УДК 004.9 **doi: 10.26906/SUNZ.2024.4.109**

О. Мошура[1], Т. Деркач[1], А. Дмитренко[1], Л. Клочко[2], В. Лоза[1]

[1] Національний університет «Полтавська політехніка імені Юрія Кондратюка», Полтава, Україна
[2] Федеральна політехнічна школа, Лозанна, Швейцарія

ОСОБЛИВОСТІ ТА МОЖЛИВОСТІ ЗАСТОСУВАННЯ LLM У СФЕРІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. У даній статті досліджено ролі та можливості Великих Мовних Моделей (LMM) у сфері розробки програмного забезпечення, які варіюються від спеціалізованих, орієнтованих на конкретні мови або домени, до загальних моделей, які можуть застосовуватися до широкого спектру задач. Проведено огляд основних особливостей LMM, акцент на їхньому потенціалі у глибинному аналізі та генерації тексту, а також розкриває можливості застосування в широкому спектрі задач. Проаналізовано основні напрями щодо оптимізації роботи з LMM, які включають в себе контекст, fine tuning, векторизація інформації, використання вбудованого інструментарію платформ, prompt engineering, zero-shot prompting, few-shot prompting, chain-of-thought prompting, directional stimulus prompting, промпт з використанням dsp, промпт без використання dsp, tree of thought prompting, reward prompting, Developer driven LLM, Repository driven LLM, Project driven LLM. Детально аналізуються переваги та недоліки комерційних та відкритих (OpenSource) моделей. Представлено стратегії використання LLM моделей для розробників, як особистий досвід роботи з LLM та ідеї які ще доведеться реалізовувати. Особливу увагу приділено концепціям, які орієнтовані на розробника, які надають підтримку та відповіді на основі великих даних та контексту проекту та включають в собі певні звички в кодуванні, вподобання до певних технологій чи бібліотек, і навіть специфічні доменні знання, які розробник застосовує у своїй рутинній роботі. Визначено необхідність враховувати, хто буде використовувати модель: лише розробники чи також замовники та користувачі, в залежності від аудиторії, адаптування рівеня доступу до інформації. Сформовані ймовірні проблеми роботи з LLM з якими можуть зіткнутися розробники при роботі з великими мовними моделями, які можуть генерувати нерелевантні відповіді, містити помилкову інформацію чи створювати її (галюцинації), а також мати упередження та затримку у врахуванні останніх подій. Визначено людський фактор, як ключовий фактор в оцінці і користуванні результатами роботи мовних моделей, перед її впровадженням у проект. Робота призначена для інформування розробників про стратегії вибору та адаптації моделей LMM для специфічних вимог проектів з урахуванням їхнього контексту.

Ключові слова: Великі Мовні Моделі, штучний інтелект, fine-tuning, векторизація, OpenSource.

Вступ

У сучасному світі технологій та швидкоплинної інформації, розробники постійно шукають шляхи оптимізації своїх робочих процесів.

Інтеграція мовних моделей може стати революційним кроком для оптимізації процесу розробки, надаючи додаткові ресурси для аналізу, творчості та інновацій, завдяки використанню штучного інтелекту. Використання таких потужних інструментів розробниками змінює вектор розвитку програмування.

Мовні моделі вносять значний вклад у пошук швидких та ефективних рішень для складних завдань, дозволяючи розробникам зосередитися на більш критичних та творчих аспектах своїх проектів.

Метою даної роботи є дослідження ролі та можливостей Великих Мовних Моделей (LMM) у сфері розробки програмного забезпечення.

Основна частина

LLM (Large Language Model, Велика Мовна Модель) – тип штучного інтелекту, який може аналізувати та генерувати текст. Ці моделі тренуються на великих масивах інформації (звідси й назва "Велика").

Простіше кажучи, ВММ – це комп'ютерна програма, яка отримала достатньо прикладів, щоб розпізнавати та інтерпретувати людську мову на глибинному рівні.

Такі моделі використовуються для широкого спектру завдань, від автоматичного генерування тексту до аналізу емоційного забарвлення (сентимент-аналізу), відповідей на запитання, автоматичного перекладу та багато іншого.

