуже широку сферу застосування, дозволяючи автоматизувати процеси, пов’язані з використанням природної мови чи формальних мов або які мають відповідний інтерфейс.

Проте, такий підхід має істотні вади, що впливають на точність відповідей та ефективність роботи незалежно від якості великої мовної моделі.

Неоднозначність за замовчуванням – більшість задач на основі природної мови не наводять явно весь релевантний контекст. Через це модель звикає працювати в умовах невизначеності та робити припущення замість уточнення в ході взаємодії. Причому така впевненість може проявлятись і при наведенні фактів (так звані галюцинації). Часто її можна упередити, але дуже складно позбавитись, навіть коли модель в текстових відповідях погоджується із помилкою. Така поведінка зазвичай пов’язана із конфліктом цілей моделі, що має генерувати текст, який відповідатиме прикладам з тренувальним даних, прикладам, що навів користувач і відгукам середовища [7].

Авторегресія – модель зобов’язана обирати наступний токен, тобто має фіксований «бюджет» на міркування, але має вирішувати задачі різної складності. Наприклад, в задачах планування це може призвести до відтворення стандартної структури плану замість власне планування. Також це означає, що при виборі токена помилка поширюється, бо модель робить передбачення наступного токена на основі попередніх, в тому числі тих, що були передбачені моделлю.

Використання лише «гарних даних» – очевидно, щоб навчити модель надавати правильні відповіді, їй треба надавати приклади успішних діалогів. Проте в результаті для моделі будь-які помилки стають даними поза вибіркою, що стрімко знижує ефективність моделі. Певною мірою це можливо врегулювати через навчання з підкріпленням, але відповідно до досліджень з безпеки штучного інтелекту властивості базової моделі зазвичай продовжують проявлятись.

Цілком можливо, що такі проблеми можна вирішити через внесення змін до підходів до побудови та навчання мовних моделей, але це не вирішує проблему для вже існуючих моделей.

Методи, що намагаються вирішити ці проблеми для вже натренованих моделей називають методами масштабування під час виведення (test-time scaling або inference-time scaling). Ці методи покладаються на залучення додаткових обчислень для підвищення ефективності моделей і, зокрема, вирішення наведених вище проблем.

Відповідно до переходу до використання мовних моделей в агентних задачах саме методи масштабування вважають наступною сходинкою в розвитку передових моделей ШІ після «пройдених» етапів попереднього тренування (створення базової мовної моделі, задача вирішена архітектурою Transformer) і дотренування (перетворення загальної моделі на модель-асистента, розкриття окремих персон або експертиз мовної моделі). Також вважається, що наступним етапом після масштабування є колаборація – здатність агентів з різною експертизою до комунікації в середовищі. Логіка такого твердження наступна – незалежно від свого розміру та якості модель не може бути оракулом, що вгадує контекст або на всякий випадок розглядати усі можливі сценарії. І навіть така модель потребуватиме здатності до навчання для ознайомлення із новою інформацією. Для моделі цінною є її гнучкість, що дозволить покращувати чи змінювати результат за потреби. Відповідно, бажаною є модель чи система на основі моделі, що здатна ефективно користуватись новою інформацією. Серед таких методів можна виділити наступні категорії.

На основі семплювання – такі методи застосовують до виведення стратегії альтернативні жадібному декодуванню. Поширеним є виведення на основі ймовірності – найбільш ймовірний токен не обирається за замовчуванням, токени обираються відповідно до наданої моделлю ймовірності, пропорційно. Завдяки цьому виклик моделі кілька разів надає вибірку з кількох різних результатів. Цей метод часто називають семплюванням за температурою, адже в більшості випадків користувачу надається право обирати значення параметри температури операції softmax, що контролює варіацію розподілу. Це дозволяє отримувати вибірку

відповідей з певними властивостями або таку, що краще демонструє простір рішень. Причому такі методи можуть використовуватись для впливу як на конкретну генерацію, так і для створення більш репрезентативних вибірок з різних альтернатив. Наприклад, метод семплювання на основі beam search (пошук «лучем») [8] полягає в створенні спекулятивних траєкторій на основі кількох кращих відповідей і вибору найкращої за ланцюговим правилом. Або при застосуванні семплювання за температурою можна аналізувати вибірку з кількох незалежних генерацій для вибору найкращої (за можливості це підтвердити, best-of-n (найкращий-з-n) [9]) або «медіанної» (self-consistency (самоузгодженість) [10]).

Інженерія контексту – ця група методів полягає у створенні стратегії взаємодії з моделлю таким чином, що забезпечить її релевантним контекстом, розбиватиме задачі на підзадачі та управлятиме видимим діалогом. Очевидно, що відповідні взаємодії потребують розуміння природної мови і покладаються на використання моделі. Тим більше, що попередниками методів інженерії контексту були методи інженерії інструкцій – практичні спостереження користувачів асистентів щодо ефективної роботи з ними. Серед таких методів були «few-shot prompting» (інструкція з кількома прикладами, де в інструкції наводиться приклад бажаного результату і форматування відповіді), «chain of thought» (ланцюг думок) де користувач надає завдання поступово. Тобто методи інженерії контексту – автоматизована інженерія інструкцій. Перша хвиля відповідних методів ставила задачу лише автоматизувати роботу, що мав виконувати користувач і не зовсім відповідала задачі масштабування. Проте, після досягнення задовільного рівня успішності все більше методів користуються відсутністю вузького місця у вигляді користувача і розглядаються саме з точки зору покращення результатів.

На основі «міркування» – ці методи полягають в додатковому тренуванні моделей-асистентів за допомогою навчання з підкріпленням на справжніх діалогах (у тому числі з помилками), в ході якого моделі

заохочуються до використання рефлексії. Як було зазначено раніше, авторегресивне виведення змушує обирати певний токен одразу. Завдяки рефлексії модель може використати завершену «думку» в якості додаткового бюджету для міркування. Такий бюджет може також бути витрачено на аналіз вгаданого контексту. Відповідний процес може ініціюватись вставленням спеціального токена або виконуватись моделлю самостійно (так звані «aha moments» [11]) через регулярне застосування відповідних рефлексивних конструкцій. Перший метод не дуже гнучкий і передусім застосовується в комерційних моделях з метою унеможливити витік даних про реалізацію механізму (відповідні послідовності генеруються на певному етапі «планування відповіді», яку користувач не отримує), а другий є менш контрольованим. З точки зору метрик такі послідовності міркування роблять «м’який перезапуск» моделі, підвищуючи ентропію її логітів (тобто прибирають упередження моделі).

На практиці такі методи часто застосовуються в різних комбінаціях для отримання найкращого результату. Тим більше, що поодинці такі методи мають доволі істотні вади.

Методи на основі семплювання мають низьку ефективність з точки зору дослідження альтернатив та стабільності. В основі цього – невибірковість семплювання, через яку зависокі значення температури можуть призвести до виведення моделлю незрозумілих послідовностей коли застосовані до синтаксичних конструкцій. За помірної температури семплювання зазвичай використовується як спосіб отримати справжній розподіл ймовірності текстів моделі і використовується як метод Монте Карло. До того ж застосування таких методів потребує можливості впливати на роботу моделі або підтримки такого методу програмою, що керує виведенням моделі.

Методи на основі інженерії контексту мають доволі складну задачу, адже мають застосувати дані про обрану траєкторію та відгук середовища на неї, при тому, що збереження цього в самому діалозі часто не виправляє

ситуації. Причому вузьким місцем також є модель-виконавець, адже виправлення помилок зводиться до фактичної здатності моделі зрозуміти свою помилку.

Методи на основі міркування переважно позбуваються помилок, пов’язаних із потребою у складніших послідовностях виведення чи паралельному виведенні [12] (наприклад, вирішенні математичних задач), але для цього значно збільшують кількість використовуваних токенів, від яких квадратично залежить швидкість роботи моделі та які впливають на роботу механізму уваги [4]. До того ж контроль відносно кількості застосувань рефлексії часто належить лише моделі, для якої коректне виведення є скінченним, а значить не може контрольовано масштабуватись.

1.2 Масштабування під час виведення на основі пошуку. Виведення за допомогою мовних моделей як процес пошуку

Якщо звертатись до класичних методів у штучному інтелекті, гарним прикладом масштабування під час виведення є процес пошуку і відповідні йому пошукові агенти. За класифікацією AIMA [13] серед агентів можна виділити агентів, які в своїх діях керуються моделлю середовища (моделлю світу). Відповідні агенти мають певне представлення про стан, можливі дії і, за допомогою моделі середовища, здатні обчислювати функцію переміщення між станами для прогнозування результатів своїх дій. Простіші агенти працюють із майбутніми станами жадібно і діють відповідно до певного списку правил. Проте, при наданні агенту функцій цілі або вигоди агент отримує необхідну основу для виконання пошуку в просторі станів: він знає свій початковий стан, які дії він може виконати, як це вплине на стан, як оцінити стан і визначити термінальний стан. Відповідно до практичних обмежень агента чи середовища обирається алгоритм пошуку, що відповідає бажаній кількості викликів моделі середовища, часових або обчислювальних обмежень тощо. Такі обмеження

можуть включати і здатність до роботи в умовах невизначеності, коли замість чітких станів і реакцій середовища альтернативи формують розподіл. Саме на основі таких алгоритмів працюють сучасні ШІ-моделі для гри в шахи або го.

Особливо цікавою для наведеної теми є наявність серед алгоритмів пошуку таких, що вирішують проблеми аналогічні зазначеним вище. Зокрема, факт того, що класичні системи не мають таких самих вад, як мовні моделі, часто є причиною критики останніх.

Бектрекінг – через авторегресивну архітектуру модель не може зробити «крок назад» і тому фактично не здатна до бектрекінгу. Схожі на цей механізми присутні в мовних моделях, але мають скоріше рефлексивну природу, причому модель зазвичай фокусуватиметься лише на останній виконаній операції. Саме «повернення» носить спекулятивний характер, адже очевидно, що відтворення попереднього стану означатиме для моделі утворення циклу. До речі, ця проблема проявляється не лише як нездатність виправити помилку, але й у випадках поступового уточнення задачі [2].

Здатність до планування – через тренування на прикладах, де модель з першої спроби надає коректну відповідь мовні моделі мають погані здібності в плануванні й складні сценарії, що потребують спочатку отримати відгук від користувача/системи зазвичай можливі тільки через спеціальні агентні інтерфейси. Методи пошуку відрізняє те, що в них зазвичай застосовується очікувана оптимальність результату, а не безпосередньої дії. Не можна сказати, що мовна модель керується лише цим, але для неї горизонтом часто є одна генерація, а не вся розмова.

Вичерпність – хоч пошук через «грубу силу» зазвичай є нереалістичним для багатьох задач, велика кількість алгоритмів має явні можливості з ефективного перебору рішень (в тому числі з точки зору виконуваних обчислень та використання пам’яті) завдяки евристичним функціям, обмеженням доменів тощо. Як мінімум здатність до перебору дозволяє гарантувати певну кількість альтернативних рішень. Така гарантія

є достатньо цікавою як, наприклад, противага фіксації моделі на певних вгаданих умовах.

Відповідні вади особливо впливають на використання мовних моделей в якості основної компоненти різних програмних агентів, а їх усунення фактично формує основні підходи до дизайну таких систем:

– через відсутність здатності до планування планом опікується власне зовнішня оболонка, часто план є статичним, або покладається на наведення користувачем чи системою достатньо довгого і точного опису задачі аби використати токени з нього;

– спеціальні формати для збереження «траєкторії» та її компактного представлення. Такий механізм забезпечує не лише економію обчислювальних ресурсів, але і здатність якісно підготувати пам’ять агента. Це також часто включає уникнення довгих траєкторій та керує обробкою відгуків;

– переважання «горизонтального» масштабування, тобто паралельних генерацій із застосуванням семплювання замість ітеративного покращення однієї.

1.3 Недоліки мовної моделі як пошукового алгоритму

В цій роботі пропонується виведення ефективних підходів до масштабування під час виведення великих моделей через порівняння процесу виведення великої мовної моделі та виконання пошуку пошуковим агентом.

По-перше, варто визначити яким, є простір пошуку для мовної моделі. В такому просторі стан – це текст. Дія яку можна виконати, для переміщення між станами – це додати токен. Ціль пошуку зберігається – досягти стану, в якому модель вважає текст завершеним. Повне визначення процесу пошуку описується за формулою 1.1:

$$A = V,\ S = A^*,\ T(s,a) = T(s||a),\ P(a|s) = M(s), \qquad (1.1)$$

де V – множина токенів, алфавіт;

M – модель;

A – множина дій;

S – простір станів;

P – стратегія;

T – функція переходу.

Альтернативно, стан (і відповідно дію) можна визначити семантично. Часто додавання одного токена не змінює інформаційне наповнення тексту (синтаксичні конструкції тощо). В такому разі станами можна вважати семантично різні тексти, а переміщеннями – відповідні послідовності токенів. Таке визначення також передбачає, що однаковими можуть бути стани, зовсім різні за строковими метриками. Проте, таке визначення потребує визначення способу семантичного порівняння F. Такою функцією, може бути в тому числі, і порівняння на основі внутрішнього стану моделі який в певній мірі описує семантичне значення тексту. Тоді отримуємо визначення дій відповідно стану за формулою 1.2.

$$A(s) = \{a \in V^+ | F(s, s||a) = True \cup \forall a'\ F(s, s||a') = False\}, \qquad (1.2)$$

де a' – префікс a.

Також визначити стан можна за відгуком – в рамках конкретної задачі стани можна відрізняти за результатом валідації запропонованого рішення. Проте, у такого підходу є велика кількість недоліків. По-перше, він потребує дуже детальних відгуків (Правильно/неправильно – не дуже інформативно). По-друге, усі проміжні стани, що не відповідають формату завдання, автоматично не враховуються, хоча серед станів таких більшість.

На основі цього визначення можемо порівнювати виведення за допомогою мовної моделі із конкретними алгоритмами пошуку.

