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)

Дослідження та використання великих мовних моделей
в задачах обробки природної мови
(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-23-2
Дмитро Сергієнко
(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 проф. Наталія Рябова
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 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 Сергієнку Дмитру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та використання великих мовних моделей в задачах обробки природної мови

затверджена наказом університету від 21 квітня 20 25 р. № 295Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10 червня 20 25 р.

3. Вихідні дані до роботи науково-технічні публікації та дані інтернет-джерел з тематики, досліджень; найбільш відомі проекти зі створення NLP систем, у тому числі спроможніх вирішувати задачі розпізнавання іменованих сутностей; великі мовні моделі

4. Перелік питань, що потрібно опрацювати в роботі
1) Теоретичні основи NLP та LLM
2) Підходи до NER з використанням LLM
3) Дослідження GLINER для задачі NER

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	21.04.2025	виконано
2	Ознайомлення з предметною галуззю	26.04.2025	виконано
3	Визначення структури роботи	30.04.2025	виконано
4	Пошук та аналіз літератури за тематикою	03.05.2025	виконано
	дослідження		
5	Формування постановки завдання	06.05.2025	виконано
6	Теоретичні дослідження	12.05.2025	виконано
7	Експерементальні дослідження	20.05.2025	виконано
8	Написання пояснювальної записки	30.05.2025	виконано
9	Нормоконтроль	04.06.2025	виконано
10	Підготовка презентації та доповіді	01.06.2025	виконано
11	Попередній захист	05.06.2025	виконано
12	Рецензування	06.06.2025	виконано
13	Захист	10.06.2025	

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф Рябова Наталія
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 71 с., 8 рис., 3 табл., 1 дод., 30 джерел.

ВЕЛИКІ МОВНІ МОДЕЛІ, ГЛИБОКЕ НАВЧАННЯ, МАШИННЕ НАВЧАННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ, ТРАНСФОРМЕРИ, FEW-SHOT, GLiNER, ZERO-SHOT.

Об’єкт дослідження – процес розпізнавання іменованих сутностей у текстах природною мовою з використанням великих мовних моделей.

Предмет дослідження – великі мовні моделі, зокрема модель GLiNER, та їх використання в задачах розпізнавання іменованих сутностей.

Мета роботи – аналіз та апробація сучасних великих мовних моделей у задачі розпізнавання іменованих сутностей, зокрема моделі GLiNER, та визначення ефективності їх застосування в умовах обмежених навчальних вибірок (zero-shot, few-shot).

Результатом виконання роботи є підтвердження ефективності використання моделі GLiNER для автоматичного розпізнавання іменованих сутностей та рекомендації щодо її практичного застосування й подальшого розвитку в реальних сценаріях обробки природної мови.

ABSTRACT

Master’s thesis contains: 71 pp., 8 fig., 3 tabl., 1 ann., 30 references.

DEEP LEARNING, FEW-SHOT, GLiNER, LARGE LANGUAGE MODELS , MACHINE LEARNING, NAMED ENTITY RECOGNITION, NATURAL LANGUAGE PROCESSING, TRANSFORMERS, ZERO-SHOT.

The object of the research is the process of Named Entity Recognition (NER) in natural language texts using large language models.

The subject of the research is large language models (LLMs), specifically the GLiNER model, and their applications in Named Entity Recognition tasks.

The aim of this thesis is to analyze and experimentally validate modern large language models in the task of Named Entity Recognition, specifically the GLiNER model, and to determine their efficiency under limited training scenarios (zero-shot, few-shot).

The result of the research confirms the effectiveness of the GLiNER model in automatic Named Entity Recognition and provides recommendations for its practical usage and further development in real-world NLP scenarios.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 10
Вступ 11
1 Теоретичні основи NLP та LLM 13
1.1 Визначення та задачі NLP 13
1.2 Від статистичних методів до трансформерів в NLP 14
1.3 Визначення і типологія LLM 16
1.4 Основні способи використання LLM 20
2 Підходи до NER із використанням LLM 23
2.1 Задача NER сутність, приклади, розмітка, дані 23
2.2 Проблеми в NER 25
2.2.1 Вкладені та перехрещені сутності 26
2.2.2 Шуми та невідповідна розмітка 26
2.2.3 Зсуви доменів та їх адаптація 28
2.2.4 Амбівертність й розв’язування співвідносних назв 28
2.2.5 Іменовані сутності, що не є послідовностями токенів 29
2.3 Архітектури-моделі у NER: від BERT до GLiNER 29
2.3.1 Токен-класифікація 30
2.3.2 Підхід через генерацію 30
2.4 Огляд досліджень з фокусом на LLM у NER 32
3 Дослідження GLINER для задачі NER 35
3.1. Обґрунтування вибору моделі GLiNER 35
3.1.1. Актуальність GLiNER у контексті сучасного NER 35
3.1.2. Відмінність GLiNER від класичних та генеративних підходів 36
3.1.3. Переваги використання у zero-shot та few-shot 37
3.2. Архітектура та принцип роботи моделі GLiNER 38
3.2.1. Формат вхідних даних 38
3.2.2. Використання DistilBERT / DeBERTa як базової моделі 40
3.2.3. Механізм побудови ембеддингів типів та спанів 40

3.2.4. Метод відповідності і обчислення ймовірностей 42
3.2.5. Жадібний декодинг та вкладеність 43
3.3. Навчання моделі на корпусі Pile-NER 45
3.4. Практичне застосування GLiNER для українських email-повідомлень 46
3.4.1. Формування задачі та типізація сутностей 46
3.4.2. Підготовка даних для донавчання 48
3.4.3. Навчання моделі 52
3.4.4. Оцінка якості моделі 56
3.4.5. Інтеграція та застосування 59
3.4.6. Подальші напрямки вдосконалення 64
Висновки 67
Перелік джерел посилання 68
Додаток А Відомість кваліфікаційної роботи 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BERT – Bidirectional Encoder Representations from Transformers – двонаправлені контекстні представлення з трансформерів;

CRF – Conditional Random Fields – умовні випадкові поля;

FFN – Feed-Forward Network – прямопередавальна нейронна мережа;

GLiNER – Generalist and Lightweight NER – універсальна та легка модель для розпізнавання іменованих сутностей;

GRU – Gated Recurrent Units – рекурентні блоки з керованими вентилями;

LLM – Large Language Model – велика мовна модель;

LSTM – Long Short-Term Memory – довготривала короткочасна пам’ять;

NER – Named Entity Recognition – розпізнавання іменованих сутностей;

NLP – Natural Language Processing – обробка природної мови;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

UIE – Unified Information Extraction – уніфіковане вилучення інформації.