Особливістю LLM є їх здатність не просто працювати з текстом на поверхневому рівні, але й розуміти нюанси та контекст мови, що робить їх надзвичайно потужним інструментом в галузі штучного інтелекту.

Моделі. У більшості випадків, коли говорять про LLM у популярних медіа, акцент робиться на великих комерційних моделях, таких як ChatGPT від OpenAI, Gemini (також відомий як Bard) від Google, або Claude від Anthropic.

Ці моделі є дуже відомими прикладами використання LLM для створення потужних інструментів генерації тексту, діалогових систем та інших додатків.

Проте, у більшості ці моделі закриті, а застосунки можуть читати повідомлення з переписок користувачів, що не є рекомендовано для NDA проектів.

Наразі існує велика кількість моделей LLM різної спрямованості, багато з яких є відкритими (OpenSource) і доступними для використання та модифікації спільнотою.

Ці моделі варіюються від спеціалізованих, орієнтованих на конкретні мови або домени, до загальних моделей, які можуть застосовуватися до широкого спектру задач.

Більшість відкритих моделей можна переглянути на сайті huggingface.co, там публікують свої моделі як і більшість провідних IT компаній, так і стартап команди, чи навіть звичайні ентузіасти. А великим плюсом цієї платформи є можливість підняття будь якої моделі в приватному репозиторії що дозволяє зекономити час і кошти на тестування.

У сучасних реаліях розробники мають обирати ШІ асистента так само ретельно, як і колір інтерфейсу у своїй IDE.

Контекст. Розширення контексту для мовних моделей є ключовим фактором для отримання більш релевантних та точних відповідей.

Нижче представлені декілька підходів, які можуть допомогти розробникам покращити взаємодію з LLM.

Fine tuning. Fine-tuning дозволяє адаптувати загальну модель під конкретні потреби проекту. Це досягається шляхом додаткового навчання моделі на меншому, специфічному наборі даних, який відображає особливості завдань, з якими модель буде працювати.

Переваги: збільшення точності відповідей у специфічних доменах, краще розуміння контексту завдання.

Недоліки: потребує додаткових даних для тренування, може зайняти додатковий час та ресурси.

Векторизація інформації. Векторизація інформації передбачає перетворення текстових даних у вектори чисел, які мовна модель може ефективніше обробляти. Це може допомогти моделі краще розуміти відносини між словами та концепціями у запиті.

Переваги: покращене розуміння нюансів мови, здатність моделі виявляти зв'язки між різними елементами контексту.

Недоліки: потребує розуміння процесів векторизації та оптимізації моделі під векторизовані дані.

Використання вбудованого інструментарію платформ. Багато платформ, що надають доступ до LLM, також пропонують інструменти та API для роботи з контекстом.

Ці інструменти можуть допомогти розробникам керувати контекстом запитів та відповідей більш ефективно.

Переваги: зручність використання, оптимізовані під конкретну платформу рішення, які можуть полегшити роботу з контекстом.

Недоліки: залежність від можливостей та обмежень конкретної платформи, потенційна необхідність адаптації до специфіки інструментарію.

Prompt engineering. Для того щоб отримати бажану відповідь, треба правильно прописати промпт (запит) до такої моделі. Зазвичай це питання: "Чому Python 2 вже не підтримується?" або інструкція: “Напиши рецепт для вечері, якщо у мене є кіло картоплі”.

Більше інформації про техніку промптингу можна отримати на ресурсі https://www.promptingguide.ai/techniques [1]. Всі відповіді на промпти були згенеровані через модель gpt-4 від OpenAI

Поради.

1. Починайте з простого. При продумуванні промпту важливо розуміти що це творчий процес, який вимагає певного часу для отримання оптимальних результатів. Важливо послідовно покращувати свій промпт із плином спроб додаючи певні деталі або відрізаючи непотрібний контекст

2. Уникайте неточностей. Завжди вказуйте що вам конкретно треба. Якщо ви очікуєте на якийсь результат то варто його вказати в промпті. "Напиши мені два речення про LLM для розробників" виглядає набагато краще ніж "Розкажи коротко що таке LLM".

3. Вказуйте контекст. Уявіть що ви розмовляєте з колегою щодо певного нового функціоналу в коді. Питання "Чому функція get_weather() не повертає інформацію?" викличе тільки ще більше питань у вашого колеги.