1.3.1 Порівняння процесу виведення мовної моделі з методами пошуку для задач задоволення обмежень

Окремі задачі, наприклад, алгоритмічні часто можна представити в якості набору обмежень, визначених семантично. Найпростішою аналогією в такому випадку будуть алгоритми пошуку для задач задоволення обмежень (CSP). Такі задачі не зовсім відповідають агентному виконанню адже шукають не траєкторію з дій, що ведуть до цільового стану, а множину коректних станів за обмеженнями. Проте, можна справедливо зауважити, що результатом виведення за допомогою моделі також буде саме стан, який, до речі, також відображає послідовність дій у середовищі. Іншою цікавою особливістю таких методів є зосередженість на ефективному переборі можливих варіантів і, як результат, просунуті підходи до технік бектрекінгу та використання траєкторій для оптимізації простору пошуку.

Узагальнену процедуру алгоритмів для задач задоволення обмежень можна описати відповідно до лістингу 1.1 [14].

Лістинг 1.1 – Загальний алгоритм пошуку для бінарних задач задоволення обмежень

```
PROCEDURE bcssp (n,status)
     BEGIN
          consistent ← true;
          status ← “unknown”;
          i ← 1;
          WHILE status = “unknown”
          DO BEGIN
               IF consistent
               THEN i ← label(i, consistent)
               ELSE i ← unlabel(i, consistent);
               IF i>n
               THEN status ← “solution”
               ELSE IF i = 0
```

Продовження лістингу 1.1

```
                    THEN status ← "impossible"
                END
            END;
```

Згідно з алгоритмом виконується перебір змінних, що передаються за індексом i до функцій label та unlabel відповідно до узгодженості поточного стану. Функція label повертає оновлений стан, що включає передану змінну, а функція unlabel виключає її зі стану. Різні алгоритми мають однакову основу, наведену у лістингу, але різні сторонні ефекти від застосування функцій label та unlabel. Наприклад, наївна реалізація label після включення змінної до стану змінить індекс на наступний, а unlabel на попередній. Більш витончені методи, що виконують перебір швидше, відповідно, можуть змінювати індекс більш агресивно, перестрибуючи завідомо погані значення. Для оновлення даних про узгодженість поточного стану в будь-якій реалізації також присутня функція check, яку через це цілком справедливо вважати третьою фундаментальною операцією. Якщо порівняти це з процесом виведення мовної моделі отримуємо наступні відмінності.

Операція check (в даному випадку прихована за реалізацією label/unlabel), присутня у таких методах, виконується моделлю і цілком спирається на її представлення про коректність, а справжня перевірка відбувається лише по завершенні генерації. Це означає, що модель пропонує відповідь, отриману спекулятивно.

Операція label також виконується моделлю, але це і є бажаний результат, адже клас задач, які вирішують мовні моделі-асистенти доволі великий, через що кількість змінних, які беруть участь в процесі прийняття рішень складно оцінити. Це погано з точки зору інтерпретації роботи моделі і робить атрибуцію помилок спекулятивної траєкторії ще складнішою.

Зазвичай результатом цього є практична неможливість охопити усі релевантні дані у контексті запиту. Як наслідок, модель може не тільки

неправильно обрати змінну, але й некоректно оцінити її область допустимих значень, адже керуватиметься власними представленнями. Відповідно до перерахованих відомих вад мовних моделей слабким місцем також є крок unlabel, що фактично відповідає виконанню бектрекінгу, який, як відомо, є слабким місцем мовних моделей. Окрім самої операції бектрекінгу, увесь процес контролю за доменами змінних та їх звуженням відбувається за рахунок рефлексивних механізмів моделі, тобто є не дуже надійним і робить цей процес менш контрольованим.

В підсумку можна дати наступну оцінку: мовні моделі не мають відомих ефективних засобів для контролю прийнятих рішень чи виведених змінних, через що до них складно застосовувати аналогії, пов’язані із алгоритмами, такими як пошук для задач задоволення обмежень. Окремо цей процес ускладнює той факт, що отримані траєкторії є спекулятивними і не гарантують, що усі попередні операції були коректними. Тобто, найбільш надійним механізмом атрибуції та встановлення конкретних причинно-наслідкових зв’язків все ще є модель, хоча це не її сильна сторона.

Проблеми з бектрекінгом демонструють, що можливості до застосування навчання в контексті та рефлексії достатньо обмежені. Найслабшим місцем є операція unlabel, яку виконати можна лише радикально – прибравши історію чату з контекстного вікна. В такому разі потрібно забезпечити збереження даних про виконану траєкторію, що ускладнюється тим, що вона спекулятивна. Проте, успішний аналіз дозволить розпочати нову генерацію із оновленою, більш точною інструкцією для моделі.

1.3.2 Порівняння процесу виведення мовної моделі з методом пошуку A*

Пошук A* є більш універсальним методом пошуку, що доволі широко використовується і для агентних задач. Концепція такого пошуку полягає в

використанні евристичної функції, що оцінює вартість шляху до цільового стану [15]. За допомогою такої евристики алгоритм формує чергу з пріоритетом доступних дій за сумарною вартістю пройденого та майбутнього шляху, що також забезпечує «природній» бектрекінг у разі якщо у кінці пріоритетного шляху є перешкода, а евристична оцінка – некоректна. Ефективною відповідна евристика є, якщо не переоцінює кількість необхідних дій, адже в іншому разі (якщо шлях може виявитись коротшим, ніж його оцінила функція) пріоритет стає некоректним. Такий підхід є достатньо зручним при роботі з невизначеністю, адже не потребує агрегації оцінок усього розподілу можливих наслідків, а лише поточного стану. Основною перешкодою часто є можливість формулювання відповідної евристики. Проте, саме штучний інтелект через просунуті методи нелінійної оптимізації часто допомагає із створенням апроксимацій.

Фактичного аналогу евристичної функції немає, адже ціллю пошуку є логічне, а не швидке (навіть зважаючи на те, що моделі додатково тренують для лаконічного виведення) завершення поточного тексту. В той же час можна вважати, що окремі елементи такого впливу присутні, бо на вибір моделлю токена впливає, зокрема, той факт, що текст має завершитись взагалі, а не прямо зараз. Відповідно, в вагах моделі є певне усвідомлення довготривалого впливу вибору (але не повноцінного планування) і «вартості» такої помилки.

Одразу можна помітити, що виникає невідповідність щодо натурального бектрекінгу – на практиці моделі без відповідного тренування з використанням навчання з підкріпленням віддають пріоритет ідеї, що наданий текст – правильний, а замість виправлення краще підлаштуватись.

Причому застосовуючи підхід пошуку A* до генерації у реальних сценаріях (генерація відкритого типу) отримуємо іншу невідповідність – у випадку коли завершення виведення фактично залежить від зовнішнього відгуку будь-яка оцінка може бути занизькою, а значить навіть за потреби алгоритму не потрібно буде виконувати бектрекінг.

Така оцінка цілком справедлива, зважаючи на визначене поняття стану, переходу і цілі пошуку, за яким попередньо згенерований текст залишається частиною траєкторії, а отримання відгуку від середовища – «нав’язаним переходом». Такий перехід не гарантує стану, еквівалентного поверненню до наступного найкращого варіанту в черзі, й фактично демонструє, що в разі некоректності оцінки модель продовжує рухатись за обраним маршрутом.

Таку поведінку моделі можна описати через гіпотезу, за якою процес виведення представлено в якості пошуку локального мінімуму в просторі станів. Відповідно, повідомлення від системи та користувача пересуваються цим простором у напрямку, що не має конкретного впливу на «енергію» стану (не гарантує ні підвищення, ні зниження). Проте, рухаючись від локального мінімуму, майже будь-які переходи підвищують цю енергію, змушуючи модель генерувати далі, але знову не гарантують залишення моделлю «басейну» навколо цього локального мінімуму. Окремий випадок – слабко пов’язані запити, «ортогональні» за якими нова генерація просто шукає мінімум за новою віссю. Таким чином, легко пояснити, чому окремі види довгих діалогів можуть мати навіть не негативний, а позитивний вплив на виведення (chain-of-thought), але не покращують здатність моделі до масштабування через ітерацію.

Проте, ситуацію можна певним чином виправити. Для наближення мовних моделей до такої моделі пошуку знову повертаємось до ідеї про розділення генерації та стану «пошуку». Якщо весь процес завжди складається з однієї генерації, то наведене протиріччя з недооцінкою стану «за замовчуванням» – відсутнє, бо тепер пошук є скінченним. Аналогічно, натуральний бектрекінг можливий, якщо метод здатний вплинути на вибір моделлю нової, наступної в черзі траєкторії, але за допомогою пам’яті про попередній вибір.

1.4 Постановка задачі

Сформувати та реалізувати методи масштабування під час виведення як для моделей типу «чорна скриня», так і моделей типу «біла скриня», що керуються виведеними правилами роботи з мовними моделями:

– розділення стану – збереження важливого контексту окремо від історії діалогу, окремі генерації з оновленими вхідними даними замість довгих траєкторій, що користуються пам’яттю процесу для дослідження нових шляхів міркування;

– активний вплив на вхідні дані, використання відгуків до попередніх спроб та інших методів диверсифікації результатів генерації для забезпечення ефективного дослідження простору рішень;

– атрибуція відгуків до конкретних дій моделі, в тому числі статистичними чи спекулятивними методами.

Оцінити отримані методи за ефективністю (зміна оцінки з кількістю ітерацій, використання токенів та відношення оцінки до кількості токенів) і складністю їх застосування (вимоги до моделі, середовища, завдання та інженера) та порівняти з існуючими методами масштабування у відповідних категоріях. Визначити переваги та недоліки отриманих методів, їх відповідність прогнозам на основі моделі пошуку-виведення та можливі рекомендації щодо їх застосування.

2 АНАЛІЗ СУЧАСНИХ МЕТОДІВ МАСШТАБУВАННЯ ПІД ЧАС ВИВЕДЕННЯ ТА ЇХ АДАПТАЦІЯ

2.1 Методи масштабування для моделей типу «чорна скриня»

Відповідно до визначення, моделі типу «чорна скриня» не надають можливості дізнатись проміжні результати своєї роботи або вплинути на них. Це значно ускладнює моніторинг за цікавими в контексті пошуку рішення метриками і дозволяє впливати на процес лише через зміну вхідних даних, що окрім тексту також часто включають параметри генерації, за якими можна налаштувати, наприклад, семплювання. Проте задача пошуку потребує більш адресного впливу на генерацію та аналіз її результатів.

2.1.1 Сучасні методи масштабування для моделей типу «чорна скриня»

Спочатку розглянемо вже існуючі методи масштабування під час виведення для моделей такого типу.

Best-of-N – наївний підхід, за яким генерується N різних відповідей, серед яких зовнішній оцінювач обирає найкращу [9]. Емпірично продемонстровано, що збільшення значення N підвищує точність моделі як за наявності точного верифікатора, так і при застосуванні принципу голосування більшістю. Проте ефективність голосування більшістю досягає «плато» за відносно низьких значень N, адже в такому разі збільшується не покриття простору рішень, а відповідність обраного рішення справжньому оптимальному рішенню моделі. Приклад того, як змінюється точність різних комбінацій моделі та способу визначення найкращого рішення зі збільшенням вибірки наведено на рисунку 2.1. Цю властивість використовують для одночасного забезпечення як вищої точності, так і стабільності.

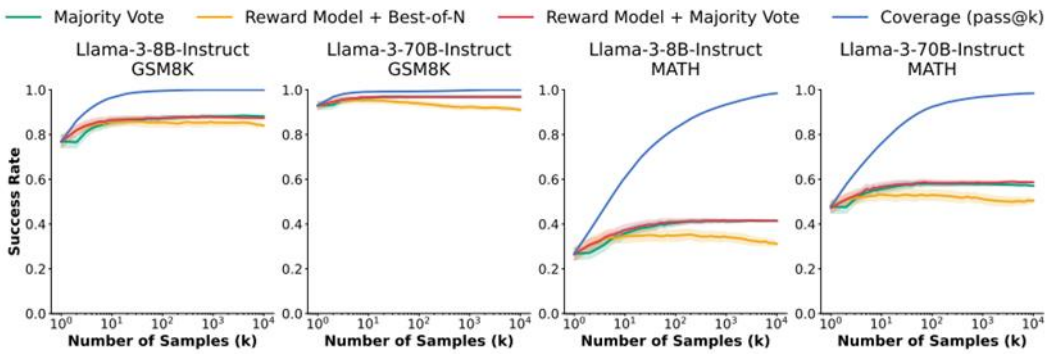


Рисунок 2.1 – Порівняння ефективності Best-of-N за типами верифікаторів

Self-refine полягає в використанні моделі для ітеративного покращення її ж відповіді. Після генерації нової відповіді вона передається моделі, яка генерує власний відгук з необхідними правками, що додається до діалогу [16], що схематично зображено на рисунку 2.2. Цй алгоритм пропонується як метод для оптимізації рішень, тобто вважається, що модель кожного разу вирішує задачу, але Self-refine дозволяє обрати найкраще з можливих рішень.

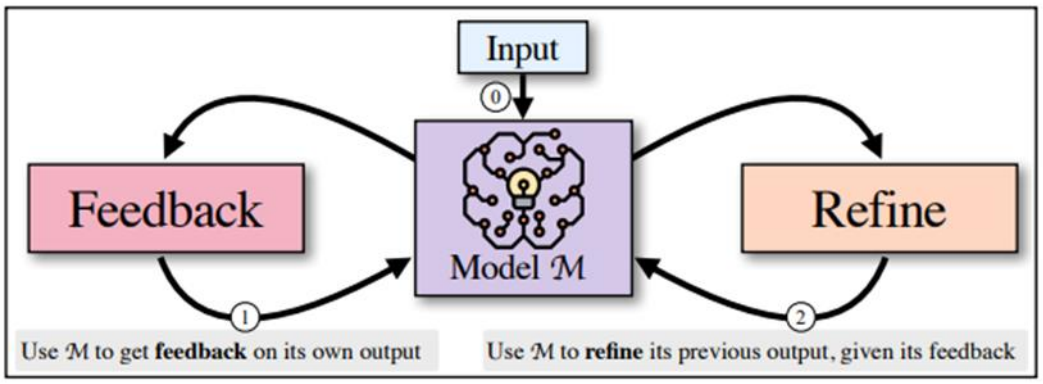


Рисунок 2.2 – Графічне представлення методу Self-Refine

Reflexion – підхід, схожий на Self-Refine, за яким в ході формування відгуку моделі також передається відгук користувача/системи на останнє запропоноване рішення [17].