ВСТУП

У сучасному інформаційному суспільстві обсяг текстових даних стрімко зростає, а їх ефективний аналіз набуває критичного значення для вирішення широкого кола завдань: від автоматичного перекладу та інформаційного пошуку до аналізу настроїв, класифікації документів і вилучення важливих відомостей. Традиційні методи обробки природної мови NLP), такі як правило-орієнтовані підходи або класичні алгоритми машинного навчання, вже не в змозі ефективно справлятися з масштабами, складністю та різноманітністю сучасних текстових ресурсів.

Поява великих мовних моделей (LLM) ознаменувала принципово новий етап у розвитку NLP. Мовні моделі, зокрема ті, що базуються на архітектурі трансформерів, такі як GPT, BERT, та їх численні модифікації, значно розширили можливості автоматичного розуміння та генерації тексту [6]. Завдяки здатності враховувати широкий контекст і семантичні нюанси, LLM продемонстрували високу ефективність у вирішенні таких завдань, як розпізнавання іменованих сутностей (Named Entity Recognition, NER), класифікація текстів, питання-відповіді, генерація тексту, а також автоматична анотація та реферування документів.

Розпізнавання іменованих сутностей є однією з ключових задач у NLP, що полягає у виділенні з тексту важливих інформаційних одиниць, таких як імена людей, назви організацій, географічні об'єкти, дати тощо. Традиційні методи NER, які спираються на ручну розмітку даних і фіксований набір категорій, не можуть швидко адаптуватися до нових завдань і доменів. Водночас генеративні моделі, хоч і демонструють значну універсальність, часто виявляються неефективними для широкомасштабного застосування через велику кількість параметрів та високі обчислювальні витрати [13].

Особливе місце серед сучасних підходів займають моделі, які поєднують гнучкість генеративних мовних моделей із ефективністю

класичних нейромережних архітектур. Однією з таких моделей є GLiNER, яка реалізує підхід відкритого набору сутностей та демонструє високі результати як у zero-shot, так і у few-shot сценаріях, при цьому залишаючись порівняно компактною та швидкою. Дослідження цієї моделі та її практичне використання для задач розпізнавання іменованих сутностей є перспективним напрямом, що дозволяє поєднати переваги великих мовних моделей із практичною застосовністю у реальних системах.

Актуальність теми роботи визначається стрімким розвитком технологій обробки природної мови та зростаючими потребами суспільства в ефективних і гнучких засобах аналізу текстової інформації. Особливо гостро ця потреба відчувається в умовах інформаційного перенасичення, коли швидке та точне вилучення релевантних даних стає критично важливим для ухвалення обґрунтованих рішень у бізнесі, науці та державному управлінні. Тому дослідження та розробка методів, що базуються на великих мовних моделях, є не лише актуальним науковим завданням, а й має практичну значущість для широкого спектра галузей сучасної діяльності.

Під час підготовки кваліфікаційної роботи було підготовлено та представлено публікації: «Challenges in Named Entity Recognition» в рамках 29-го Міжнародного молодіжного форуму Радіоелектроніка та молодь у XXI столітті: матеріали, 16–19 квітня 2025 р., м. Харків[3], а також робота «Комплексний підхід до NER: вкладеність, шум і доменна адаптація» в межах міжнародної науково-практичної конференції студентів і молодих учених, Інформаційні технології в соціокультурній сфері, освіті та економіці, 23–24 квітня 2025 р., м. Київ [25].

1 ТЕОРЕТИЧНІ ОСНОВИ NLP ТА LLM

1.1 Визначення та задачі NLP

Обробка природної мови (NLP) – це напрям штучного інтелекту й комп’ютерної лінгвістики, що вивчає комп’ютерний аналіз і синтез людської мови. Інакше кажучи, NLP займається тим, як навчити машини розуміти значення тексту і генерувати осмислені речення природною мовою. Основна мета NLP – надати комп’ютерам можливість «розуміти» людську мову, щоб ефективно взаємодіяти з людьми та отримувати знання з текстів [18].

Прикладні задачі NLP дуже різні. До низькорівневих задач належать токенізація (розбиття тексту на слова чи речення), морфологічний аналіз (визначення частин мови та основ слів), лематизація та стемінг (приведення слів до базової форми). На більш високому рівні вирішуються задачі семантичного аналізу та розуміння тексту: розпізнавання іменованих сутностей, класифікація текстів за темою або тоном, аналіз почуттів, видобування відношень, відповіді на запити тощо. Окремо виділяються генеративні задачі – машинний переклад, резюмування тексту, побудова діалогових систем (чатботів), генерація зв’язного тексту. NLP сьогодні відіграє ключову роль у багатьох системах: пошукові машини використовують обробку мови для розуміння запитів, голосові помічники (такі як Siri, Alexa) – для розпізнавання мовлення та формування відповіді, соціальні мережі – для фільтрації контенту, а бізнес-аналітика – для обробки клієнтських відгуків природною мовою.

Одним із базових прикладів позначення тексту в NLP є схема BIO (або IOB) для розмітки іменованих сутностей. В цій нотації кожен токен (слово) помічається як початок сутності (B-), всередині сутності (I-) або поза сутностями (O). Наприклад, речення «Кеті Пері приїхала до Києва.» може бути розмічене так:

Кеті B-PER

Пері I-PER

приїхала O

до O

Києва B-LOC

. O

Тут «PER» позначає людину, «LOC» – локацію. Відмітки B- та I- вказують на початок та продовження сутності відповідно. Схему BIO було запропоновано для задачі часткового розбору тексту, і вона стала стандартом для NER[15]. BIO розмітка дозволяє точно визначати границі сутностей навіть у випадку, коли кілька слів підряд належать до однієї сутності (наприклад, ім’я + прізвище для персони, як «Кеті Пері»).

Прикладні галузі застосування NLP охоплюють майже всі сфери, де потрібен аналіз текстових даних. В медицині – автоматичне виділення діагнозів і назв ліків з лікарських звітів; у фінансах – аналіз новин та звітів для прогнозування ринків; в юридичній сфері – автоматизація аналізу договорів; у біоінформатиці – витяг фактів про гени та білки з наукових публікацій. Зокрема, техніки NER широко застосовуються для побудови баз знань та отримання інформації: виділені сутності (імена людей, організацій, локацій, дат тощо) можна використовувати для побудови знань, пошуку та індексування даних.

1.2 Від статистичних методів до трансформерів в NLP

На ранніх етапах розвитку NLP (1960–1990-ті роки) переважали правила та статистичні методи. Перші системи обробки мови були засновані на експертних правилах і словниках. Згодом з’явилися статистичні моделі, такі як N-грамні моделі для оцінки ймовірності послідовностей слів та приховані марковські моделі [11] для позначення послідовностей (наприклад, частин мови). У 1990-х набули популярності

методи на основі максимального ентропійного моделювання та CRF – вони успішно застосовувалися, зокрема, для задачі NER, дозволяючи враховувати контекст сусідніх слів при визначенні мітки токена.