Так і з мовними моделями.

Звідки вони знають що це за метод. Додайте додатковий контекст у вигляді шматку коду, очікуваного\реального результату, трейсбек, і т.д.

Наприклад:

1. "Дай мені поради щодо програмування." – можна замінити на: "Дай мені п'ять порад для початківців у Python."

2. "Чому мій код не працює? [помилковий код]" – варто замінити на: "Мій Python код видає помилку 'IndexError: list index out of range' в цьому фрагменті кода: [помилковий код]. Як це виправити?"

3. "Розкажи мені щось цікаве про ШІ в IT." – можна покращити через: "Які основні зміни в індустрії IT принесла штучний інтелект за останні 5 років?"

4. "Покращи код: [код]." – варто замінити на: "Як можна оптимізувати цей фрагмент коду на Python для кращої продуктивності при роботі з великими списками? [код]"

5. "Як зробити щось круте у програмуванні?" – можна замінити на: "Як створити веб-додаток із використанням React та Python для управління проектами?"

Zero-shot prompting. Простими словами промпт без прикладів. Ідеально для питань які не вимагають від вас додаткової інформації, наприклад:

Ви: "Чи кавун це фрукт?"

LLM: "Ні, це ягода."

Ви: "Як зробити HTTP GET запит в Python?"

LLM: "Використовуйте вбудовану бібліотеку urllib, або requ…"

Ви: "Що треба для відправки звітності по ФОПу? "

LLM: "Для відправки звітності по ФОП (фізичній особі-підприємцю) в Україні…"

LLM на то і є "Великими Мовними Моделями" щоб мати змогу відповісти на загальні питання

Few-shot prompting. Це промтинг з додаванням прикладів. Спосіб також зосереджений на нескладних питаннях чи інструкіях, бажано не використовувати його для складних тем.

Chain-of-Thought prompting. Ланцюжок думок – це ідеальний тип промптів коли ви отримуєте неправильний результат на складне питання. Його використання досить елегантне, варто дописати до свого промпту: "Думай послідовно" чи "Розпиши крок за кроком"

Ви: "Коли мені було 6 років, моя сестра була вдвічі молодша. Зараз мені 70 років, скільки років моїй сестрі?"

LLM: "Сестрі 35 років. "

Так як мовні моделі не виконують математичні операції вони схильні відповідати закономірно. Тому спробуємо такий підхід

Ви: "Коли мені було 6 років, моя сестра була вдвічі молодша. Зараз мені 70 років, скільки років моїй сестрі? Розпиши крок за кроком".

LLM: "Коли вам було 6 років, ваша сестра була удвічі молодша, тобто їй було 6/2 = 3 роки. Різниця у віці між вами та вашою сестрою завжди буде однаковою, тобто 6 - 3 = 3 роки. Таким чином, вашій сестрі зараз 70 - 3 = 67 років"

Також цей тип промптингу можна розширити за допомогою Few-shot prompting задля отримання найбільш релевантного результату

Directional Stimulus Prompting. Спрямоване стимулювання – це промптинг у якому ви додаєте підказки або спрямовуєте на якусь відповідь. Наприклад при генерації підсумку зі статті можна вказати "підказки" того що ви хочете бачити в результаті.

Наприклад, контекст “Сонячна енергія - це чисте та відновлюване джерело енергії, яке отримують за допомогою сонячних панелей, що перетворюють сонячне світло на електрику. Цей метод виробництва електроенергії не лише допомагає знизити вуглецевий слід, але й може забезпечувати значні економічні переваги, зокрема зниження рахунків за електроенергію.

Незважаючи на початкові високі витрати на установку, сонячні панелі часто окупаються протягом декількох років, пропонуючи довгострокове рішення для виробництва енергії."

Промпт з використанням DSP:

Ви:"Підкресліть екологічні та економічні переваги використання сонячної енергії на основі наведеного тексту."

LLM: Сонячна енергія знижує вуглецеві викиди та забезпечує економію на рахунках за електроенергію, швидко окупаючи початкові витрати на установку.

Промпт без використання DSP:

Ви:"Узагальніть текст про сонячну енергію."