За словами авторів, основна ідея в тому, що саморефлексія не просто засіб для оптимізації рішень моделі, а ключовий елемент роботи моделі з новими даними в якості агента зі стратегією, визначеною як кортеж з траєкторії/пам'яті та вхідних параметрів моделі. В дослідженні метод порівнюється із навчанням з підкріпленням, але на основі природної мови. Відповідне графічне представлення наведено на рисунку 2.3. Схематичне зображення LATS та SWE-Search наведено на рисунку 2.4.

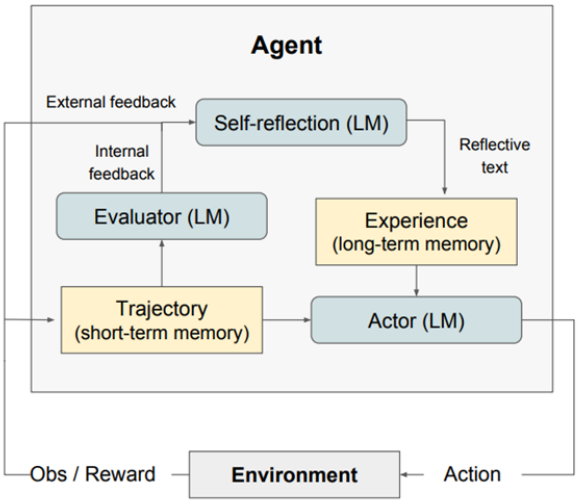


Рисунок 2.3 – Графічне представлення методу Reflexion

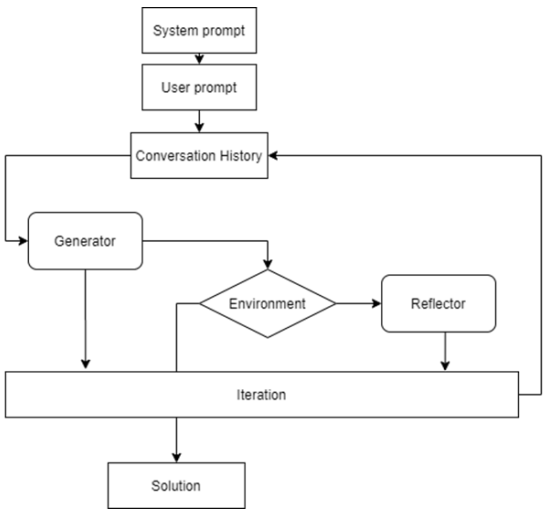


Рисунок 2.4 – Схематичне зображення алгоритмів Self-refine та Reflexion

LATS (Language Agent Tree Search) – підхід на основі пошукового алгоритму Monte Carlo Tree Search, за яким модель також генерує власний відгук на основі попередньої генерації і відповіді системи, проте такий відгук містить не лише текстову частину, але й оцінку яка і використовується в ході визначення пріоритетної траєкторії [18]. Після цього до обраного вузла застосовується операція «розширення»: генерується N альтернативних (за рахунок семплювання) рішень на основі попередньої траєкторії і наданого відгуку. В LATS пошук використовується як абстракція, що робить процес більш «модульним» та гнучким, розділяючи через вузли різні стани, дії та їх оцінки.

Алгоритм MCTS застосовується для дослідження середовищ, що складно оцінити аналітично. Відповідно до інших алгоритмів Монте Карло в ході такого пошуку через випадкове блукання формується певна статистична оцінка. У випадку MCTS виконується пошук в ширину, але обхід кожного з рівнів (а також вибір дій під час використання отриманого дерева) відбувається за пріоритетом який залежить від того, скільки разів вже було відвідано стан і наскільки успішними були траєкторії що його містили, який називають UCT (upper confidence threshold, дослівно «верхній поріг впевненості») і обчислюють відповідно до формули 2.1, де ліва частина відповідає за експлуатацію, а права – за дослідження:

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}, \qquad (2.1)$$

де i – індекс поточного кроку агента в середовищі;

w – кількість виграшів або сума оцінок траєкторій, що виходили з цього вузла;

n – кількість траєкторій, що виходили з цього вузла;

N – кількість симуляцій батьківського вузла;

c – коефіцієнт дослідження, що визначається емпірично.

Популярною є також варіація з використанням штучного інтелекту, за якою в процесі пошуку застосовують модель яка оцінює стани, а замість повноцінного пошуку в ширину розглядаються лише кілька найкращих за оцінкою моделі станів, що також впливає на пропорції при визначенні оцінки з дослідження. Відповідну оновлену формулу називають PUCT [19], або UCT передбачення:

$$PUCT = \frac{w_i}{n_i} + c\,P(i,a)\frac{\sqrt{\sum_b N_{(i,b)}}}{N_{(i,a)}+1}, \tag{2.2}$$

де a – дія, за якою відбувається переміщення з батьківського вузла до оцінюваного;

P – оцінка стану моделлю;

N – кількість траєкторій батьківського вузла де у ньому було обрано певну дію.

Підхід LATS є дуже схожим, адже орієнтується на оцінку мовної моделі при виборі дій в ході генерації. Станом в LATS вважається діалог, як послідовність з оригінальної інструкції та наступних дій (виведення моделі) і спостережень (відгук середовища). Кожен відвіданий стан – вузол у дереві пошуку. Графічне представлення наведено на рисунку 2.5.

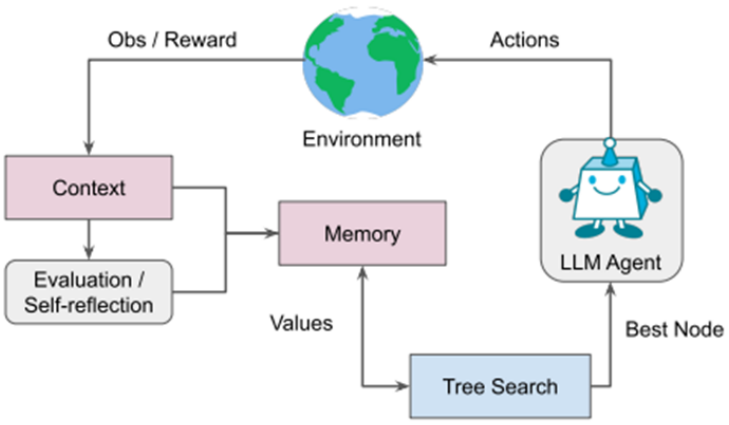


Рисунок 2.5 – Графічне представлення методу LATS

Але, якщо «розгорнути» частину процесу, що відповідає пошуку за деревом станів отримаємо алгоритм, що складається з шести кроків:

– вибір вузла, з якого починати з використанням UCB;

– розширення – семплювання N можливих наступних станів;

– попередня оцінка отриманих станів;

– симуляція – виконання кроків 2 та 3 до досягнення термінального стану;

– зворотнє поширення – виставлення коректних оцінок в усіх пройдених вузлах;

– рефлексія – модель робить текстовий аналіз виконаної роботи.

SWE-Search – метод на основі LATS із багатьма покращеннями, створений передусім для агентів-кодерів [20]. По-перше, пріоритетним є надання відгуків із чіткою числовою оцінкою (наприклад, процент виконаних тестів), яка заміняє оцінку надану відгуком моделі. По-друге, в ході «розширення» використовується не повна траєкторія, а лише остання генерація і відгуки до неї. Саме відгуки, адже операція «розширення» у SWE-Search відбувається поступово, а моделі надаються відгуки не лише до батьківського рішення, а й до «сестринських». По-третє, симуляція до досягнення термінального стану не виконується. Також в процесі генерації відгуків рекомендується застосування не однієї моделі, а консиліуму з кількох моделей (умова ефективності аналогічна подібній в ансамблевих методах – моделі мають сперечатися). Тобто диверсифікація через вибірку в LATS заміняється чи доповнюється диверсифікацією відгуків. Описана в LATS гнучкість такого підходу теж має місце, адже дозволяє SWE-Search обирати і відмічати вузли відповідно до характеру дії (планування, збір інформації, генерація програми, створення тестів). Графічне представлення методу наведено на рисунку 2.6. Загальну схему роботи алгоритмів LATS та SWE-Search наведено на рисунку 2.7.

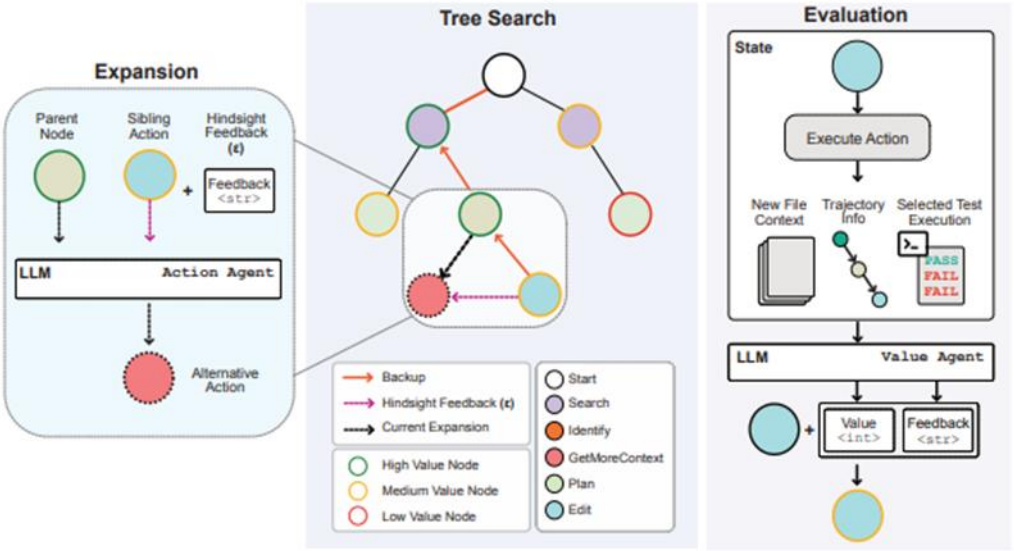


Рисунок 2.6 – Графічне представлення методу SWE-Search

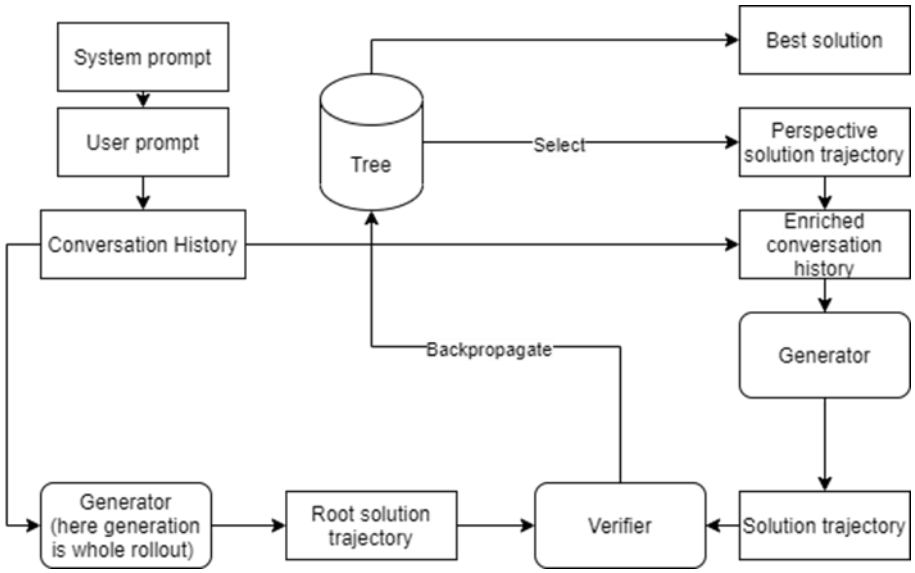


Рисунок 2.7 – Схематичне зображення алгоритмів LATS та SWE-Search

На основі принципів роботи цих рішень та їх ефективності можна зробити наступні висновки.

Модель «розумніше» ніж може здатися при генерації однієї відповіді. Саме це демонструє підхід Best-of-N, що явно вказує на те, що «перебором» через семплювання можна досліджувати різні можливі рішення, причому окремі дослідження [9] вказують на те, що в окремих випадках це дешевше

за використання більших моделей. Дуже ймовірно, що такий результат є наслідком того, що коли модель має обирати певну траєкторію, то найбільш ймовірні траєкторії – це список коректних альтернатив, упорядкованих за частотою відповідно до тренувальних даних. Ефективною в такому разі буде диверсифікація, що розглядає подібні траєкторії більш рівномірно. Це узгоджується і з іншими відкритими проблемами штучного інтелекту. Наприклад, навчання глибоких штучних нейронних мереж часто відбувається з використанням механізмів зворотнього поширення та градієнтного спуску, які не гарантують досягнення глобального мінімуму, проте компенсують це кількістю параметрів у просторі і, відповідно, можливих напрямків для мінімізації. Це схоже на авторегресивне виведення великих мовних моделей і також спонукає до створення методів, що дозволяють досліджувати простір на наявність траєкторій, кращих у довгостроковій перспективі [21]. Взагалі, такі результати вказують на велику роль, яку грає диверсифікація в процесах масштабування під час виведення.

Тим більше, що саме висока ефективність у запам’ятовуванні є характерною ознакою механізмів уваги й саме вона забезпечує головні сильні сторони великих мовних моделей. В ході даного дослідження розглядається ідея, що вивчені моделлю патерни є значно більш важливою стороною саме для мовних моделей, а їх здатність до навчання в контексті порівняно слабка і також забезпечується механізмом уваги. Відповідно до цього погляду на мовні моделі в запропонованих методах не використовуються довгі діалоги, а станом опікується зовнішній контролер. Це відрізняється від підходів Self-refine та Reflexion, які покладаються саме на здатність моделі покращувати власні рішення через формалізацію роботи з відгуками, але на практиці такі рішення демонструють помірний приріст ефективності.