З середини 2000-х років відбувся перехід до нейронних мереж. Спочатку використовували RNN – архітектури, здатні обробляти послідовності довільної довжини завдяки циклічним зв'язкам. Важливою віхою стало введення модифікацій RNN: мереж довгої короткочасної пам'яті LSTM [11] та вентильні рекурентні вузли (GRU), які розв'язували проблему «згасаючих градієнтів» і могли зберігати інформацію на більшій відстані в тексті. RNN успішно застосовувалися для машинного перекладу, послідовні моделі «encoder-decoder» з двох LSTM започаткували підхід sequence-to-sequence[19]. Проте фундаментальним обмеженням рекурентних мереж є послідовний характер обчислень: RNN обробляють слова по одному за раз від початку до кінця, що унеможливлює паралельну обробку і утруднює засвоєння дуже довгострокових залежностей. Навіть з механізмом уваги традиційні RNN-системи все одно проходять слова послідовно, що обмежує їх продуктивність.

Революційним кроком стало впровадження механізму самоуваги та моделі трансформер. Трансформери повністю відмовилися від рекурентності, натомість введена мультиголовну увагу для обробки всіх слів паралельно. Це дозволило моделі бачити весь контекст речення одразу і вловлювати залежності між будь-якими парами слів, незалежно від їх відстані.

У статті «Attention Is All You Need» трансформер показав видатні результати у машинному перекладі, перевершивши попередні рекурентні моделі, і фактично започаткував нову парадигму в NLP. Головна ідея – увага дозволяє моделі зважувати важливість кожного слова відносно інших при побудові внутрішнього представлення, що значно ефективніше для уловлювання смислових взаємозв'язків [1]. Трансформери усунули проблему послідовності RNN, проте мають іншу ціну: їх час обчислення

зростає квадратично зі збільшенням довжини вікна контексту. Попри це, переваги переважили недоліки, і трансформер став фундаментальною моделлю сучасного NLP.

Однією з найвпливовіших розробок став BERT – велика двонапрямна трансформерна модель. BERT показав, як можна попередньо навчити глибоку двонапрямлену модель на величезному нерозміченому корпусі текстів, а потім ефективно донавчати її під конкретні задачі. Завдяки маскованому прогнозуванню слів BERT навчився контекстних представлень, що враховують як лівий, так і правий контекст слова [2].

Вже перша версія BERT встановила нові рекорди у 11 різних NLP-завданнях одночасно, зокрема в системах питання-відповідь і логічними висновками. BERT став прототипом енкодерних трансформерів для розв’язання задач класифікації та витягування інформації. Його успіх започаткував ера великих попередньо навчених моделей у NLP: дослідники зрозуміли, що одна універсальна модель, навчена на величезному корпусі, може бути адаптована під різні задачі з небаченою раніше якістю.

Отже, еволюція NLP-моделей пройшла шлях від простих статистичних моделей до складних нейронних архітектур. Ключовими віхами стали: впровадження рекурентних мереж для роботи з послідовностями; поява механізму уваги для подолання обмежень RNN; створення трансформерів – які кардинально змінили підхід до обробки тексту; поява великих претренованих-моделей на основі трансформерів, таких як BERT – що продемонстрували універсальність і силу попереднього навчання на великих даних.

1.3 Визначення і типологія LLM

Великі мовні моделі (LLM) – це мовні моделі нового покоління, які характеризуються величезною кількістю параметрів (сотні мільйонів, мільярди і більше) та масштабними навчальними корпусами. Такі моделі

навчаються на терабайтах текстових даних, часто охоплюючи багато мов і різні домени, що надає їм універсальних знань про мову. Великі параметри дозволяють LLM засвоювати складні мовні патерни, емерджентні властивості (здатність виконувати нові завдання, не явно представлені в тренувальних даних). Історично першим LLM, що привернув масову увагу, став GPT-3 [8]з 175 мільярдами параметрів. Ця модель показала, що просте масштабування параметрів і даних здатне дати якісний стрибок: GPT-3 вміє вирішувати широкий спектр NLP-задач без додаткового навчання, лише отримуючи підказки-промпти. Це привернуло увагу до такого підходу та з'явилися ще більші моделі – наприклад, багатомовна модель XLM-R для багатьох мов [5], PaLM від Google з 40 млрд параметрів GPT-4 тощо.

LLM, як правило, належать до категорії фундаментальних моделей – вони не прив'язані до однієї задачі, а слугують універсальною основою, яку можна застосувати в різних контекстах. Їх універсальність пояснюється масштабом: вони поглинають знання з інтернет-корпусів, тому можуть генерувати зв'язний текст, писати програми, відповідати. Важливо зазначити, що великими такі моделі робить не лише кількість параметрів, а й обсяг та різноманітність корпусу: чим більше даних і чим вони різноманітніші, тим краще модель охоплює мовні явища. Проте надвеликі розміри приводять до колосальних витрат ресурсів на навчання, що робить LLM доступними переважно великим компаніям.

За архітектурою великі мовні моделі можна поділити на три основні категорії: енкодери, декодери та енкодер-декодерні моделі.

– Encoder-only LLM – це моделі, що складаються лише з блоків-енкодерів трансформера. Вони добре підходять для задач розуміння тексту та класифікації, де потрібно отримати глибоке представлення вводу. Приклад – згаданий BERT (340 млн параметрів у версії Large) або його полегшені/модифіковані версії: RoBERTa [21], DistilBERT [4], багатомовний XLM-RoBERTa [28]. Encoder-моделі як BERT навчаються маскувати слова і відновлювати їх, тому добре захоплюють контекст з обох

боків слова. Вони універсальні для класифікаційних задач: NER, визначення тональності, тематична класифікація, екстракція фактів – усюди, де треба позначити чи класифікувати наявний текст, енкодери показують високі результати. Недолік – такі моделі не генерують новий текст, їх вихід обмежений мітками або вибором з наявних слів.