LLM: Сонячна енергія перетворює світло на електрику, забезпечуючи екологічні та економічні переваги, незважаючи на високу початкову вартість.

Tree of Thought Prompting. Дерево думок – це тип промптингу для складних задач, де прості методи вже не допомагають вирішити проблему, або обгрунтованості їх відповідей недостатньо. При використанні цього підходу треба визначити кількість кандидатів та кроків після яких буде отримано відповідь.

Reward Prompting. Цей підхід промптингу, включає систему винагород або стимулу для моделі, для того щоб вона могла самостійно оцінювати якість своїх відповідей.

Також він ефективний в ітеративному процесі – мовна модель вже знає на основі попередніх відповідей що вам треба, а що ні.

Типи використання LLM моделей для розробників. Стратегії використання описані в дослідженні – це особистий досвід роботи з LLM та ідеї які ще доведеться реалізовувати.

Тут згадано механіки які можна застосовувати як індивідуальному розробнику так і командам або навіть компаніям для пришвидшення бізнес процесів.

Developer driven LLM. Концепція мовної моделі, яка орієнтована на розробника.

Для того щоб отримувати релевантні відповіді від мовних моделей, уникаючи базових відповідей "А чи ви спробували перезапустити ПК" розробник може “розказати” трохи про себе, адаптувавши модель до його персонального досвіду, стилю роботи, або конкретних потреб у проектах.

Це може включати в себе певні звички в кодуванні, вподобання до певних технологій чи бібліотек, і навіть специфічні доменні знання, які розробник застосовує у своїй рутинній роботі.

Елементи контексту:

1. CV. Це встановить рамки стеку для LLM і дасть розуміння того, що розробник знає, а про що варто розказати. Якщо CV немає то в контекст можна додати таку інформацію як:

a. спеціалізація та напрямки роботи. Наприклад Backend Development чи Data Science чи Automation QA;

b. мови програмування які розробник знає. Варто виділити основні, якщо їх декілька;

c. технології. Наприклад Postgres, Docker, Redis, GraphQL і т.д.;

d. інструментарій та сервіси (Git, Jenkins, Github, Gitlab, Heroku і т.д.);

e. фреймворки та бібліотеки з якими розробник працює.

2. Слабкі сторони та нові знання. Зазначення областей, де розробник відчуває нестачу досвіду або тільки починає вчитися.

3. Проекти та портфоліо. Детальний опис розроблених проектів, включаючи використані технології, вирішені проблеми, роль розробника у проекті та досягнуті результати. Це допоможе моделі краще розуміти досвід розробника та його спеціалізацію.

4. Освітній фон. Інформація про формальну освіту та самоосвіту, включаючи курси, сертифікати та спеціалізовані тренінги. Це надасть моделі уявлення про теоретичні знання та вміння розробника

5. Звички при розробці. Розуміння особливостей робочого процесу розробника, таких як підходи до тестування, використання систем контролю версій, стиль кодування, може допомогти моделі надавати поради, що відповідають цим звичкам.

Repository driven LLM. Це концепція мовної моделі, яка аналізує інформацію проекту в репозиторіях, включно з кодом, змінами, запитами на злиття, проблемами та документацією.

Це дозволяє моделі надавати точніші відповіді та рекомендації, враховуючи контекст розробки.

Щоб покращити аналіз та зменшення кількості надлишкової інформації, модель може використовувати стратегії:

1. Оптимізація коду. Виносить оголошення класів, методів і констант у окремий файл, векторизує їх для швидкого пошуку, потім виконує кроки:

a. Пошук декларацій за запитом.

b. Додавання коду відповідних декларацій до контекста запитання.

2. Сумаризація обговорень. Замість повного тексту обговорень, модель створює їхній короткий зміст з ключовою інформацією, потім:

a. Пошукує заголовки та документацію.

b. Додає аналізований зміст до контекста запитання.

3. Кешування даних. Зберігання часто запитуваних даних у кеші може значно прискорити доступ до них і зменшити час відповіді моделі. Це особливо корисно для повторюваних запитів

4. Семантичне групування. Застосування методів машинного навчання для семантичного групування схожих запитів, іш'ю, пул-ріквестів та документації дозволяє виявляти загальні теми або проблеми, а також може пришвидшити відповідь від мовної моделі

І вже на основі застосованих стратегій така мовна модель буде готова для відповіді на питання. Її можна застосовувати:

1. Новим розробникам. Для входження в новий проект і для оптимізації часу отримання відповіді замість обговорення з іншими розробниками.