Також можна побачити, що вже існують методи на основі пошукових алгоритмів, і вони переважно покладаються на алгоритм Monte Carlo Tree Search. Цей алгоритм часто застосовується в задачах пошуку де складно чи неможливо оцінити «якість» стану, не досягаючи термінального стану. В ході виконання пошуку застосовуються «випадкові обходи», після завершення яких оновлюється оцінка вже відвіданих станів, що включає як оцінку досягнутих термінальних станів так і повноти досліджених піддерев для балансування дослідження та експлуатації. Зрозуміло, що робить такий алгоритм привабливим: можна користуватися «випадковим обходом», виконаним через семплювання аби дійти термінального стану і оцінити його. Для того, щоб алгоритм працював, не треба знати, чому саме ця генерація була кращою за інші, достатньо спробувати зробити схожу на неї. І саме це дозволяє з часом отримати атрибуцію через статистику. Навіть якщо в загальному вигляді такий підхід погано відповідає цілі дослідження, що полягає в активному пошуку конкретних причин переваги одних рішень над іншими і використання цього знання.

Відповідно до проблематики дослідження генерація вважається не зовсім надійною, через що і може потребувати ітеративного покращення. Це певним чином ускладнює роботу з методами на основі інженерії контексту, які покладаються на використання багатьох допоміжних генерацій разом із основною. В рамках запропонованої стратегії побудови методів масштабування можна додатково навести правила роботи із комплексними сценаріями, що включають кілька генерацій:

– генерація в один крок умовно надійна, особливо на достатньо «прямих задачах», таких як перекласти, переписати в іншому стилі, зробити підсумок. Відповідні задачі часто не потребують контексту окрім власне тексту, з яким мають працювати;

– кілька генерацій в один крок – більш надійні відповідно до особливостей семплювання;

– генерація в багато кроків – ненадійна, її краще розбивати на кілька генерацій в один крок.

Ці принципи покладаються на ефективність підходу з розгляду диверсифікованих рішень моделі без застосування рефлексії. При цьому зберігаються етапи, коли модель має виконати переписування, оцінку тощо, адже для них весь відповідний контекст наведений у завданні.

2.1.2 MCTS-Rewrite

З урахуванням виведених корисних практик пропонується наступна версія механізму масштабування під час виведення на основі MCTS:

– як і в SWE-Search використовується числова оцінка від системи і письмова від моделі;

– стан пошуку відокремлено від моделі, кожна генерація має лише два попередні повідомлення: системне та інструкцію. За великої кількості ітерацій це також означає менше використання токенів;

– для кращої диверсифікації на основі шаблонів типу оригінальна «інструкція + відгук» виконується не генерація, а переписування інструкції з метою уточнити завдання для конкретної моделі.

Схематично алгоритм зображено на рисунку 2.8.

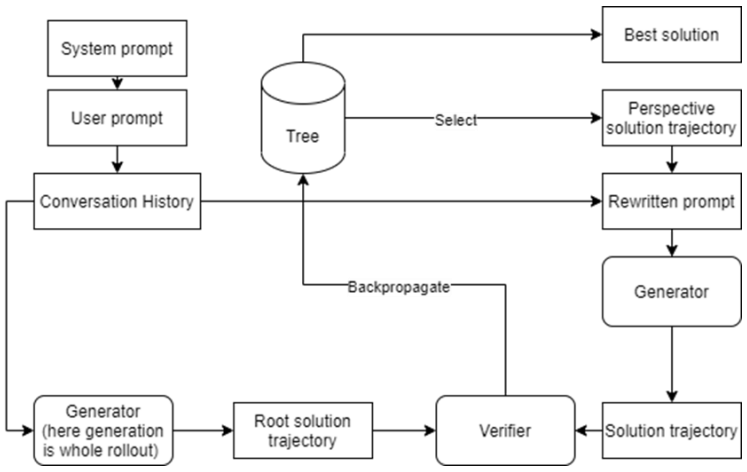


Рисунок 2.8 – Схематичне зображення алгоритму MCTS-Rewrite

За схемою можна зрозуміти, що отриманий метод дуже схожий на LATS та SWE-Search. Це дозволяє йому зберігати їх плюси, такі як гнучкість та статистична атрибуція. При цьому переосмислюється процес створення нових траєкторій. До того ж відсутність значних змін сприяє зручному аналізу ефективності методу переписування в ході експериментів.

2.1.3 LLM-CSP

Навіть маючи обмежені відомості про стан моделі, його все ще можна певним чином виділити силами самої моделі. Можна спитати модель про припущення, зроблені при написанні програми. Це жодним чином не гарантує правильну відповідь, але може бути наближеним до правди, що часто використовується методами з інтерпретування роботи мовних моделей. При виконанні такого кроку на основі діалогу зі збереженим префіксом під час повторного запуску модель не матиме даних про значення активацій, але зберігатиме значення матриці уваги, тож такий метод має працювати краще відповідно до покращення здатності моделі до рефлексії.

Інший крок алгоритму відповідатиме за визначення конфліктів між обмеженнями та змінними. Очевидно, що базу конфліктів потрібно розширювати, але розмір інструкції, її ясність також є важливими факторами в процесі генерації. Додатково, визначимо операцію оптимізації бази конфліктів, що дозволятиме видаляти дублікати, в тому числі такі, що не є еквівалентними, але де одна умова покриває усі наслідки з іншої.

В результаті після кожної генерації отримуємо відповідь, оцінку, спекулятивні оцінки прив’язаних змінних, виведених обмежень, конфліктів. На основі цього можна формувати стан пошуку, що зосереджений на збереженні типових для моделі конфліктів і їх вилученні з генерації шляхом зміни вхідних даних за допомогою операції переписування.

Важливою частиною такого підходу є припущення про те, що можливість значно покращити точність моделі використовуючи лише

багаторазове семплювання свідчить про концентрацію правильних відповідей у невеликій підмножині виборів (тобто насправді рівень розгалуження доволі низький), а значить «відрізання» невдалих спроб може бути доволі ефективним. Фінальне схематичне зображення алгоритму наведено на рисунку 2.9.

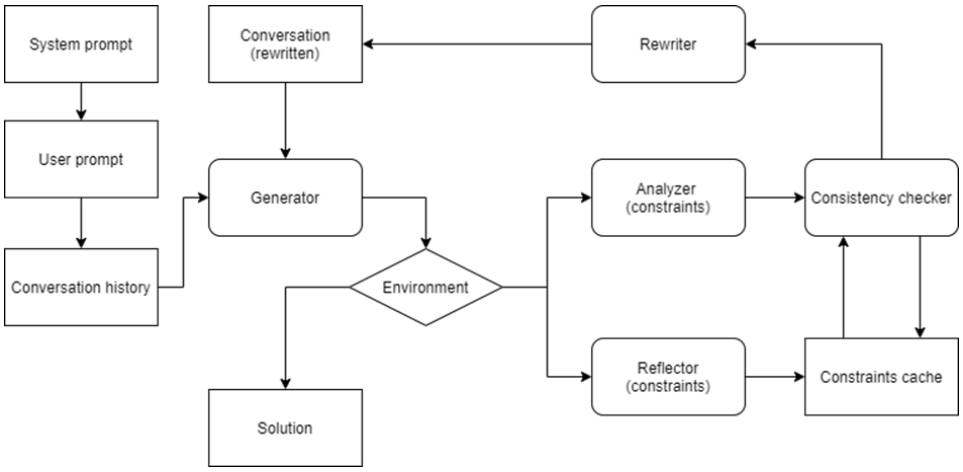


Рисунок 2.9 – Схематичне зображення алгоритму LLM-CSP

2.2 Методи масштабування для моделей типу «біла скриня»

Наявність достатніх обчислювальних потужностей дозволяє користувачам агентів на основі великих мовних моделей запускати такі моделі локально, що підвищує безпеку, контрольованість і часто є дешевшим варіантом. Проте, серед відповідних моделей популярніші менші варіанти, тому для локальних моделей задача підвищення ефективності є особливо важливою.

Моделі типу «біла скриня» дозволяють напряму втручатись у процес генерації, в тому числі й дізнаватись результати роботи окремих шарів моделі. На основі цих можливостей можна отримати більш ефективні або економні варіанти алгоритмів для моделей типу «чорна скриня». Але варто також зважати на принцип роботи мовних моделей у режимі виведення – для прискорення граф обчислень зазвичай компілюється, що робить процес

менш гнучким і сильно впливає на реальну ефективність та працездатність окремих видів впливу. Серед перспективних напрямів:

– контроль семплювання, тобто переходів у просторі станів;

– можливість порівнювати тексти за допомогою внутрішніх векторних представлень моделі;

– вплив на окремі групи «голів» моделі;

– застосування steering'у для корекції траєкторії моделі без засмічення текстового контексту.

2.2.1 Сучасні методи масштабування для моделей типу «біла скриня»

Спочатку розглянемо вже існуючі рішення.

Моніторинг ентропії – підхід до генерації відповіді, за яким зовнішній контролер спостерігає за ентропією та її варіацією у рішеннях моделі. Відповідно до визначення висока ентропія – ознака невизначеності моделі. Проте, за відсутності високої варентропії (якщо ентропія це середнє «здивування», то варентропія – варіація здивування) це зазвичай вибір між синонімами, близькими числами, але коли обидва значення високі – це не просто семантична невизначеність, а явна відсутність очевидного для моделі рішення. Це часто використовується в рішеннях, які оцінюють і забезпечують безпеку та правдивість результатів, які надає модель, як на прикладі з рисунку 2.10.

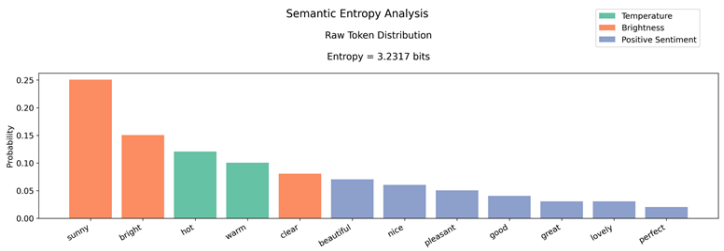


Рисунок 2.10 – Приклад застосування семантичного аналізу ентропії в рішеннях для оцінки правдивості відповідей моделі

При масштабуванні подібні методи використовуються разом із методами на основі міркування і також є популярними в алгоритмах адаптивного семплювання.

Process Reward Models – підхід до генерації, за яким процес виведення ділиться на сегменти (наприклад, речення, або за метриками) [22]. По завершенні кожного такого сегменту окрема модель, Process Reward Model («модель нагороди процесу»), надає явну оцінку узгодженості нового сегменту з усім попереднім текстом. Таку оцінку можна використовувати як під час навчання (наприклад, під час донавчання з використанням навчання з підкріпленням), так і при роботі з моделлю. Такий підхід є доволі цікавим, але може бути достатньо дорогим (умовно, на кожне речення моделі доводиться робити нову генерацію оцінки з PRM, а при пошуку – ще більше). До того ж PRM ніяк не впливають на диверсифікацію рішень моделі і лише аналізують її семантичні стани. На рисунку 2.11 зображено приклад агента, що використовує PRM та ORM.

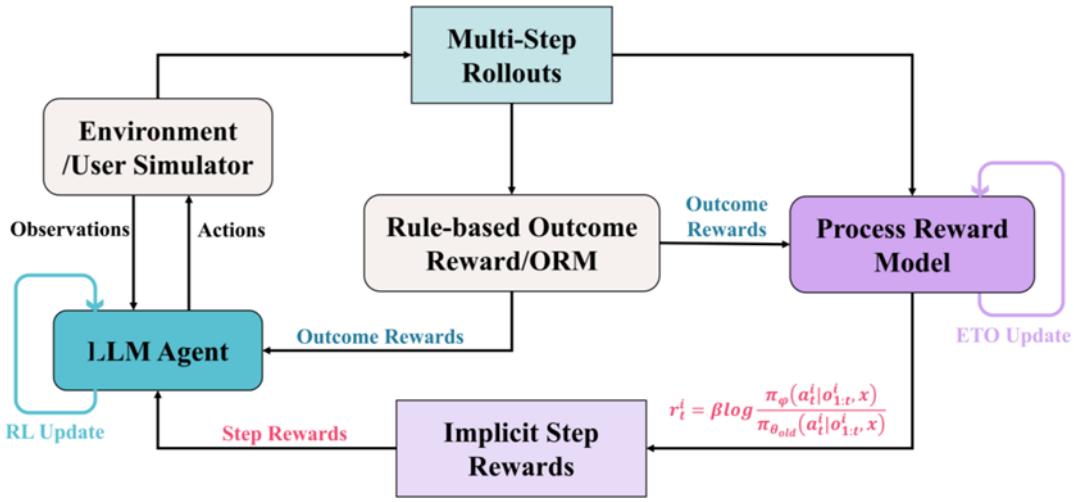


Рисунок 2.11 – Приклад схеми роботи LLM-агента із використанням PRM та ORM

На рисунку 2.11 можна також побачити ORM – модель, що передбачає оцінку для усієї траєкторії. В нашому випадку її аналогом є верифікатор, і

така модель є сумісною з моделями типу чорна скриня. Існують й інші типи моделей винагород, але вони достатньо подібні до PRM.

Steering (руління) – збірна назва для методів, що використовують отримані особливим чином вихідні дані моделі для того, щоб підвищувати вираженість окремих властивостей, стилю тощо моделі. Серед таких методів: ActAdd [23] (отримання вектору поведінки як різниці між текстом-прикладом та середнім значенням моделі або контр-прикладом і його ін'єкція в активації моделі), Head attribution (визначення голів у механізмі уваги які найбільше впливають на певну поведінку й зміна внеску таких голів), Patching [24] (пряма заміна значень виведених моделлю на значення підготовлені раніше) та інші. Такі методи дозволяють впливати як на стильову поведінку моделі, так і на більш складні взаємодії: переклад, виведення синонімів [25], перефразування [26] і навіть міркування як у відповідних моделей. Приклад роботи методу із цієї групи наведено на рисунку 2.12.