– Decoder-only LLM – моделі, що містять лише декодери трансформера. Вони навчаються прогнозувати наступне слово у послідовності, тому підходять для генерації тексту. Приклади: усі моделі GPT (GPT-2, GPT-3, GPT-4), а також відкриті аналоги на кшталт GPT-Neo/GPT-J чи LLaMA [16]. Декодерні LLM відзначаються здатністю продовжувати заданий текст, писати осмислені відповіді чи довгі статті, кодувати програми тощо. Такі моделі чудово підходять для задач, де треба згенерувати розв’язок – творчі завдання, діалоги, заповнення пропусків, перефразування. В контексті NER декодерні LLM можуть описати знайдені сутності у вільній формі або видати результат у вигляді відформатованого тексту. Перевага – гнучкість, але є і недоліки: схильність до галюцинацій, тобто вигадування недостовірної інформації, відсутність спеціалізації – без підказки модель може не здогадатися, що потрібно саме знайти сутності, складність додавання нових навичок через величезний розмір (такі моделі важко донавчати);

– Encoder-decoder LLM – гібридна архітектура, що складається з енкодера і декодера. Енкодер читає вхідний текст і стискає значення в контекстне представлення, а декодер генерує вихід на основі цього представлення. Такі моделі зручні для перетворення формату: машинний переклад (вхід англійською – вихід українською), сумаризація (вхід стаття – вихід короткий зміст), відповіді на питання (вхід питання + контекст – вихід відповідь) тощо. Відомі приклади – трансформери T5 [7] і BART[2]. Модель T5 запропонувала підхід текст до тексту: вона представляє будь-яке NLP-завдання як перетворення тексту у текст (наприклад, для класифікації вхід – це речення, а вихід – назва категорії як слово). Це дозволило єдиній encoder-

decoder моделі досягти високих результатів на різних задачах. В контексті NER енкодер-декодерні моделі можуть приймати на вхід речення, а на виході генерувати перелік знайдених сутностей чи переформатований текст із позначками.

У таблиці 1.1 наведено порівняння типів LLM та їх впливу на NLP.

Таблиця 1.1 – Порівняння LLM

Архітектура	Приклад моделей	Типові задачі	Сильні сторони	Обмеження
Енкодер	BERT, RoBERTa, XLM-R	Класифікація, NER, оцінка подібності, видобування фактів	Глибоке розуміння контексту, висока точність на розмічених задачах	Не генерує текст; потребує додаткового шару для кожної задачі
Декодер	GPT-3, GPT-4, LLaMA	Генерація тексту, діалоги, творчі задачі	Відмінна генерація, універсальність (zero-shot/few-shot)	Галюцинації, відсутність явного контролю виходу, обмежена довжина контексту
Енкодер-декодер	T5, BART, mT5	Переклад, резюмування, питання-відповідь, перетворення форматів	Може узгодити вхід з виходом (контекст та відповідь), гнучкість форматів	Великий розмір, складність навчання двох частин, не оптимізована під чисту генерацію довгого тексту

Масштаб LLM значно вплинув на результати в NLP. Більш великі моделі, як правило, демонструють кращі результати в zero-shot і few-shot сценаріях. Але продуктивність не зростає нескінченно.

Наприклад, GPT-3 (175B) майже в 100 разів більший за свого попередника GPT-2 (1.5B) і значно перевершує його по ряду задач, але подальше збільшення до GPT-4 дало менш помітний якісний стрибок, більше фокусуючись на безпеці та точності[10]. Важливо відмітити, що LLM середніх розмірів, навчені на якісних даних, можуть змагатися з більшими моделями. Це демонструє, що корпус і підготовка даних не менш важливі, ніж кількість параметрів.

1.4 Основні способи використання LLM

Великі мовні моделі можна використовувати різними способами залежно від потреб задачі. Виділимо кілька основних підходів: попереднє навчання, донавчання, промптинг та навчання в контексті.

LLM зазвичай проходить тривалу фазу попереднього навчання на великому корпусі нерозмізченого тексту. На цьому етапі модель вчиться мовної залежності загального характеру: імовірності послідовностей слів, світові факти, мовні закономірності. Наприклад, GPT-3 навчався на корпусі обсягом приблизно 45 ТБ текстів. Попереднє навчання створює універсальні мовні представлення, які потім можуть бути адаптовані під конкретні задачі.

Донавчання – це адаптація великої моделі під конкретну задачу на основі розмічених даних. Класичний приклад – донавчання BERT під задачу NER: до виходу BERT додається шар класифікатора токенів, і модель донавчається на розміченому корпусі (наприклад, датасеті CoNLL-2003 з NER-мітками). Таке кероване навчання дозволяє LLM підлаштуватися під домен і вимоги задачі, часто досягаючи найвищої якості. Fine-tuning ефективний навіть при відносно невеликих вибірках, адже LLM вже знає «мову» загалом. Останні дослідження показали також дієвість інструкційного донавчання – коли LLM донавчається відповідати на інструкції, сформульовані людською мовою (як це зроблено для моделей GPT-3.5, GPT-4).

Спеціалізовані версії LLM з’явилися і для NER: наприклад, моделі InstructUIE [12] – донавчений Flan-T5 (11B) під задачі інформаційної екстракції, UniNER – модель на основі LLaMA-13B, донавчена витягувати будь-які типи сутностей через підказки. Донавчені моделі зазвичай показують чудові результати на конкретному наборі даних, випереджаючи універсальний zero-shot підхід. Наприклад, BERT, донавчений на NER, може давати F1 > 90% на стандартних наборах (CoNLL), в той час як zero-

shot GPT-3 суттєво відставав. Проте мінус – треба мати розмічені дані, і для кожної нової сутності чи мови бажано донавчати заново.

Промптинг – новий режим роботи з LLM, коли модель не донавчається, а скеровується промптом – спеціально сформульованим текстовим запитом, який включає інструкцію або приклади. Цей підхід став особливо актуальним після успіху GPT-3, продемонструвавши, що LLM можуть донавчатися з кількома прикладами. Режими промптингу можна поділити на: zero-shot, few-shot та навчання в контексті.

Zero-shot – модель отримує лише інструкцію, без жодного прикладу, і повинна генерувати відповідь. Наприклад, для NER zero-shot промпт може бути: «Extract all person names from the text: ...», і модель видасть перелік імен. Це працює, бо LLM розуміє формулювання задачі завдяки знанням, набутим під час pretraining. Однак zero-shot може давати збої, якщо інструкція не достатньо точна або задача складна.

Few-shot – до інструкції додаються декілька прикладів виконання задачі (демонстрації). LLM читає ці приклади і за рахунок цього налаштовується краще. Такий режим називають також in-context learning – модель як би «навчається в контексті запиту», не змінюючи своїх ваг. Наприклад, для NER можна надати 2-3 приклади речень з уже позначеними сутностями, і далі попросити помітити сутності в новому реченні. GPT-3 показав сильні результати в few-shot режимі, подекуди наближаючись до якості моделей, що тренувалися на задачі. Few-shot prompting – дуже гнучкий інструмент: він не потребує затратного fine-tuning, достатньо кількох правильних прикладів у запиті.

Промптинг з роздумами – спосіб, коли модель у промпті просять спочатку пояснити хід міркувань, а потім дати відповідь. Це інколи підвищує точність на складних логічних задачах. Для NER CoT(Chain-of-thought) не є типовим, але може використовуватись для пояснення, чому те чи інше слово є сутністю.

