### Конспект лекції 4: Узгодження (Alignment) Великих Мовних Моделей для Коду

#### 1. Вступ: Що таке узгодження і чому воно критично важливе?

У життєвому циклі великих мовних моделей (LLM) процес **узгодження (alignment)** є ключовим етапом, що відбувається *після* попереднього навчання (pre-training). Його мета — адаптувати модель із загальними знаннями до виконання конкретних завдань та відповідності людським інструкціям і вподобанням. Якщо попереднє навчання дає моделі фундаментальне розуміння мов програмування, то узгодження перетворює її на спеціалізованого, корисного та надійного помічника для розробника.Стратегічну важливість цього етапу неможливо переоцінити, оскільки він безпосередньо усуває критичні недоліки попередньо навчених моделей у професійних контекстах розробки програмного забезпечення. Загальні LLM, незважаючи на широту своїх знань, часто зазнають невдач у задачах, що вимагають глибинної спеціалізації. Вони можуть генерувати код, що поверхнево виглядає правильним, але не відповідає специфічним доменним обмеженням, таким як "тонкі контракти API, політики безпеки", і демонструють слабке "розуміння на рівні репозиторію". Саме узгодження визначає практичну цінність моделі та її здатність генерувати точний, безпечний і корисний код.Далі ми розглянемо перший і фундаментальний метод узгодження — кероване донавчання (Supervised Fine-Tuning).

#### 2. Кероване донавчання (Supervised Fine-Tuning, SFT): Навчання на Прикладах

Кероване донавчання (SFT) є першим і базовим кроком у процесі узгодження моделі. Його основна мета — навчити модель слідувати інструкціям і генерувати бажані відповіді. Це досягається шляхом донавчання на високоякісних наборах даних, що складаються з пар "інструкція-відповідь".Для завдань, пов'язаних з кодом, SFT має свої унікальні аспекти та сучасні підходи:

* **Важливість зворотного зв'язку від виконання (Execution Feedback):** Ключова відмінність коду від природної мови полягає в тому, що його можна виконати. На відміну від завдань з природною мовою, де узгодження покладається на дорогий та суб'єктивний людський зворотний зв'язок, код отримує перевагу від детермінованого, дешевого та високоефективного автоматизованого циклу зворотного зв'язку через компілятори та інтерпретатори. Цей механізм дозволяє миттєво перевіряти та покращувати згенеровані рішення без участі людини.
* **Багатовиткове (Multi-Turn) донавчання:** Цей підхід використовує для навчання діалоги, в яких модель робить помилку, отримує зворотний зв'язок і виправляє її. Такий формат допомагає моделі вчитися на власних помилках. Щоб адаптувати цей підхід до практичних сценаріїв, де очікується відповідь за один раз, був запропонований метод **самодистиляції** (Ren et al. 844). Він дозволяє моделі навчатися на багатовиткових даних, але на етапі використання (інференсу) генерувати фінальний, виправлений код за один виток.
* **Використання мультиагентних систем:** Для створення якісних навчальних даних можуть використовуватися системи з кількох агентів. Наприклад, у підході **AIEV-Instruct** 528 один агент виступає в ролі "програміста" і генерує код, а інший — "запитувача". Якщо код містить помилки, "запитувач" ставить уточнюючі питання, що стимулює "програміста" виправити код. Це дозволяє автоматично генерувати якісні багатовиткові діалоги для навчання.Ефективність SFT безпосередньо залежить від якості навчальних даних. Тому наступним логічним кроком є розгляд процесу їх підготовки.

#### 3. Підготовка даних для SFT: Основа для якісного узгодження

Якість даних є наріжним каменем успішного керованого донавчання. Мета цього етапу — створити надійний, чистий та різноманітний набір даних, який дозволить моделі узагальнювати знання, а не просто запам'ятовувати приклади (перенавчатися). Процес підготовки даних можна розділити на кілька ключових етапів.