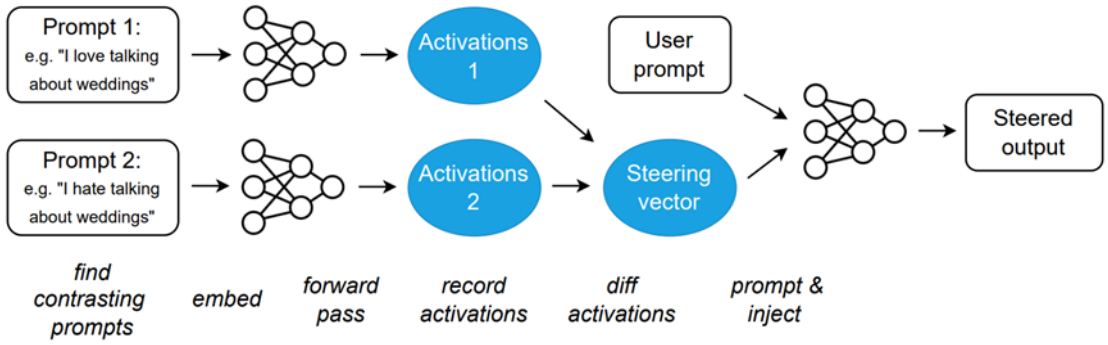


Рисунок 2.12 – Схема роботи методу ActAdd

В якості рішення для моделей типу «біла скриня» послідовно пропонуються три механізми: для диверсифікації, запам'ятовування та кращої атрибуції рішень моделі з використанням евристик на основі внутрішніх значень моделі. Кожне з рішень інтегрується поступово і

працює таким чином, що з часом евристичні очікування замінюються на статистичні.

2.2.2 Entropy bruteforce

Користуючись механізмом знаходження потенціальних розгалужень на основі ентропії і варентропії та припущенням про те, що оптимальна відповідь зазвичай присутня серед найбільш ймовірних, можна побудувати альтернативний підхід до семплювання. З доступом до активацій моделі можна ефективно групувати вже відвідані стани моделі, кластеризуючи їх у режимі online з використанням скалярного добутку (що в даному випадку аналогічний косинусній відстані). Також на основі активацій можна отримувати більш інформативні розподіли логітів з шарів всередині моделі завдяки техніці, відомій як logit lens [27], за якою активація одразу подається до голови-класифікатора.

Таку кластеризацію можна будувати за принципом системи неперетинних множин – структури даних, що дозволяє в режимі онлайн розділяти універсум на неперетинні множини і асоціювати кожну множину з певним набором метаданих. У кожної множини, є представник який і є справжнім ключем у асоціативній структурі даних, а для кожного елемента множини надається спосіб отримати посилання на представника своєї множини. При додаванні нового елемента до системи він порівнюється з представниками, після чого приєднується до однієї з множин чи утворює нову.

У наведеному варіанті таку операцію легко векторизувати як операцію скалярного добутку. Механізм «кластеризації» також можна зробити дуже простим. Через відсутню необхідність зберігати конкретні отримані моделлю приховані стани роль усього кластеру може відігравати центроїд – штучний вектор, що відповідає усередненому значенню всіх векторів, що потрапляли до кластера.

За допомогою такої структури даних можна зберігати історію прийнятих рішень (виведених токенів) і користуючись нею робити більш агресивну диверсифікацію аніж звичайне семплювання, виключаючи вже відвідані траєкторії, або знижуючи відповідні їм ймовірності відповідно до їх «свіжості».

Лістинг 2.1 – Псевдокод методу Entropy-bruteforce

```
Initialize Model, Evaluator, DSU: M, E, D
completion = []
Do
     Do
          Intercept activation
          activation = Normalize(activation)
          If Entropy(activation) > threshold
          context = D(activation)
          activation -= RecencyPenalty(context)
          End
          token = Model(instruction||generation)
          context.append(token)
          completion.append(token)
     While token != eos_token
While not E(completion)
```

2.2.3 Entropy search

Користуючись створеною для попереднього механізму системою неперетинних множин можна зберігати не просто список токенів, що вже були використані, а представляти відповідні стани вибору в якості направленого графу, який можна використовувати для пошуку послідовності виборів натхненної MCTS, але з певними модифікаціями:

– замість UCT використовується PUCT (2.2), оцінка, що також включає передбачення моделі;

– на відміну від MCTS розширення не має відбуватись за усіма можливими діями, замість цього застосовується аналогічний до entropy bruteforce механізм штрафування токенів;

– усі невідвідані токени також мають власну загальну оцінку, відповідну їх загальній ймовірності. Якщо їх оцінка вища від оцінки відвіданих відбувається дослідження через розширення;

– використання графа, а не дерева, дозволяє краще працювати із невизначеною послідовністю дій.

2.2.4 Entropy CSP

Попередні методи використовують ентропію моделі для активної диверсифікації рішень та виконують атрибуцію через оцінку фінального результату. Таку атрибуцію можна покращити, припускаючи або активно забезпечуючи відношення часткового порядку відносно оцінок роботи моделі. Наприклад, якщо це задача з програмування з набором тестів, які виконуються послідовно, то відношення часткового порядку присутнє. В такому випадку можна казати про те, що:

– якщо всі шляхи, що йдуть з певного вузла мають оцінку x, або нижчу, то є висока ймовірність, що рішення пов’язане із відповідним станом – причина;

– ймовірність тим вища, чим частіше шляхи, що йдуть з цього вузла ведуть до «листів» з відповідною помилкою;

– але тим нижча чим різноманітніші кінцеві вузли шляхів;

– очевидно, що варто враховувати і «батьківські» вузли, але зважаючи на те, що це механізм семплювання «батьківські» вузли вже пройдено.

Для того щоб реалізувати відповідний механізм, пропонується зберігати список усіх отриманих винагород і на їх основі множити оцінку UCT вузла на обернену ймовірність того, що рішення напряму пов’язане із помилкою, яка обчислюється за формулою 2.3:

$$P_{fallacy} = \left(\frac{N_{max}}{N}\right) \cdot \max\left(0, 1 - e^{-\frac{N_{max}-1}{\tau}}\right), \quad (2.3)$$

де R_{max} – нормалізована максимальна нагорода;

N – кількість шляхів, що йдуть через вузол;

N_{max} – кількість шляхів, що відповідають максимальній винагороді;

τ – спеціальна константа.

Отриману загальну для методів Entropy search та Entropy CSP схему наведено на рисунку 2.13.

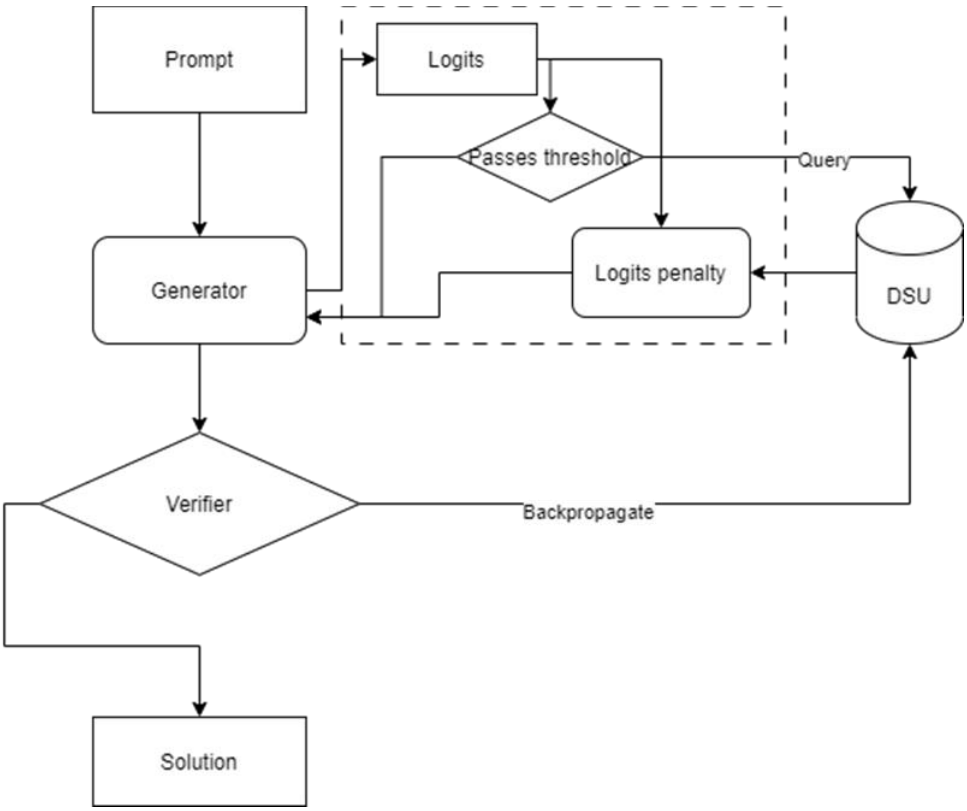


Рисунок 2.13 – Схематичне зображення алгоритмів Entropy-Search та Entropy-CSP

2.2.5 Додаткові переваги запропонованих методів

Загальна простота методу дозволяє швидко інтегрувати його в сучасні рішення для навчання або використання мовних моделей. Однією з

основних переваг є відсутність взаємодії з механізмом уваги і детермінованість рішень в ході однієї генерації. Завдяки цьому метод також легко застосовувати у рішеннях, що орієнтуються на високу швидкість через компіляцію графу обчислень. Для цього достатньо попередньо обчислити значення штрафів для кожного зі станів і включити до значень, що повертає модель необхідні для обчислення наступної ітерації значення активацій.

Ще однією перевагою запропонованих методів є відносна простота масштабування такого підходу та можливість його композиції як з методами семплювання, так і з підходами на основі інженерії контексту (наприклад, розглянуті методи для моделей типу «чорна скриня»).

Під масштабуванням підходу мається на увазі можливість виконання такого пошуку розподіленими системами. Фактично для цього необхідно забезпечити лише принцип розподілу задач між виконавцями для запобігання виникненню однакових траєкторій, наприклад, дослідження не найкращого варіанту, а того, що відповідає рангу моделі, визначення підмножини станів якою опікується виконавець або введення факторів на основі випадковості, таких як семплювання або збурення гіперпараметрів методу.

Наведені методи легко композувати із іншими, адже результатом роботи методу є лише зміна значення активації, причому опціональна. Для методів з використанням інженерії контексту необхідно лише явно прив’язати окремі етапи роботи до стану пошукового алгоритму. Наприклад, для методів схожих на LATS, Entropy-CSP може бути основою для визначення оцінки стану чи навпаки – джерелом диверсифікованих генерацій під час розширення дерева.

Траєкторії, які створює Entropy-CSP, також є важливою перевагою. В них змінено лише невелику кількість токенів, що дозволяє виділяти конкретні помилки моделі, зберігаючи розподіли, типові для моделі. При дотренуванні чи навчанні з підкріпленням такі дані можуть надати меншу

кількість більш якісних прикладів, що вплине на швидкість та ефективність дотренованої моделі. До того ж, у дані для тренування можна включити метадані графової моделі і визначити як ціль не виведення конкретного токена, а наближення розподілу ймовірностей до статистичного, отриманого через MCTS. Траєкторії можуть мати позитивний вплив і на сам пошук, якщо використовувати їх в якості додаткової попередньої оцінки (аналогічно до тієї, що надається моделі).

Слабким місцем такого підходу є робота з рефлексією, адже вона відповідає схожим на цільовий станам і також асоціюються з високою ентропією. В таких випадках метод пошуку знову спрацьовуватиме, що може заважати роботі як самого пошуку так і рефлексивних механізмів.

2.3 Особливості запропонованих методів

Методи MCTS-Rewrite, LLM-CSP та Entropy CSP використовують велику кількість елементів, вже присутніх у інших методах і перевірених на практиці. Втім, сама необхідність адаптації сучасних методів масштабування під час виведення вказує на суттєву різницю між стратегією масштабування на основі пошуку та іншими існуючими підходами. За нею пошуком рішень займається не модель, а алгоритм масштабування, що використовує методи атрибуції для експлуатації та диверсифікації для дослідження. Така структура є досить гнучкою, що дозволило легко адаптувати і комбінувати існуючі методи в робочі рішення.

Наприклад, метод SWE-Search виконує атрибуцію через MCTS, а також відділяє загальну історію взаємодії від тієї, що надається моделі. Проте, підхід до диверсифікації рішень, що розглядаються, покладається лише на відгуки середовища. У запропонованому методі MCTS-Rewrite на цьому етапі застосовується переписування моделлю інструкції.

Аналогічно, методи на основі семплювання не покладаються на попередню історію і зосереджуються саме на диверсифікації відповідей, що

генеруються. Проте, такі методи не мають пам’яті, через що не здатні до атрибуції результатів роботи моделі до прийнятих рішень. Алгоритм Entropy CSP пропонує підхід схожий на адаптивне семплювання, але із можливістю зберігати траєкторії у графовій моделі та на її основі виконувати атрибуцію.

2.3.1 Методика застосування в складніших агентних задачах

Однією з визначних особливостей представлених в роботі методів є відділення стану виконання задачі та діалогу з моделлю, а саме підхід, коли в діалозі є лише одне повідомлення від користувача. Такий підхід зрозумілий при застосуванні до завдань з одного кроку, але сучасні сценарії виконання часто включають завдання з кількох етапів. Наприклад, популярний бенчмарк SWE-Bench [28] надає агенту доступ лише до файлової системи програмного проєкту і опис задокументованої проблеми, а ціллю є формування текстового файлу у форматі .diff, що описує необхідні зміни в репозиторії. Такий сценарій як мінімум включає аналіз проблеми, пошук релевантних файлів, формування файлу і взаємодію з автоматизованими тестами проєкту.

В таких випадках масштабувати ефективність треба все ще на весь процес, а не на лише окремі його частини, аби впевнитись, що в ході масштабування не виконуються зайві дії. Така ситуація можлива, якщо, наприклад, перезапускати увесь сценарій заново за помилки. Для запропонованих методів скоріше підійде підхід з «версіями» відокремленних станів пошуку. За ним між конкретними даними в станах кожного однокрокового етапу є зв’язки, що визначають, які налаштування треба «відкотити» при зміні процесу-попередника. Це потребуватиме доволі складних механізмів, як локалізації помилок, так і визначення їх першопричин.

Водночас для методів, запропонованих для моделей типу «біла скриня», легко визначити підходи до розподіленого виконання та композиції, дозволяючи масштабувати як складніші задачі, так і адаптуватись до більшої кількості додаткових обчислювальних ресурсів завдяки графовому представленню задачі. А методи на основі LATS (що доведено SWE-Search) демонструють відповідну гнучкість у траєкторіях вищого рівня абстракції (вузол-генерація) завдяки можливості додавати вузли, що відповідають різним діям (аналіз, планування, розвязання).