Для задачі NER існують цікаві приклади використання LLM через промптинг. Зокрема, метод PromptNER [21] поєднує генеративну модель і спеціальний формат підказки для few-shot NER. PromptNER пропонує шаблон промпту з двома слотами – позиції і типу сутності, куди підставляються кандидати, і модель одночасно вирішує і локалізацію, і класифікацію сутностей. В експериментах PromptNER досяг новітніх результатів на наборах CoNLL та FewNERD у few-shot режимі, перевершивши попередні методи на 4-9% F1. Причому для цього знадобилося лише 2% від повного обсягу тренувальних даних – модель навчилася отримувати сутності, спираючись на кілька прикладів та гнучкий промпт, що описує нові типи сутностей. Такий підхід вигідний, коли треба швидко адаптуватись до нової доменної області або нестандартного набору сутностей без повноцінного навчання з нуля.

Інший приклад – GPT-NER [11], підхід, що перетворює стандартну задачу NER (послідовне маркування токенів) на задачу генерації тексту для LLM. GPT-NER пропонує вставляти спеціальні маркери в текст: наприклад, перетворити речення «Columbus is a city» на «@@Columbus## is a city» для позначення, що «Columbus» – сутність місце. Велика модель тоді просто генерує відформатований текст з маркерами, а не передбачення на кожен токен. Це дозволяє обійти розрив між природою LLM (генерувати суцільний текст) і вимогою NER (дати мітку кожному слову). Важливим компонентом GPT-NER є самоперевірка моделі. Оскільки LLM схильна «вигадувати» сутності, там де їх немає, додано крок, де модель отримує власний результат і первіряє «чи дійсно слова належать до відповідного типу сутності?». Це знижує число хибних спрацьовувань. GPT-NER показала гідні результати, в порівнянні з спеціалізованими моделями NER, і вперше продемонструвала, що генеративний підхід може конкурувати з традиційним навчанням. Особливо помітна перевага GPT-NER проявилася в умовах нестачі даних. В ситуації коли розмічених прикладів дуже мало, GPT-NER значно перевершила класичні моделі, які без великого корпусу не працюють.

2 ПІДХОДИ ДО NER ІЗ ВИКОРИСТАННЯМ LLM

2.1 Задача NER сутність, приклади, розмітка, дані

Розпізнавання іменованих сутностей – це завдання виділити у тексті фрагменти, що відповідають реальним об’єктам, і класифікувати їх за наперед визначеними категоріями. Простіше кажучи, NER знаходить в реченнях імена людей, назв компаній, географічні назви, дати, суми, назви творів тощо і присвоює їм відповідний тег (PERSON, ORG, LOCATION, DATE, MONEY, WORK_OF_ART тощо) [17]. Результат роботи NER зображено на рисунку 2.1.

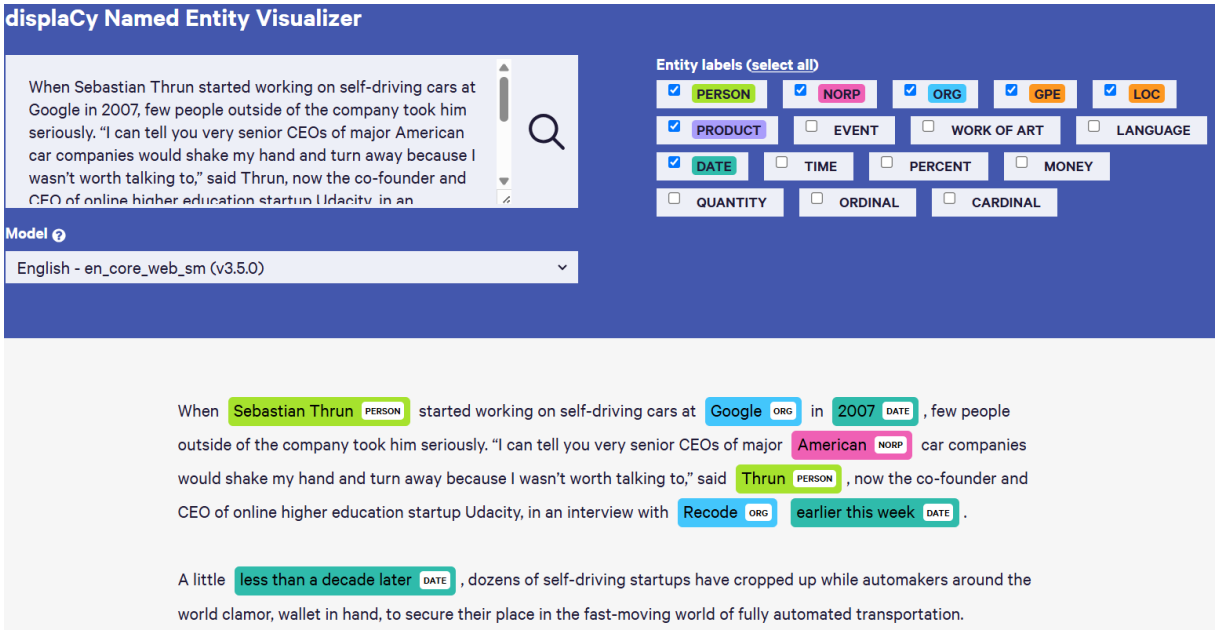

Рисунок 2.1 Приклад роботи NER

Виділені сутності надають структуроване представлення неструктурованого тексту, що дуже корисно для подальшого інформаційного пошуку, побудови баз знань, аналізу зв’язків. Наприклад, з повідомлення «Apple відкрила новий офіс у Лондоні в липні 2023 року» NER повинна отримати: Apple – Organization, Лондон – Location, липень

2023 – Date. Це дозволить далі відповісти на запит «які компанії відкривали офіси в Лондоні у 2023?» через пошук (компанія-місто-дата).

Приклади сутностей залежать від домену. В новинах типові категорії – Person, Organization, Location, Event, Date, Money. В біомедичному тексті – назви генів, білків, захворювань, хімічних сполук. У твітах можуть бути мітки для емодзі чи хештегів як окремих сутностей. В цілому, схема категорій визначається потребами: CoNLL-2003, класичний датасет, містить 4 типи (PER, ORG, LOC, MISC), OntoNotes 5 (більш масштабний корпус) має 18 типів (додає NORP – національності, LAW – закони, PRODUCT тощо). У промислових застосунках часто формують власний набір сутностей. Примітно, що сучасні open-NER системи прагнуть дозволити довільні типи – тобто модель на льоту отримує список типів, які треба шукати.

У попередньому розділі розглянуто формат розмітки BIO для токенів. Альтернативні схеми – BIO2, BILOU (Begin, Inside, Last, Outside, Unit), де позначається не тільки початок, а й кінець сутності. Існують також підходи не до кожного токена, а до діапазонів символів наприклад, задавати сутність як позиції початку і кінця в рядку. В сучасних датасетах розмітка зазвичай надається або в BIO-форматі по токенах, або у вигляді списку сутностей з їхнім текстом і позиціями в документі (корисно коли працюємо з довгими текстами). В NER дані часто організовані у вигляді окремих фраз чи речень, кожне з яких надходить з відповідними тегами.