2. Для швидкого відновлення у пам’яті старого коду.

3. Для продумування стратегій рефакторингу.

4. Для пошуку вразливостей у коді.

Project driven LLM. Концепція "Project Driven LLM" втілює в собі глибоке інтегрування в проектні процеси, аналізуючи та адаптуючи інформацію не лише з репозиторіїв, але й з обговорень, зворотного зв'язку від користувачів та замовників. Це дозволяє створити багатошаровий контекст, збагачений різноманітними джерелами даних, які відображають реальну картину проекту.

В собі така мовна модель повинна містити усі пов’язані репозиторії проекту (наприклад, мікросервісна архітектура, чи окремі репозиторії для Android, IOS, WEB) а також в неї варто додати:

1. Сумаризація обговорень розробників. Простими словами можна сумаризувати чат розробників (подобово, потижнево і т.д.) відфільтрувавши лише технічні дискусії для швидкого пошуку обговорень

2. Сумаризація обговорень з клієнтами та замовниками. Зведення до мінімуму комунікації з замовниками та клієнтами дозволяє виявити основні вимоги та очікування, забезпечуючи точне розуміння проектних цілей. Цей процес гарантує, що фінальний продукт максимально відповідає потребам користувачів.

3. Сумаризація фідбеку від користувачів. Аналіз відгуків користувачів є невід'ємною частиною процесу розробки, оскільки він надає цінне бачення того, як продукт використовується в реальному житті та як його можна покращити.

4. Прогнозування потреб проекту

5. Інтеграція машинного навчання для аналізу існуючих даних проекту може допомогти в автоматизації прогнозування майбутніх потреб розробки, зокрема щодо ресурсів, часу на реалізацію та потенційних ризиків.

6. Визначення пріоритетів розробки. Оцінка критичності задач і функціоналу дозволяє оптимізувати порядок реалізації проектних робіт, зосереджуючись на найбільш значущих аспектах для досягнення стратегічних цілей.

При цьому, важливо враховувати, хто буде використовувати модель: лише розробники чи також замовники та користувачі. В залежності від аудиторії, необхідно адаптувати рівень доступу до інформації

Ймовірні проблеми роботи з LLM.

Мовні моделі можуть бути потужним інструментом для розробників, надаючи підтримку та відповіді на основі великих даних та контексту проекту. Однак, не всі результати, які вони генерують, завжди ідеально відповідають потребам користувачів.

Нижче наведені деякі можливі проблеми, з якими можуть зіткнутися розробники при роботі з великими мовними моделями:

1. Нерелевантні відповіді. Незважаючи на великий обсяг знань, великі мовні моделі можуть генерувати відповіді, які не відносяться до суті поставленого запитання. Це може бути результатом неправильного інтерпретування запиту або недоліків структури знань моделі.

2. Помилкова інформація. Мовні моделі покладаються на навчальні дані для генерування відповідей, і якщо ці дані містять помилкову інформацію, LLM може реплікувати ці помилки у своїх відповідях.

3. Галюцинації. Іноді мовні моделі можуть "галюцинувати" відповіді – творити факти або дані, які не мають підстав в дійсності. Це особливо проблематично в технічних або наукових контекстах, де точність критично важлива.

4. Затримка у врахуванні останніх подій. Великі мовні моделі часто відстають щодо включення останніх новин або трендів у свою базу знань через цикли оновлення. Це може призводити до того, що поради або аналізи здійснюються на основі застарілої інформації.

5. Упередження і стереотипи. Кожна мовна модель несе у собі приховані упередження, які беруться з даних, на яких вона навчалася. Це може включати гендерні, расові чи культурні упередження, що може відображатись у відповідях мо-

делі та спотворювати загальний контекст спілкування з розробниками.