3 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ ТА ЇХ АНАЛІЗ

3.1 Методика тестування

В ході експериментів буде порівняно базовий (продовження діалогу) та альтернативні підходи до масштабування під час виведення:

– Best-of-N;

– Reflexion;

– LATS;

– SWE-Search;

– MCTS Rewrite;

– LLM-CSP;

– Entropy bruteforce;

– Entropy search;

– Entropy CSP.

В ході тестування використовуватимуться власні реалізації наведених методів на основі відповідних оригінальних статей та коду. Реалізація SWE-Search не є повною відповідністю через те, що цей алгоритм передбачає застосування в задачах супроводження програмного забезпечення і виконує додаткові дії, які не стосуються виконуваної задачі. Усі реалізації та застосовані у них інструкції наведені у додатках.

Тестуватися методи будуть на задачах з програмування із набору даних LostInConversation [2], що серед інших задач містить вибірку олімпіадних задач із програмування разом з набором тестів з різних відповідних бенчмарків і містить в цілому 100 задач різного рівня складності та з різної тематики. В якості відгуку системи буде використовуватись власне результат проходження тесту. Це означає, що модель матиме доступ до достатньо точної функції оцінки, яка також додає певний обсяг додаткової інформації про помилки.

Вибірка моделей складається з моделей для локального використання з відкритими вагами з лінійок Llama 3.2 [29] та Gemma 3 [30]:

– Llama3.2-1B-Instruct;

– Llama3.2-3B-Instruct;

– Gemma-3-4b-it;

– Gemma-3-27b-it.

Ефективність методів буде оцінюватись на основі наступних метрик:

– Pass@N – кількість завдань розв’язаних за N спроб;

– Score@N – найвищий відсоток пройдених рішенням тестів серед N спроб;

– приріст точності зі збільшенням N, в тому числі його динаміка;

– використання токенів (хоч це і не точна метрика для визначення витрат на обчислення, вона є найбільш використовуваною постачальниками послуг з використання мовних моделей).

Варто зауважити, що результати роботи мовних моделей доступних через хмарні сервіси рідко є детермінованими. Для забезпечення рівності між моделями, що тестувались локально та через провайдера, замість використання seed-значень експерименти, що включають використання хмарних сервісів або семплінгу проводитимуться по кілька разів. У відповідному випадку поряд будуть зазначатись найкраще та середні значення.

3.2 Тестування запропонованих методів для моделей типу «чорна скриня»

Моделі працюють зі стандартними налаштуваннями семплювання: temperature = 1.0.

Моделі Gemma 3 використовуються через хмарний сервіс, для кожного методу вибірка складається з трьох експериментів. В таблицях для

відповідних моделей наведені середнє значення та стандартне відхилення, а на графіках відповідні дані позначенні діаграмою розмаху.

Моделі Llama 3.2 були запущені локально, з використанням формату bfloat16 на прискорювачах TPU v5e. Результати (кількість завершених завдань, середня кількість пройдених тестів, використання токенів) після 4 ітерацій наведено у таблицях 3.1–3.3 і відображено у діаграмах на рисунках 3.1–3.3.

Таблиця 3.1 – Результати Pass@4

Модель	Методи					
	Baseline	Reflexion	LATS	SWE-Search	MCTS-Rewrite	LLM-CSP
Llama 3.2 1B	20.0	18.0	12.0	21.0	**25.0**	18.0
Llama 3.2 3B	41.0	40.0	42.0	**43.0**	40.0	42.0
Gemma 3 4B	54.3 (±1.53)	56.3 (±0.58)	60.0 (±1.00)	63.7 (±1.53)	**64.7** (±2.31)	60.7 (±3.05)
Gemma 3 27B	77.0 (±1.00)	**83.3** (±0.58)	83.0 (±2.00)	81.0 (±1.00)	82.7 (±0.58)	80.7 (±0.58)

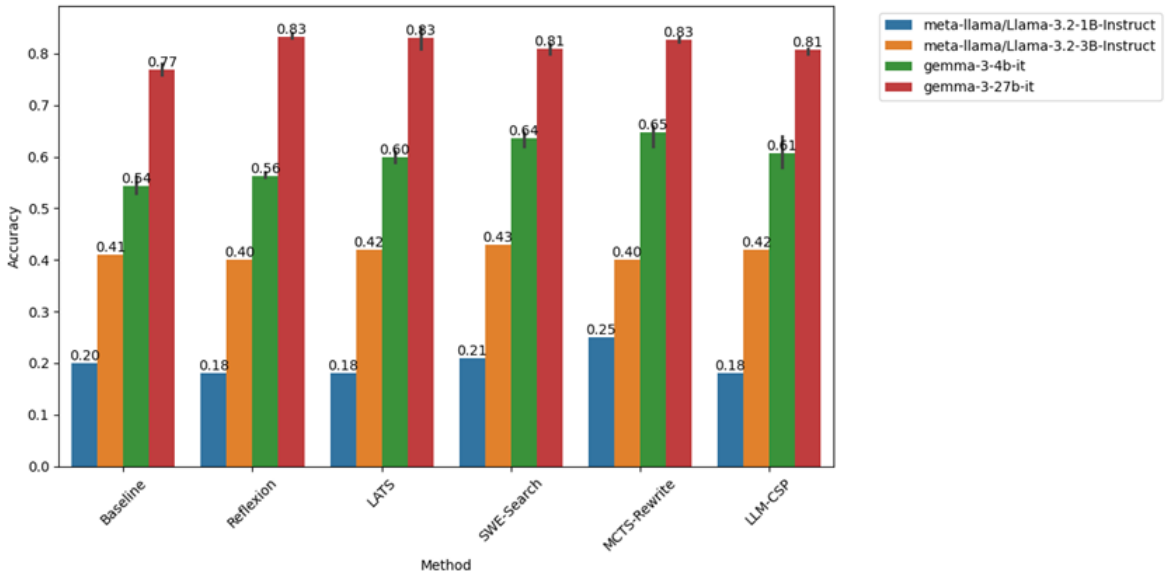


Рисунок 3.1 – Візуалізація розподілу Pass@4

Таблиця 3.2 – Результати Score@4

Модель	Методи					
	Baseline	Reflexion	LATS	SWE-Search	MCTS-Rewrite	LLM-CSP
Llama 3.2 1B	27.8	24.7	26.1	27.4	**31.6**	**31.6**
Llama 3.2 3B	50.2	49.6	48.4	49.4	50.8	**57.8**
Gemma 3 4B	66.6 (±1.71)	64.7 (±1.02)	71.0 (±0.84)	76.0 (±0.41)	76.5 (±2.30)	**79.7** (± 1.86)
Gemma 3 27B	89.0 (±0.73)	92.9 (±1.74)	90.3 (±2.05)	90.0 (±1.15)	91.4 (±1.45)	**93.8** (± 0.25)

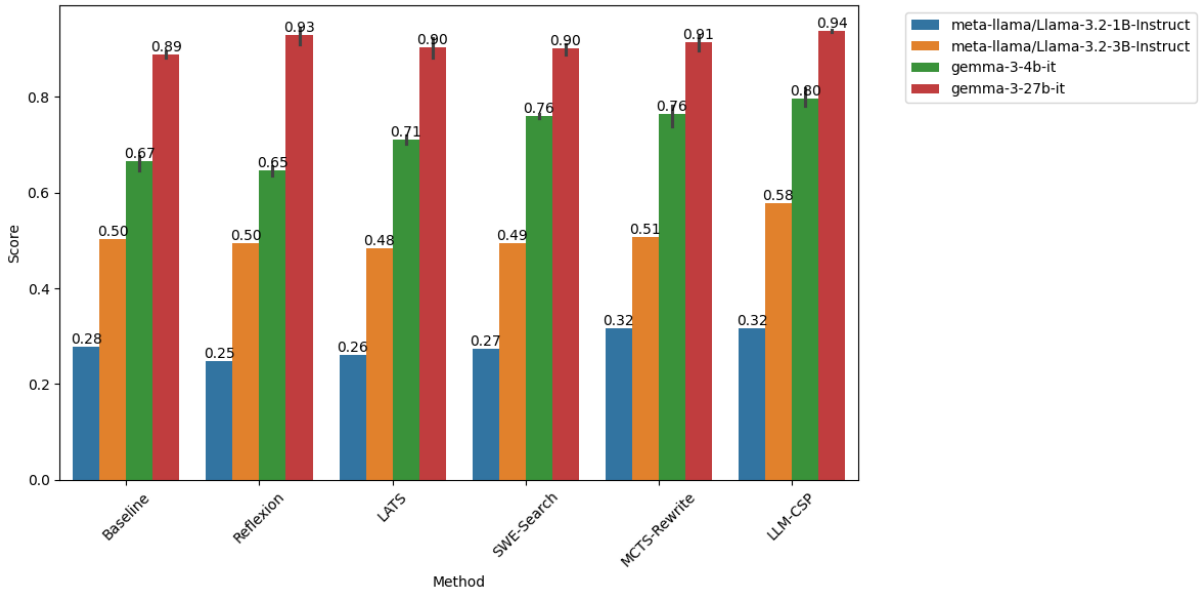


Рисунок 3.2 – Візуалізація розподілу Score@4

Таблиця 3.3 – Середня кількість використаних токенів

Модель	Методи					
	Baseline	Reflexion	LATS	SWE-Search	MCTS-Rewrite	LLM-CSP
Llama 3.2 1B	**5048**	14325	31387	29331	66015	59917
Llama 3.2 3B	**4224**	9260	18929	15164	14837	20057
Gemma 3 4B	**3194** (±236)	5267 (±282)	10082 (±550)	6580 (±263)	8198 (±651)	11063 (± 195)
Gemma 3 27B	**1790** (±56)	3010 (±216)	5901 (±754)	5003 (±558)	6588 (±257)	6566 (± 221)

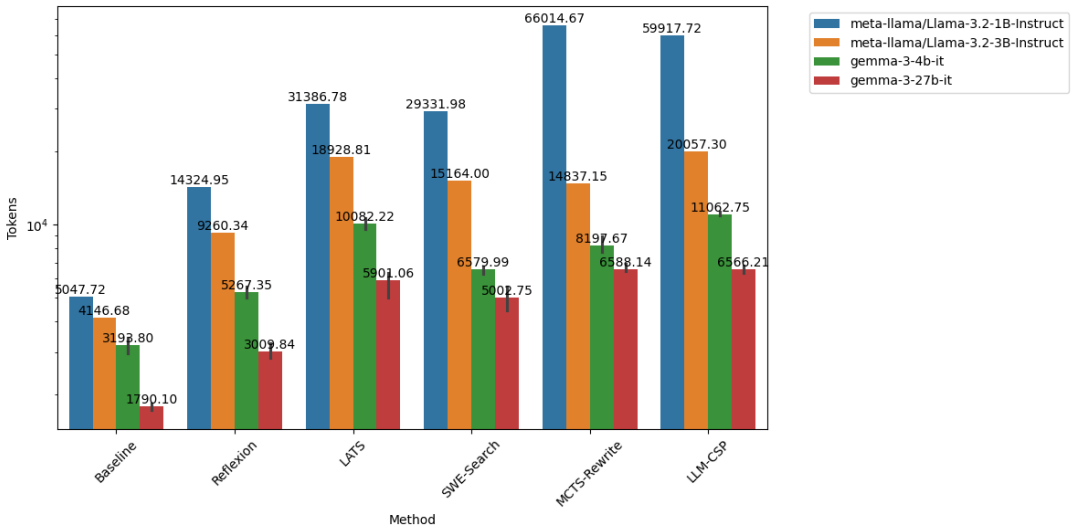


Рисунок 3.3 – Візуалізація розподілу кількості токенів

За результатами можна побачити, як послідовно новіші методи підвищують точність моделей, хоч це і не завжди корелює з підвищенням середньої оцінки методів. В той же час «ціна» методів виражена у токенах залежить лише від самих методів, що шукають баланс між завеликим засміченим і замалим недостатнім контекстами та кількістю дій, що потребують взаємодії з моделлю.

Запропоновані методи демонструють високу ефективність роботи порівняно із існуючими альтернативами і доводять відповідну ефективність принципів роботи з моделями, отриманими на основі моделі виведення-пошуку.

Алгоритм MCTS-Rewrite демонструє стабільно високу точність і завдяки цьому часто є найкращим чи другим кращим рішенням при доволі стриманому використанні токенів, що також робить його одним із фаворитів у відношенні ціна/точність.

Алгоритм LLM-CSP демонструє точність на рівні актуальних методів і найвищу середню оцінку, а його ефективність змінюється найбільше при переході до більших моделей. Пояснення такого результату може бути наступним. Можна припустити, що метод покращує здатність моделі до

забезпечення функціональної коректності програм, але не завжди активує знання моделі про алгоритми та структури даних, що відповідає структурі більшості тестів, що спочатку тестують програму на правильність відповідей, а вже потім на здатність отримувати їх швидко. Причому метод покладається на окремі рефлективні здібності моделі, що проявляються сильніше у великих моделей.

Можна побачити, що наведені методи використовують достатньо багато токенів. Це пояснюється тим, що на відносно невеликій кількості ітерацій ефект від застосування додаткового етапу замість продовження діалогу насправді є більш дорогою операцією, і це більш виражено у менших моделях, які часто є вербозними, а також більш схильні до вироджених станів. Побачити це можна на прикладі того як з різною кількістю етапів змінюється відношення кількості токенів між методами на рисунку 3.4.