Разом з тим, з поняттям розмітки з’являється проблема і вкладених сутностей: коли одна сутність повністю знаходиться всередині іншої. Приклад:

[Minister of economy [John Doe]PER]TITLE» (тут John Doe – людина, яка також частина більшої сутності «Minister of economy John Doe» з типом посада/титул).

Стандартний BIO-формат не дозволяє прямо помітити вкладеність, тому для таких підзазадач NER існують розширені схеми або багатошарова

розмітка (окрема розмітка для кожного рівня сутностей). Це значно ускладнює задачу, і багато класичних моделей не підтримували вкладені сутності.

Не менш важливою частиною є набори даних. Для англійської мови найвідоміший – CoNLL-2003, який містить розмічені новини агентства Reuters (4 типи сутностей). CoNLL-2003 стала класичним тестом для нових моделей NER – ще з 2003 року точність F1 зросла з 88% (методи на основі CRF) до 93-94% (трансформери).

Інший великий корпус – OntoNotes 5.0, який використовувався у Shared Task 2011 (CoNLL-2011/2012) для задач coreference, але містить і мітки NER (18 типів, тексти різних жанрів). OntoNotes став основою для багатьох багатомовних досліджень, бо містить дані не тільки англійською, а й китайською, арабською. Для української та мови розмічених NER даних значно менше.

Сучасні ресурси включають також специфічні доменні набори: фінансові документи, медичні статті (наприклад, BC5CDR для хімічних речовин та хвороб), соціальні мережі (WNUT – наприклад, WNUT-2017 для Twitter з незвичними сутностями типу «CreativeWork»). Для open-NER, де немає фіксованих типів, з’явився датасет Pile-NER [27] –сутності з великого текстового корпусу Pile, він містить 240 тисяч сутностей 13 тисяч різних типів.

2.2 Проблеми в NER

Незважаючи на значні успіхи у вирішенні задачі NER із використанням LLM, у реальних застосуваннях дослідники та інженери стикаються з низкою суттєвих викликів.

Далі розглянуто ключові проблеми, які ускладнюють побудову надійних NER-систем.

2.2.1 Вкладені та перехрещені сутності

Однією з таких проблем є вкладені та перехрещені сутності. Вкладені стуності це ситуація, коли одна сутність повністю знаходиться всередині іншої. Це часто трапляється в біомедичних текстах і новинах. Стандартні моделі, що маркують кожне слово однією міткою, не можуть одночасно відзначити два різні типи на тому ж фрагменті. Для вирішення застосовують або двоетапні підходи (спочатку виділити всі можливі сутності, потім відфільтрувати некоректні вкладення), або спеціальні архітектури.

Наприклад, існують моделі, які генерують списки сутностей замість послідовності тегів – тоді ніщо не заважає двом спискам перекриватися. Інший метод – навчати окремі моделі на різні рівні вкладеності. Є роботи, що перетворюють вкладені задачі на набір задач BIO для кожного рівня вкладення. Проте універсального рішення досі нема: якщо сутності можуть перетинатися чи вкладатися довільно, задача стає набагато складнішою (NP-повною в загальному випадку).

2.2.2 Шуми та невідповідна розмітка

Іншою важливою проблемою є шуми в даних та невідповідна розмітка. Розмічені корпуси часто містять помилки або непослідовності. Анотатори можуть різнитися у вирішенні, чи вважати певний термін сутністю. Наприклад, у корпусі CoNLL «the White House» може бути помічено як ORG (адміністрація) або як LOC (будівля) залежно від контексту і суб’єктивного рішення. Такі неоднорідності ускладнюють навчання – модель може збиватися через конфліктні приклади. Крім того, тексти з живих джерел (особливо соціальні мережі) містять помилки, сленг, нестандартні скорочення, емодзі – все це є шумом для моделі[4]. Система NER, навчена на новинах, різко втрачає точність на твітах чи чатах без додаткової адаптації.

Деякі дослідження зосереджені саме на надійності NER – підвищенні стійкості до шуму. Наприклад LinkNER, що комбінує звичайну NER-модель з LLM: якщо маленька модель не впевнена (висока невизначеність), то підключається LLM для уточнення. Така гібридна система суттєво підвищує надійність на неочищених даних, даючи +3–21% F1 на складних тестах порівняно зі стандартними підходами.

Інший напрям – очищення та узгодження розмітки це вручну чи автоматично виправляти помилки в датасетах, але це складна задача при великих обсягах даних, тому моделі навчаються з певною толерантністю до шуму, наприклад, через додавання негативних прикладів та регуляризацію[24].

Для багатьох мов та доменів розмічених корпусів дуже мало. Моделі, навчені на англ. даних, складно перенести через відмінності мов.

Проблему вирішують трьома шляхами: трансферне навчання – використання багатомовних моделей (типу mBERT, XLM-R), які на спільному підґрунті дозволяють перенести знання з одних мов на інші. Наприклад, для непальскої мови найкраще спрацювала локальна модель NepBERTa (спеціально перед навчена BERT для непальскої мовви), яка перевершила мультимовні аналоги на тій же задачі[8]. Водночас класичні моделі (BiLSTM+CNN+CRF) все ще були конкурентними, що підкреслює цінність ручної розмітки навіть невеликих корпусів.

В свою чергу якщо є невелика кількість розмічених зразків, можна штучно розширити вибірку – замінювати сутності на інші з того ж класу, генерувати нові речення за допомогою правил або LLM. Сучасні генеративні моделі можуть створювати правдоподібні речення з заданими сутностями, розширюючи тим самим тренувальний набір.

У випадку зовсім відсутніх даних для мови – як-от NER для рідкісних мов – можна використати великі багатомовні моделі типу GPT-4 чи LLaMA-2, які вже знають багато мов.

Дослідження показують, що GPT-4 здатен з певною точністю виконати NER навіть для мов, на яких його не навчали, через загальнолюдські назви та схожість контекстів. Проте точність zero-shot може бути невисокою. Для покращення можна застосувати переклад промпту, наприклад, у задачі NER для непальської – перекласти непальський текст на англійську.

2.2.3 Зсуви доменів та їх адаптація

Моделі, треновані на одному жанрі тексту, втрачають точність на іншому. NER-модель, навчена на новинах, може погано працювати на медичних статтях (де зовсім інші типи сутностей) [30]. Для її вирішення використовують або донавчання на невеликій кількості цільових даних (якщо вони є), або few-shot промпти з описом нових типів. Існують також методи самоадаптації: модель проганяє незнайомий корпус, вибирає найімовірніші варіанти як псевдо-розмітку і донавчається на них. Особливо актуально це стало з появою LLM: з одного боку, вони більш стійкі до доменного зсуву, бо при попередньому навчанні бачили різні жанри; з іншого – вони можуть розуміти специфічних термінів. Тут часто допомагає доповнення знаннями – наприклад, LLM + база знань для підказок про нові сутності.