* **Джерела даних (Data Sourcing)** На цьому етапі збираються початкові сирі дані. Джерела можуть бути різноманітними: від онлайн-форумів (наприклад, Math StackExchange) до наборів даних зі змагань та синтетично згенерованих завдань (наприклад, Open-CodeReasoning 12). Дослідження (OpenThoughts3 354) показують, що якість джерел є важливішою за їхню кількість; краще зосередитись на кількох надійних джерелах, ніж на великій кількості різноманітних, але низькоякісних.
* **Очищення та деконтамінація даних (Data Cleaning and Decontamination)** Мета цього процесу — підвищити цілісність даних та запобігти "витоку" тестових прикладів у навчальну вибірку. Для цього використовуються такі методи:
* **Семантичне порівняння:** Використання потужної LLM в ролі "судді" (LLM-Judge) для виявлення не лише точних дублікатів, а й перефразованих завдань (як у DeepMath-103K 395).
* **Зіставлення n-грам:** Пошук послідовностей слів, що збігаються, для виявлення схожих завдань (як у OpenMathReasoning 707).
* **Фільтрація та вдосконалення рішень (Solution Filtering and Refinement)** Важливо мати не просто правильні, а й високоякісні рішення. Цей етап спрямований на створення ідеальних "когнітивних шаблонів", на яких модель буде вчитися міркувати логічно та ефективно. Приклади методологій:
* **Оцінка на основі правил:** Використання системи балів для відбору найкращих ланцюжків міркувань за такими мета-ознаками, як **розширене обґрунтування (elaborated reasoning), самоперевірка (self-verification), дослідницький підхід (an exploratory approach) та адаптивна гранулярність (adaptive granularity)** (LIMO 1216).
* **Фільтрація за значущістю:** Відбір лише тих рішень, де використання інструментів (наприклад, виконання коду) є нетривіальним і додає реальну цінність (OpenMathReasoning 707).
* **Перевірка узгодженості відповідей:** Для одного завдання генерується кілька рішень, і зберігаються лише ті, де кінцеві відповіді збігаються. Це гарантує надійність даних (DeepMath-103K 395).Після того, як модель пройшла етап SFT, наступним кроком для подальшого вдосконалення її поведінки є навчання з підкріпленням.

#### 4. Узгодження через Навчання з Підкріпленням (RL)

Навчання з підкріпленням (RL) є потужним методом для подальшого узгодження моделей після SFT. На відміну від SFT, де модель вчиться імітувати надані приклади, RL дозволяє оптимізувати її поведінку безпосередньо, керуючись сигналами винагороди (rewards), таким як успішне проходження юніт-тестів.Ключовим підходом у цій галузі є **Навчання з підкріпленням на основі верифікованих винагород (Reinforcement Learning with Verifiable Rewards, RLVR)** . Суть методу полягає в наступному: модель генерує кілька кандидатських рішень для задачі, які потім автоматично перевіряються (наприклад, за допомогою юніт-тестів). Модель отримує позитивну винагороду і донавчається лише на тих варіантах, які пройшли перевірку.Однак, важливо розуміти обмеження цього методу. RLVR фундаментально обмежене тими шаблонами міркувань, що були закладені на початкових етапах навчання. Воно передусім відточує наявні можливості моделі, концентруючи ймовірнісну масу на відомих шляхах міркувань з високою винагородою, а не відкриває абсолютно нові. Таким чином, RLVR оптимізує та вдосконалює існуючі навички, а не створює їх з нуля.Для ефективного застосування RLVR були розроблені спеціалізовані набори даних:

* **ACECode-87K:** Синтетичний набір даних, створений за допомогою GPT-4o-mini, де для кожної задачі згенеровано 16 різноманітних тестів. Це забезпечує надійні та чіткі бінарні сигнали винагороди (пройшов/не пройшов), які ідеально підходять для RL.
* **KodCode:** Найбільший на сьогодні (представлений у 2025 році) повністю синтетичний набір даних, що містить різноманітні тематичні підмножини — від алгоритмічних головоломок до доменно-специфічних завдань. Кожна задача супроводжується вичерпним набором юніт-тестів.Сучасні моделі виходять за рамки роботи лише з текстовими інструкціями. Наступний етап еволюції узгодження — це інтеграція візуальної інформації.