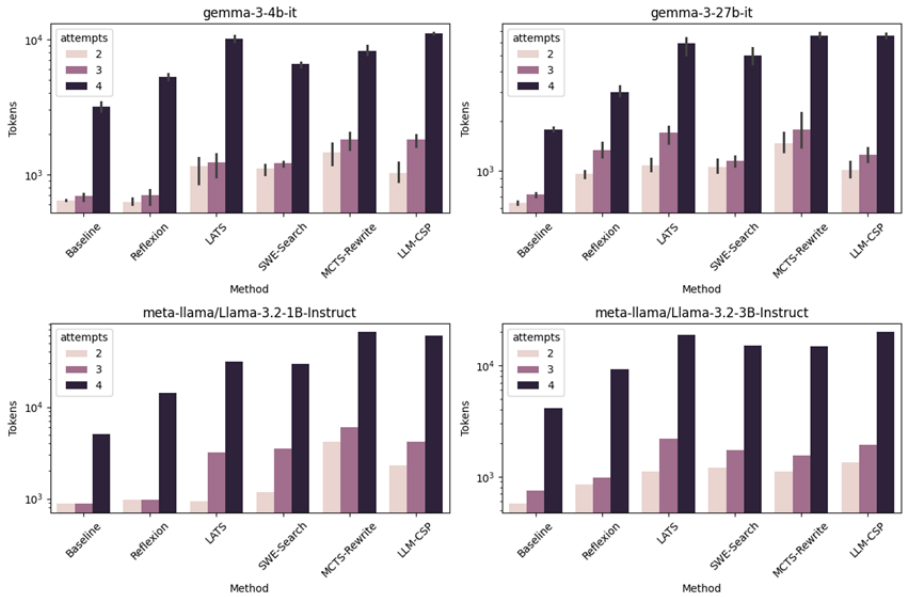


Рисунок 3.4 – Використані токени за кількістю ітерацій

Згідно з рисунком 3.4 зростання методів на основі продовження діалогу є більш стрімким. Також видно, що на кількість використаних

токенів впливає і складність задачі – категорія, що відповідає 4 спробі включає переважно неуспішні генерації, які часто є багатослівними або виродженими.

На рисунку 3.5 зображено графіки відношення точності до кількості токенів.

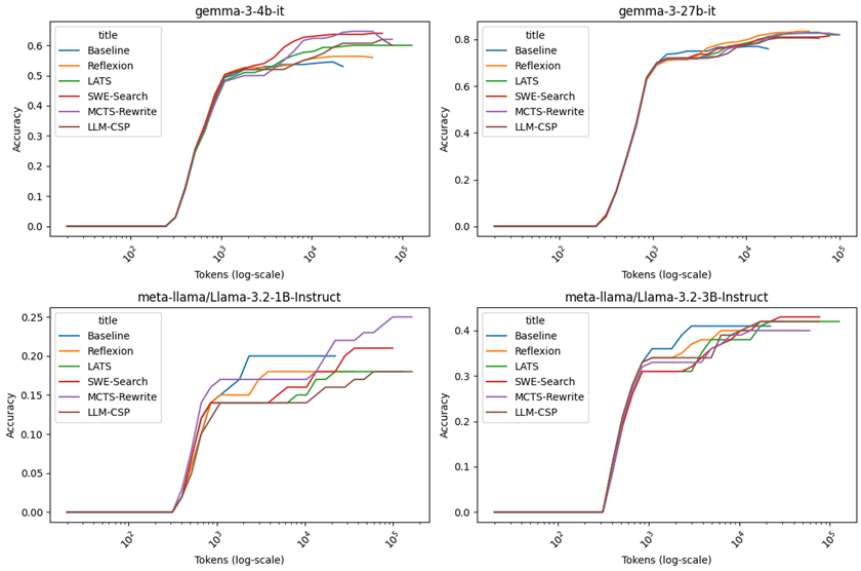


Рисунок 3.5 – Відношення точності до кількості токенів за методами

Згідно з рисунком 3.5 створені методи демонструють не тільки вищу точність, ніж базове рішення, але також є ефективнішими за нього в розрахунку за точністю на токен. Наведені методи також добре проявляють себе порівняно з іншими методами до масштабування для моделей типу «чорна скриня». Можна побачити, що для більших моделей методи на основі інженерії контексту є більш дієвими і швидко обходять наївну взаємодію. Це також збігається з загальним зростанням лаконічності моделей. Такий графік є більш репрезентативним у демонстрації їх ефективності, особливо зважаючи на типову багатослівність моделей у задачах, які вони не здатні вирішити, що на графіку відповідає довгим «хвостам».

На рисунку 3.6 можна побачити, як змінюється вплив кожного з методів зі зміною кількості ітерацій.

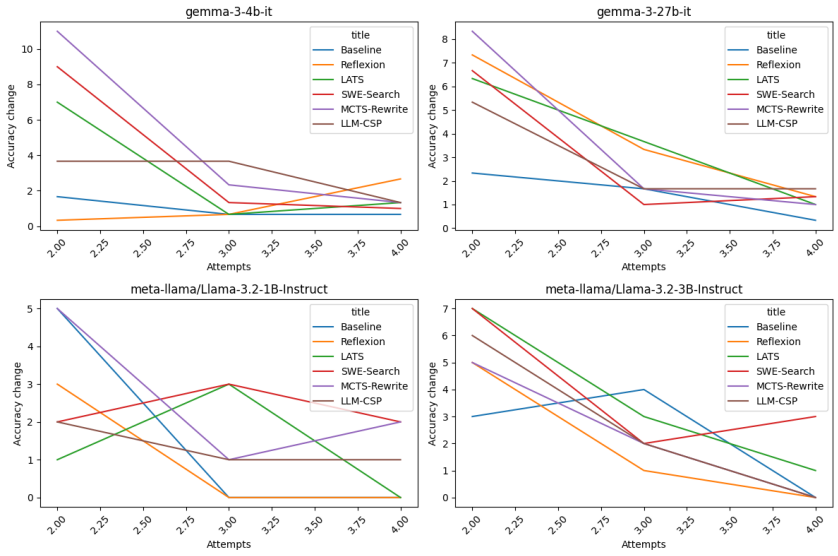


Рисунок 3.6 – Підвищення точності з кількістю ітерацій

За графіками видно, що більшість методів дуже добре «стартують», після чого з кожним кроком кількість нових вирішених задач зменшується. Подібну поведінку демонструє й метод MCTS-Rewrite, висока ефективність якого полягає в значному покращенні після першої ітерації. Метод LLM-CSP є більш стабільним, але це може бути пов’язано із більшою кількістю простіших (порівняно з іншими методами) задач.

3.3 Тестування запропонованих методів для моделей типу «біла скриня»

3.3.1 Визначення гіперпараметрів для методів на основі ентропії

Алгоритм на основі ентропії є достатньо стійким до змін у гіперпараметрах, завдяки тому що окремі значення використовуються як ініціалізатори та з часом налаштовуються адаптивно. В той же час вибір

шару, активації якого будуть використані, оптимальної температури для зважування дій за оцінкою моделі і початкових значень є важливими для роботи алгоритму.

Для отримання відповідних значень треба виконати аналіз поведінки моделі на вибірці даних, що відповідають цільовій задачі. На основі цієї вибірки формується набір даних з активацій моделі при генерації тексту за кожною із задач.

Першим обирається бажаний шар моделі, зазвичай це один із кінцевих шарів моделі, але, як правило, не останній – в останньому шарі й іноді у кількох попередніх ентропія логітів завжди є низькою, і її зміни іноді не відображають реальну невпевненість моделі. В якості критерію для вибору відповідного шару пропонується використовувати оцінку на основі таких параметрів, як кількість токенів, що співпадають з останнім шаром, якість кластеризації (коефіцієнт силуету) та кількість станів, що перетинають автоматично визначений поріг. Приклад аналізу на основі методу LogitLens показано на рисунку 3.7.

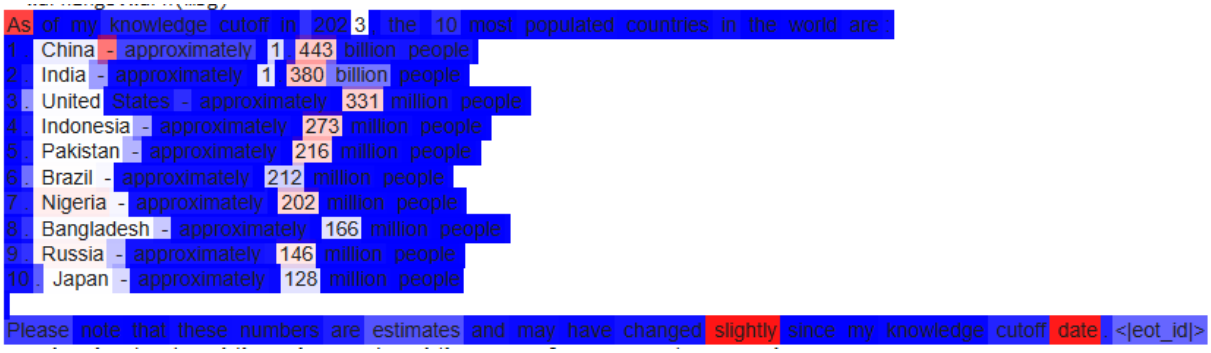


Рисунок 3.7 – Приклад застосування методу Logit Lens для вибору аналізу шарів мовної моделі

Далі обирається початкове порогове значення для визначення «критичних» станів. Відповідно до методу границя ентропії вибирається на основі логітів, отриманих з відповідного шару моделі, але температура

обирається за логітами усієї моделі. Фактично достатньо зберегти границю кластеризації, отриману при виборі шару. Далі границя буде зміщуватись автоматично відповідно до цільового відсотка цікавих станів (але лише при зменшенні порогового значення). В ході експериментів використовувались значення 0.15 для обох параметрів. Приклад відповідної кластеризації наведено на рисунку 3.8.

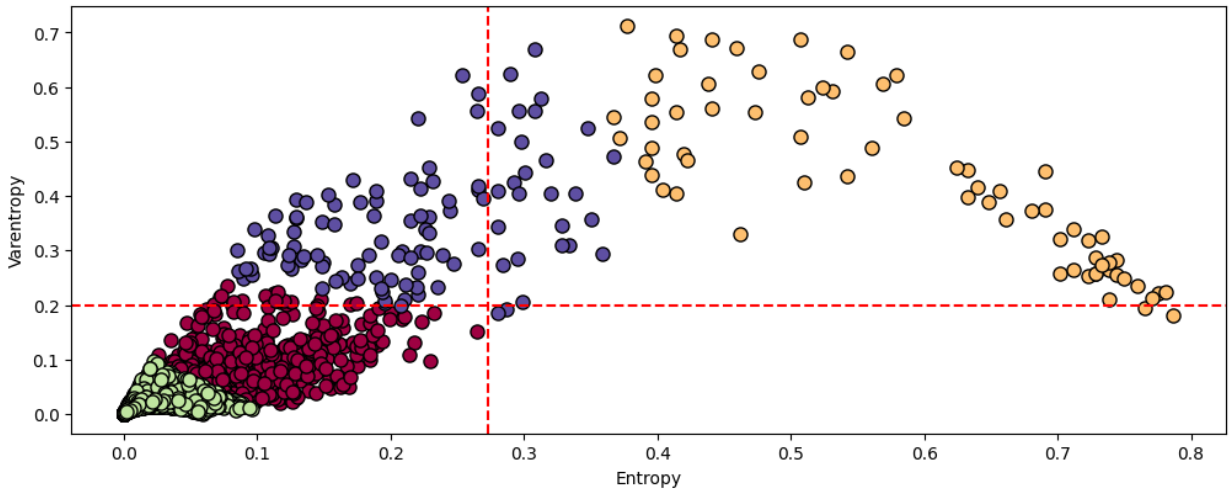


Рисунок 3.8 – Приклад кластеризації ентропії логітів моделі

Наступною обирається температура, яку використовуватиме, власне, алгоритм пошуку. Для її визначення будується графік залежності наступних метрик від температури для «критичних» станів: ентропія, ймовірність кращого токена, ймовірність n кращих токенів. Відповідно до числових констант, що застосовуються у методі, обираємо таке значення температури, для якого дія «дослідження» пріоритетніша за вибір найкращого токена. Для запобігання зайвому дослідженню обиратимемо перше значення температури, що задовольняє цю умову. Для використаних моделей це значення трохи вище за 4. Приклад графіка для вибору температури наведено на рисунку 3.9.

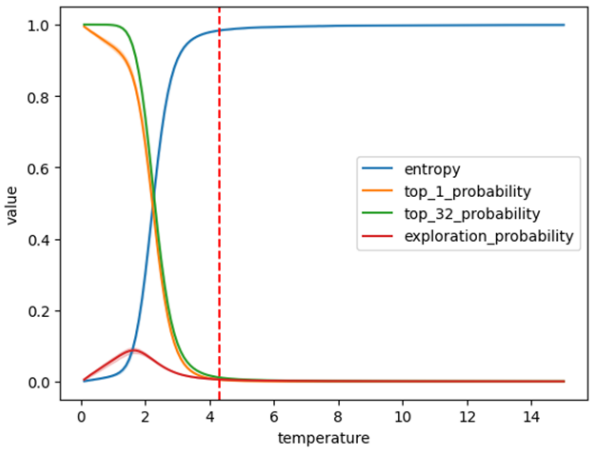


Рисунок 3.9 – Залежність основних метрик пошуку від температури

Також потрібно обрати правильне порогове значення для кластеризації, що можна зробити, якщо для зібраних логітів попарно обчислити значення розходження Кульбака-Лейблера і порівняти його з відстанню між відповідними активаціями. Далі границю можна визначити відповідно до очікуваного розходження або адаптивно, за його зниженням. Приклад визначення границі наведено на рисунку 3.10.

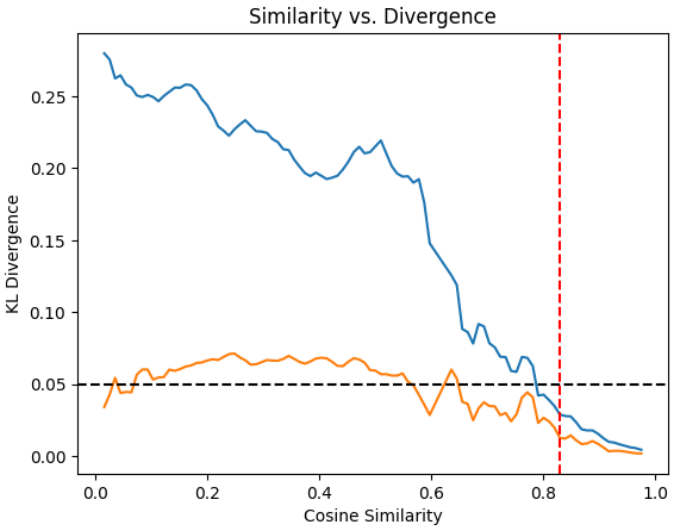


Рисунок 3.10 – Графік залежності розходження передбачень моделі від відстані між прихованими станами моделі

3.3.2 Результати експериментів

Експерименти з версіями моделей, що доступні як «біла скриня» проводились з використанням GPU Nvidia T4 і типом даних вагів float16. Результати наведено у таблиці 3.4.