2.2.4 Амбівертність й розв’язування співвідносних назв

Деякі сутності важко розпізнати через двозначність. Наприклад, слово «Мау» – це може бути ім’я людини або назва місяця (травень). Моделі інколи плутаються, якщо контекст не дає явних підказок. У діалогах виникає проблема, коли на одне питання є кілька схожих сутностей. Для діалогових систем вводиться механізм порівняння кандидатів: по суті, модель GPT-2 генерує варіанти, а BERT перевіряє, який з них підходить краще [22]. Це

приклад комбінованого підходу: генерація + ранжування для зняття неоднозначності.

2.2.5 Іменовані сутності, що не є послідовностями токенів

Наприклад, даті «2023-05-22» або email-адреса «john.doe@example.com» не так просто призначити BIO-теги (вони містять символи, які при токенізації розіб’ються). Такі випадки потребують або спеціальної обробки на рівні символів, або попереднього нормалізування. Сучасні моделі частково з цим справляються (за рахунок субтокенів), але загалом розпізнавання небезперервних або дисконтінуальних сутностей – додатковий виклик.

2.3 Архітектури-моделі у NER: від BERT до GLiNER

За останні роки в NER утворилося дві головні парадигми моделювання: токен-класифікація та генерація послідовностей. Розглянемо їх та приклади моделей кожного типу, включно з сучасними інноваціями на основі LLM.

На рисунку 2.2 зображено загальну структуру розвитку NER [22].

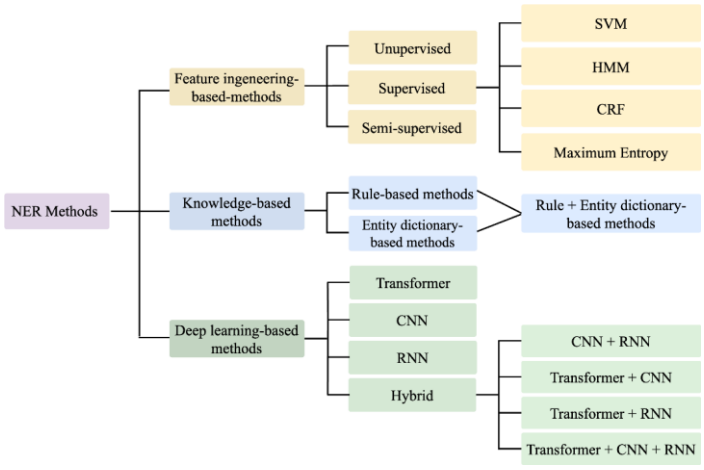

Рисунок 2.2 Основні підходи до NER.

2.3.1 Токен-класифікація

Історично більшість NER-систем будувалися як модель, що присвоює кожному токену мітку (B-PER, I-PER, O тощо). Архітектурно це зазвичай енкодер (наприклад, BERT), на вихідні векторні подання слів якого накладено шар класифікації (наприклад, проста шар, що мапує вектор слова до розподілу ймовірностей по класах BIO). Класичний стек 2010-х – BiLSTM-CRF дає вектор для кожного слова, CRF шар знаходить оптимальну послідовність тегів з урахуванням залежностей між тегами (наприклад, I-PER може йти тільки після B-PER або I-PER, але не на початку речення). З появою трансформерів стандартом де-факто став BERT-Softmax або BERT-CRF: коли вектори BERT проходять через софмтмакс класифікатор або CRF[13]. Моделі BERT, RoBERTa, DistilBERT та XLM-R стали популярними освновами для NER. Вони забезпечують високий рівень точності: наприклад, XLM-R навчений на 100 мов дозволяє виконувати NER багатьма мовами однією моделлю. Для умов нестачі даних використовували техніки тегерів з підстановкою перед навчених векторів, але з часом BERT-вектор на токені (особливо якщо модель була мовно-специфічна чи доменна, як BioBERT) став найкращим входом для класифікації токена.

2.3.2 Підхід через генерацію

Цей напрям став розвиватися завдяки LLM. Його суть: розглядати NER не як задачу маркування кожного слова, а як задачу генерувати деяке представлення сутностей. Можна виділити дві підкатегорії:

– Seq2seq NER – це використання моделі енкодер-декодер, де енкодер читає текст, а декодер видає, скажімо, список сутностей у текстовому вигляді. Приклад – модель UIE, яка може на виході видавати JSON зі знайденими сутностями. Вона тренувалась спеціально генерувати структуровані відповіді. Flan-T5, донавчений на інформаційний екстракшн

InstructUIE – саме такий випадок. Перевага – отримуємо готовий формат, але декодерні моделі можуть бути повільні на довгих текстах, бо генерують токени послідовно;

– Prompt-based generative NER – це коли використовуємо великий декодер (типу GPT-3) і правильно задаємо промпт[21]. Тут модель не навчена спеціально під NER, вона відповідає по суті на питання: «Які сутності є в тексті?». Прямий підхід може не дати максимальної якості, але розробки такі як GPT-NER[11] покращують це, додаючи форматування і самоперевірку.

Порівняно з токен-класифікацією, генеративні методи можуть бути більш гнучкими: наприклад, можна попросити модель вивести не просто сутності, а ще й короткий опис кожної – генеративна модель це зробить (в той час як токен-класифікатор такого не вміє, його треба поєднувати з додатковим компонентом). Також генеративний підхід може легше впоратися з ситуацією, коли набір типів не фіксований (в промпті можна дати різні типи) [14]. Недоліки – можливі галюцинації (LLM може «видумати» сутність, якої не було) та низька ефективність на довгих текстах (токен-класифікатор оцінює все паралельно, а генерація довгого списку – послідовна). У GPT-NER проблему галюцинацій долають самоперевіркою, а проблему ефективності – тим, що модель генерує все заразом. Проте у випадку реального застосування (наприклад, обробки тисяч документів) токен-класифікатор на основі BERT може виявитися швидшим і дешевшим, ніж багаторазове звернення до GPT.

Останнім часом з’явилися моделі, що намагаються зберегти легкість спеціалізованих енкодерів і водночас включати гнучкість LLM. Однією з таких моделей є – GLiNER. GLiNER – компактна модель (сотні мільйонів параметрів, тобто розміром з BERT), яка спеціально тренувалася на open-NER: здатність знаходити будь-які типи сутностей за запитом. Вона використовує двонапрямний трансформер (BiLM) – як основу, і рамку завдання у вигляді «пошуку відповідностей між типами та текстовими

фрагментами». Це щось середнє між класифікацією і генерацією: GLiNER отримує на вхід текст + список типів сутностей, а на виході видає набір знайдених сутностей з кожного типу (порожній, якщо нема) [9]. Таким чином, вона паралельно обробляє всі задані типи, на відміну від діалогового промптингу LLM, де типи могли запитуватись послідовно (як у UniNER, що робив багатокроковий діалог типу «What persons are in text? ... What locations? ...»). Це дає значний виграш у швидкості [27].