#### 5. Мультимодальне Узгодження: Інтеграція Коду та Візуальної Інформації

Мультимодальне узгодження розширює можливості LLM, дозволяючи їм розуміти та генерувати код на основі нетекстових вхідних даних, таких як зображення, діаграми, ескізи від руки або макети користувацьких інтерфейсів (UI).Основні напрямки та завдання в мультимодальному узгодженні для коду включають:

* **Генерація інтерфейсів (Frontend Development):** Навчання моделей перетворювати візуальні макети на готовий до використання frontend-код. Бенчмарки для цього завдання еволюціонували від простих, як-от Design2Code (перетворення готового дизайну на код), до складніших, що включають генерацію з ескізів (Sketch2Code) та створення динамічних, інтерактивних сторінок (Interaction2Code).
* **Генерація артефактів програмної інженерії (Software Engineering Artifact Generation):** Моделі вчаться створювати UML-діаграми, блок-схеми та архітектурні схеми на основі існуючого коду або його текстового опису, що допомагає у візуалізації архітектури та документуванні.
* **Еволюція метрик оцінювання:** З розвитком мультимодальних завдань змінювались і способи їх оцінки.

|  |  |
| --- | --- |
| ***Етап*** | ***Фокус метрик*** |
| **Традиційний** | Синтаксична відповідність (наприклад, BLEU) |
| **Структурний** | Ієрархічна структура коду (наприклад, TreeBLEU, DOM-ED) |
| **Мультимодальний** | Візуальна відповідність та схожість рендерингу (наприклад, Visual fidelity, Rendering similarity) |
| **Автоматизований** | Наскрізна функціональність та оцінка за допомогою MLLM |

* **Ключові технічні тренди:** Сучасні підходи стають все більш автономними. Серед них варто виділити:
* **Автоматичний зворотний зв'язок:** Системи, як-от UICoder 1086, використовують цикл "компіляція-рендерінг-перевірка" для автоматичного виявлення помилок у згенерованому коді.
* **Агентські підходи (Agentic Workflows):** Моделі, як-от Frontend Diffusion 255, використовують складніші підходи. Наприклад, пропонується трьохетапний агентський ланцюжок "Ескіз → PRD (документ з вимогами до продукту) → Код", який завершується циклом валідації "Код → Рендерінг → Самооцінка → Виправлення".Незважаючи на значний прогрес, залишається низка невирішених проблем, які визначатимуть майбутні напрямки досліджень.

#### 6. Висновок: Основні виклики та майбутні напрямки

Отже, ми розглянули ключові етапи узгодження великих мовних моделей для коду: від керованого донавчання та підготовки даних до навчання з підкріпленням та мультимодальної інтеграції. Ці процеси перетворюють LLM із загальних моделей на потужні інструменти для розробників.Один з найважливіших викликів — це **недостатня багатомовна підтримка** . Більшість навчальних наборів даних зосереджені на популярних мовах програмування (Python, JavaScript), що обмежує ефективність моделей для C++, Rust або Go. Створення якісних багатомовних наборів даних — це складне завдання, яке вимагає більше, ніж простого "перекладу" задач. Мови програмування відрізняються не лише синтаксисом, а й глибинною семантикою: ідіомами, стандартними бібліотеками, управлінням пам'яттю та патернами проєктування.Майбутнє узгодження LLM для коду лежить у створенні агентів, здатних будувати та підтримувати стійкі, динамічні графи знань програмних проєктів, а також в інтеграції формальних методів для забезпечення верифікованих гарантій коректності. Вирішення цих завдань дозволить створити по-справжньому універсальних та незамінних помічників для розробників програмного забезпечення.