Таблиця 3.4 – Результати Pass@8 методів для моделей типу «біла скриня»

Модель	Методи			
	Best-of-N	Bruteforce	Search	CSP
Llama 3.2 1B	0.18	0.24	0.3	0.3
Llama 3.2 3B	0.45	0.4	0.54	0.54

Можна побачити, що методи на основі ентропії мають явну перевагу над семплінгом, але при невеликих значеннях N гірші за методи для моделей типу «чорна скриня». При цьому варто зважати на те, що методи на основі ентропії є фактично найдешевшими, можуть виконуватись паралельно та працювати разом із методами на основі інженерії контексту. До того ж, максимум, якого можуть досягти такі методи теж вищий, адже ці методи ретельно досліджують простір станів. Це можна побачити порівнявши динаміку Best-of-N та Entropy-CSP із методами для моделей типу «чорна скриня». Методи на основі семплінгу не мають значного початкового приросту, але більш стабільні в покращенні результату. Зокрема, метод Entropy-CSP на практиці демонструє дешевший спосіб масштабування моделі до точності у 48%, що близько до максимального Pass@4 результату у 49% методу SWE-Search.

Для доведення того, що метод є ефективним для вищих значень N було виконано окремий експеримент зі значенням N=32 і порівняно зі звичайним Best-of-N. Результати порівняння на рисунку 3.11.

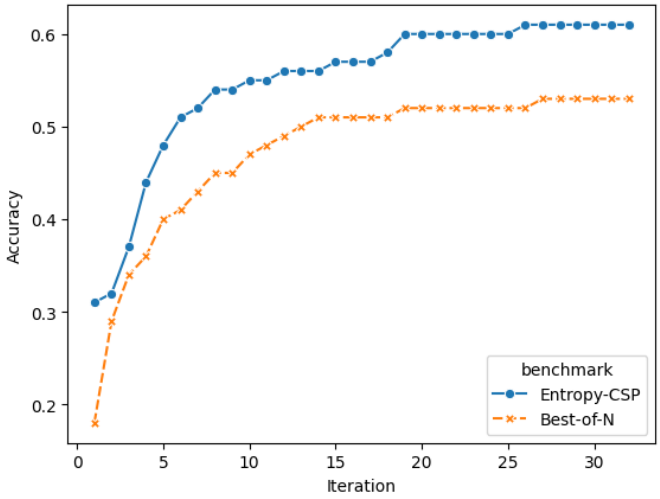


Рисунок 3.11 – Зміна точності з кількістю ітерацій при продовжуваному застосуванні методу Entropy-CSP на прикладі Llama-3.2-3B-Instruct

За рисунком 3.11 можна побачити, що за низьких значень N результати достатньо близькі, але з подальшим збільшенням кількості ітерацій розрив у точності двох методів зростає на користь Entropy-CSP. Множини вирішених завдань також зображено за допомогою діаграм Вена на рисунку 3.12.

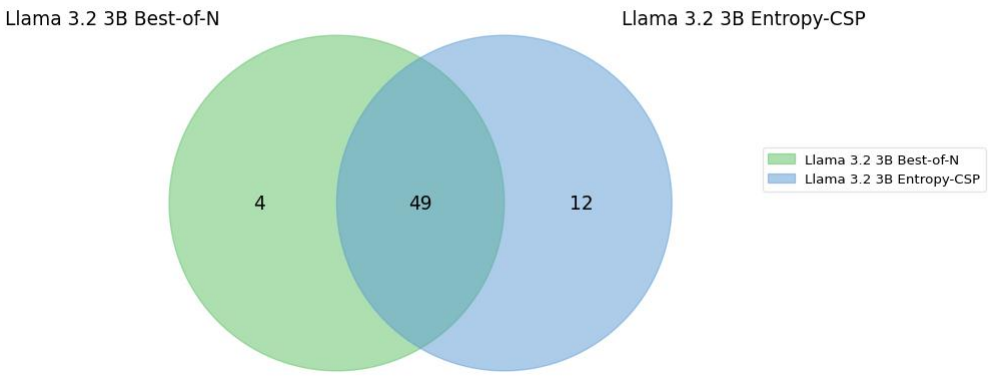


Рисунок 3.12 – Діаграма Вена для множин вирішених задач зі звичайним семплюванням і запропонованим варіантом

Якщо порівнювати множини вирішених завдань, можна побачити, що Entropy-CSP знайшов рішення не усіх задач, які модель змогла вирішити шляхом семплювання. За результатами першої ітерації Entropy-CSP (без масштабування взагалі) модель вирішує 33 задачі, що теж є одним з плюсів такого методу, адже при використанні семплювання температурої цей показник нижчий. Відповідно, використовуючи Entropy-CSP, агент розв’язав на 8 задач більше, ніж при застосуванні семплювання температурою, в тому числі 16 з 20 задач, які були розв’язані базовим методом.

ВИСНОВКИ

Аналіз поведінки мовних моделей в агентних задачах шляхом порівняння із процесом пошуку класичних агентів є дієвим підходом для визначення перспективних напрямків для покращення ефективності моделей в таких задачах. Зокрема, за результатами такого аналізу випливає обмежена ефективність більшості сучасних методів, зосереджених на застосуванні можливостей навчання в контексті та інших емерджентних здібностей великих мовних моделей.

Більшість сучасних підходів орієнтуються на те, що ефективність запропонованої стратегію масштабування зростатиме за рахунок більш вираженого застосування більшими моделями знайомих людині патернів мислення. Таке припущення також гарантує гнучкість цих підходів, адже вони не залежать від особливостей реалізації мовної моделі. Навпаки, використання нетипових і більш алгоритмічних підходів до роботи з мовними моделями може бути шляхом для збільшення ефективності вже існуючих моделей і забезпечення їх функціонального паритету із класичними агентами, але у набагато складніших середовищах. Причому можливі варіанти таких алгоритмів як для моделей типу «чорна скриня», так і для моделей типу «біла скриня». Серед таких підходів використання кількох генерацій замість довгих взаємодій, обов’язкова диверсифікація вхідних даних та даних, створюваних в ході генерації, спекулятивна рефлексія. Вони надають можливість більш контрольованого і повного перебору варіантів відповідей з використанням внутрішніх знань моделі про оптимальні напрямки і є ключовими для масштабування моделей через пошук.

На основі цих механізмів було запропоновано наступні методи масштабування: MCTS-Rewrite, LLM-CSP, Entropy CSP.

MCTS-Rewrite демонструє стабільно високу точність, а LLM-CSP найвищу середню оцінку незалежно від моделі без великого впливу на кількість токенів.

Методи на основі ентропії, зокрема Entropy-CSP, дозволяють ефективно поєднувати семплювання та жадібне декодування. Ці методи мають вищу ефективність при формуванні вибірки можливих рішень, ніж семплювання за температурою. При цьому процес визначення гіперпараметрів для таких методів можна автоматизувати, а в ході виконання формуються траєкторії, які встановлюють більш чіткі зв’язки між конкретними виборами і результатом виконання, що є важливим при використанні таких траєкторій для дотренування моделей.

Ці результати демонструють слушність та актуальність стратегії з використання додаткових обчислень з метою покращення відповідей вже натренованих моделей, що побудована на особливостях алгоритмів пошуку. Це також підтверджує на практиці теоретичні припущення на яких заснована відповідна стратегія, за якою при відокремленні поточної траєкторії та історії виконання агента можна користуватись механізмами диверсифікації для дослідження простору станів та механізмами атрибуції для експлуатації найкращих рішень. Запропоновані методи використовують додаткові обчислення ефективніше за існуючі аналоги, досягаючи цільових показників за меншу кількість ітерацій і використовуючи менше токенів. Відповідно, практична цінність отриманих методів – швидші та точніші агенти на основі мовних моделей в задачах програмування, аналізу даних, роботи з фінансами тощо.

Апробацію результатів дослідження було проведено на 30-му Міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXI столітті».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hybrid intelligence for industry 4.0: integrating large language models and reinforcement learning / V. Terziyan et al. *International Journal of Computer Information Systems and Industrial Management Applications.* 2025. Vol. 17. P. 610–627. URL: https://doi.org/10.70917/ijcisim-2025-0038 (date of access: 14.03.2026).

2. LLMs get lost in multi-turn conversation / P. Laban et al. 2025. URL: https://doi.org/10.48550/arXiv.2505.06120 (date of access: 24.03.2026).

3. Blank I. A. What are large language models supposed to model? *Trends in Cognitive Sciences.* 2023. URL: https://doi.org/10.1016/j.tics.2023.08.006 (date of access: 24.03.2026).

4. Attention is all you need / A. Vaswani et al. *arXiv.* 2017. URL: https://arxiv.org/abs/1706.03762 (date of access: 24.03.2026).

5. Training compute-optimal large language models / J. Hoffmann et al. *arXiv.* 2022. URL: https://arxiv.org/abs/2203.15556 (date of access: 24.03.2026).

6. The assistant axis: situating and stabilizing the default persona of language models / C. Lu et al. *arXiv.* 2026. URL: https://arxiv.org/abs/2601.10387 (date of access: 24.03.2026).

7. Terziyan V., Vitko O. Context-aware machine learning for smart manufacturing. *Procedia Computer Science.* 2025. Vol. 253. P. 25–36. URL: https://doi.org/10.1016/j.procs.2025.01.066 (date of access: 29.04.2026).

8. Efficient beam search for large language models using trie-based decoding / B. J. Chan et al. *arXiv.* 2025. URL: https://arxiv.org/abs/2502.00085v2 (date of access: 24.03.2026).

9. Large language monkeys: scaling inference compute with repeated sampling / B. Brown et al. *arXiv.* 2024. URL: https://arxiv.org/abs/2407.21787 (date of access: 24.03.2026).

10. Self-consistency improves chain of thought reasoning in language models / X. Wang et al. *arXiv*. 2022. URL: https://arxiv.org/abs/2203.11171 (date of access: 24.03.2026).

11. DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning / D. Guo et al. *arXiv*. 2025. URL: https://arxiv.org/abs/2501.12948 (date of access: 13.04.2026).

12. Lepori M. A., Mozer M. C., Ghandeharioun A. Racing thoughts: explaining contextualization errors in large language models. *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2025. Vol. 1. URL: https://aclanthology.org/2025.naacl-long.155/ (date of access: 24.03.2026).

13. Artificial Intelligence: A Modern Approach / S. J. Russell et al. Pearson Education, Limited, 2005.

14. Prosser P. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*. 1993. Vol. 9, no. 3. P. 268–299. URL: https://doi.org/10.1111/j.1467-8640.1993.tb00310.x (date of access: 11.03.2026).

15. Hart P., Nilsson N., Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968. Vol. 4, no. 2. P. 100–107. URL: https://doi.org/10.1109/tssc.1968.300136 (date of access: 11.03.2026).

16. Self-refine: iterative refinement with self-feedback / A. Madaan et al. *arXiv*. 2023. URL: https://arxiv.org/abs/2303.17651 (date of access: 24.03.2026).

17. Reflexion: language agents with verbal reinforcement learning / N. Shinn et al. *arXiv*. 2023. URL: https://arxiv.org/abs/2303.11366 (date of access: 24.03.2026).

18. Language agent tree search unifies reasoning acting and planning in language models / A. Zhou et al. *arXiv*. 2023. URL: https://arxiv.org/abs/2310.04406 (date of access: 24.03.2026).

19. Silver D. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv*. 2017. URL: https://doi.org/10.48550/arXiv.1712.01815 (date of access: 16.12.2025).

20. SWE-search: enhancing software agents with Monte Carlo tree search and iterative refinement / A. Antoniades et al. *arXiv*. 2024. URL: https://arxiv.org/abs/2410.20285 (date of access: 24.03.2026).

21. Terziyan V., Vitko O., Terziyan O. Beyond backpropagation: smarter neural networks for smart manufacturing. *Procedia Computer Science*. 2026. Vol. 277. P. 1651–1662. URL: https://doi.org/10.1016/j.procs.2026.02.203 (date of access: 29.04.2026).

22. The lessons of developing process reward models in mathematical reasoning / Zhenru Zhang et al. *arXiv*. 2025. URL: https://arxiv.org/abs/2501.07301 (date of access: 24.03.2026).

23. Steering language models with activation engineering / Alexander Matt Turner et al. *arXiv*. 2023. URL: https://arxiv.org/abs/2308.10248 (date of access: 24.03.2026).

24. Patchscopes: a unifying framework for inspecting hidden representations of language models / A. Ghandeharioun et al. *arXiv*. 2024. URL: https://arxiv.org/abs/2401.06102 (date of access: 24.03.2026).

25. Function vectors in large language models / Eric Todd et al. *The Twelfth International Conference on Learning Representations. arXiv*. 2024. URL: https://arxiv.org/pdf/2310.15213 (date of access: 24.03.2026).

26. Feucht S., Wallace B., Bau D. Vector arithmetic in concept and token subspaces. *Second Mechanistic Interpretability Workshop at NeurIPS. arXiv*. 2025. URL: https://arxiv.org/pdf/2511.18162 (date of access: 24.03.2026).

27. Interpreting GPT: the logit lens. *LessWrong*. URL: https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens (date of access: 13.04.2026).

28. SWE-bench: can language models resolve real-world GitHub Issues? / C. E. Jimenez et al. *arXiv*. 2023. URL: https://arxiv.org/abs/2310.06770 (date of access: 25.03.2026).

29. The Llama 3 herd of models / A. Grattafiori et al. *arXiv*. 2024. URL: https://arxiv.org/abs/2407.21783 (date of access: 13.04.2026).

30. Gemma 3 technical report / A. Kamath et al. *arXiv*. 2025. URL: https://arxiv.org/abs/2503.19786 (date of access: 13.04.2026).