Висновки

Враховуючи вищезазначене, можна констатувати, що:

1. Великі Мовні моделі (LLM) є незамінним інструментом у арсеналі розробника, подібно до того як молоток є ключовим інструментом для будівельника. Вони підвищують продуктивність та ефективність, забезпечуючи підтримку в аналізі великих обсягів данних.

2. Для отримання точніших відповідей від мовної моделі, необхідно надавати їй багатий та релевантний контекст, але при цьому додавати лише інформацію, що безпосередньо пов'язана з поставленим запитанням.

3. Взаємодія з мовними моделями часто є ітеративним процесом, що вимагає пошуку правильних промптів та коригування запитів для отримання необхідної інформації.

4. Розробка спеціалізованих мовних моделей для конкретних проектів може суттєво скоротити час розробки та прискорити пошук необхідної інформації, оскільки такі моделі оптимізовані для специфічних вимог проекту.

5. Незважаючи на їх користь та зручність, мовні моделі не є бездоганними. Вони можуть генерувати нерелевантні відповіді, містити помилкову інформацію чи створювати її (галюцинації), а також мати упередження. Виходячи з цього, фінальне рішення має прийматися людиною на основі критичного аналізу відповідей моделі.

6. Людський фактор залишається ключовим в оцінці і користуванні результатами роботи мовних моделей, і розробникам слід ретельно перевіряти інформацію, отриману від LLM, перед її впровадженням у проект.

Список літератури

1. (2024), “Prompting Techniques (Техніки промптингу)”, *Prompt Engineering Guide*, URL: https://www.promptingguide.ai/techniques/
2. Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le (2022),”Finetuned language models are zero-shot learners”, *Published as a conference paper at ICLR 2022*, pp. 1–46, URL: https://arxiv.org/pdf/2109.01652.pdf
3. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei (2020), *Language Models are Few-Shot Learners*, aXiv:2005.14165, 75 p., URL: https://arxiv.org/abs/2005.14165
4. Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, Denny Zhou (2022), *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*,14 p., URL: https://arxiv.org/abs/2201.11903
5. Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, Xifeng Yan (2023), *Guiding Large Language Models via Directional Stimulus Prompting*, 27 p., URL: https://arxiv.org/abs/2302.11520
6. Shunyu Yao, Dian Yu, Jeffrey Zhao, (2023), Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 14 p., URL: https://arxiv.org/abs/2305.10601

Received (Надійшла) 20.09.2024
Accepted for publication (Прийнята до друку) 06.11.2024

Features and capabilities of LLM application in software development

O. Moshura, T. Derkach, T. Dmytrenko, A. Dmytrenko, V. Loza

Abstract. This article explores the roles and capabilities of Large Language Models (LLMs) in software development, ranging from specialized models targeted at specific languages or domains to general models applicable to a wide array of tasks. It provides an overview of the key features of LLMs, emphasizing their potential for deep analysis and text generation, and reveals opportunities for their application across various challenges. The article analyzes primary directions for optimizing interactions with LLMs, which include context management, fine-tuning, information vectorization, leveraging built-in tools from platforms, prompt engineering, zero-shot prompting, few-shot prompting, chain-of-thought prompting, directional stimulus prompting, prompts using domain-specific prompts (dsp), prompts without using dsp, tree of thought prompting, reward prompting, Developer-driven LLMs, Repository-driven LLMs, and Project-driven LLMs. It details the advantages and disadvantages of both commercial and open-source models. Strategies for utilizing LLMs by developers are presented, drawing on personal experiences with LLMs and ideas yet to be realized. Special attention is given to developer-oriented concepts that provide support and responses based on large datasets and project context, including coding habits, preferences for specific technologies or libraries, and even specific domain knowledge that developers apply in their routine work. The necessity of considering who will use the model−whether developers, clients, or end-users−is emphasized, highlighting the importance of adapting the level of access to information based on the audience. Potential issues that developers may encounter when working with LLMs are identified, including the generation of irrelevant responses, misinformation or "hallucinations," as well as biases and delays in accounting for recent events. The human factor is identified as a critical element in evaluating and utilizing the outputs of language models before their implementation in a project. This work aims to inform developers about strategies for selecting and adapting LLMs to meet the specific requirements of projects while taking their context into account.

Keywords: Large Language Models, artificial intelligence, fine-tuning, vectorization, OpenSource.