2.4 Огляд досліджень з фокусом на LLM у NER

Генеративні підходи PromptNER, GPT-NER. Обидва націлені на використання LLM без повного fine-tuning. PromptNER цікава тим, що вона не змінює параметри LLM, а навчає промптовий шаблон і модуль адаптації (наприклад, невеликий алгоритм, що комбінує відповіді LLM). Це економить час і ресурси: не треба проганяти мільярди параметрів через градієнти, достатньо правильно їх «спитати». PromptNER [21] досягла state-of-the-art для few-shot NER, що показує: LLM з добре продуманим промпт-інжинірингом + постобробкою може змагатися з моделями, що тренуються на тих кількох прикладах. Однак, це було на певних датасетах (CoNLL, GENIA, FewNERD). У реальному світі може бути складніше підібрати prompt, особливо якщо сутності специфічні. PromptNER давала значний виграш над просто Cot промптингом GPT-4, але GPT-4 – комерційна модель, і не завжди доступно зробити 1000 викликів з довгими промптами (це дорого). Отже, PromptNER – чудовий результат, але його застосування може обмежуватися тими випадками, де few-shot і мала вибірка. Наприклад, якщо у вас новий домен з 100 анотованих прикладів – PromptNER підходить. А якщо 10000 – краще вже fine-tune свій моделей.

GPT-NER у своїй статті зробив наголос на few-shot superiority: вони показали графіки, де при <100 прикладах GPT-NER значно кращий, ніж класичні підходи[11]. Це очікувано: LLM вже «навчена» на мові. Але якщо

прикладів багато (1000+), то fine-tuned спеціальна модель переганяє GPT-NER. Так автори і позиціонують: GPT-NER особливо корисний там, де обмаль даних, а також може використовуватися як відправну точку для анотації. Наприклад, можна GPT-NER'ом розмітити 1000 сирих текстів, вручну виправити, і вже на цьому навчити маленьку модель. Це зекономить час розмітки.

LinkNER – цікавий підхід: замість покладатися лише на LLM або лише на модель, вони беруть краще з двох [15]. Ідея – невелика NER-модель (скажімо, BERT-CRF) швидка і точна на звичайних випадках, але пропустить щось нове. LLM (типу GPT-4) знає багато імен і фактів, може розпізнати рідкісне ім'я чи абревіатуру, але повільний. В LinkNER маленька модель працює першою; де вона невпевнена (низька ймовірність або висока ентропія), вона «запитує» LLM підказати сутність. Потім знову перевіряє, об'єднує. В результаті, LinkNER показав високий виграш (до +21% F1), якщо в тексті багато незнайомого шуму. Це вказує на практичну рекомендацію: у важливих системах, де ціна помилки висока, варто робити такий бекап LLM. Наприклад, у системі моніторингу новин: основна модель BERT відловлює 95% потрібних сутностей, а для 5% сумнівних випадків викликається GPT-4, який добиває. Це скоротить витрати (замість обробляти GPT-4 усі тексти, він тільки 1 з 20 обробить, решту – дешева модель). І точність буде навіть вища, ніж якби GPT-4 сам все робив, бо GPT-4 теж помиляється, а маленька модель фільтрує зрозуміле для неї [15].

Зовсім свіжий напрям – навчити LLM вчитися самому для NER. Робота «Self-Improving for Zero-Shot NER» запропонувала такий фреймворк: маємо LLM (наприклад, 7B), є купа сирого тексту (без міток), і немає жодної людської розмітки [26]. Ідея наступна: LLM проганяють весь корпус і збирають його ж передбачення сутностей, причому кілька разів (самодовіра, якщо модель кілька разів видає ту саму сутність, це сигнал довіри). Відфільтровують найбільш надійні «само-розмічені» сутності різними стратегіями. Потім використовують ці надійні псевдо-дані як демонстрацію

для навчання в контексті на тесті. LLM сам собі зробив невелику тренувальну вибірку і нею ж скористався при розв’язанні. Це дозволило суттєво підняти точність zero-shot[23]. Фактично, модель виконала роль і аннотатора, і вирішувача. Це цікавий підхід, бо він не потребує взагалі людських даних. Звісно, він працює якщо LLM хоч трохи може визначати сутності сам (інакше він назбирає помилок і їх же відтворить). Та для відомих типів (як PER, ORG, LOC) це виглядає перспективно: можна взяти корпус Вікіпедії українською, яка буде розмічена ChatGPT, відібрати надійні – і отримати українську NER без жодної ручної розмітки.

3 ДОСЛІДЖЕННЯ GLINER ДЛЯ ЗАДАЧІ NER

3.1 Обґрунтування вибору моделі GLiNER

3.1.1 Актуальність GLiNER у контексті сучасного NER

У сучасному NER спостерігається перехід від класичних моделей з фіксованим набором категорій до більш гнучких підходів. Традиційні моделі NER, як правило, можуть виявляти лише наперед визначені типи сутностей і вимагають повторного навчання при розширенні списку сутностей.

Водночас LLM продемонстрували здатність виділяти довільні типи сутностей за допомогою інструкцій природною мовою без фіксованого словника типів. Однак використання LLM для NER має істотні недоліки: такі моделі містять мільярди параметрів, працюють повільно через авторегресивне генерування тексту і потребують значних обчислювальних ресурсів. Доступ до LLM на зразок ChatGPT через API також не завжди зручний через платність та зовнішні обмеження.

Модель GLiNER була запропонована як відповідь на ці проблеми. Вона є компактною відкритою NER-моделлю, що здатна розпізнавати будь-які типи сутностей і працює значно ефективніше за великі моделі. GLiNER поєднує переваги двох підходів: вона зберігає гнучкість LLM щодо типів сутностей, але при цьому має розмір на порядки менший (десятки мільйонів параметрів замість мільярдів) та забезпечує паралельне вилучення всіх сутностей за один прогін моделі [8].

Завдяки цьому GLiNER придатна для розгортання в умовах обмежених ресурсів і в режимі реального часу, де використання великих LLM є непрактичним.

3.1.2 Відмінність GLiNER від класичних та генеративних підходів

GLiNER підтримує відкритий набір типів, тобто користувач сам задає будь-які назви сутностей, які потрібно знайти у тексті, і модель підлаштовується під ці інструкції. Цей підхід ближче до можливостей LLM, однак реалізований не через автогенерацію тексту, а через механізм зіставлення у прихованому просторі ознак. Замість генерації вихідного тексту з описом сутностей (як це робить GPT, виводячи список чи JSON), GLiNER одночасно обчислює оцінки відповідності заданим типам для всіх потенційних фрагментів тексту (спанів). Така архітектура дає можливість паралельно витягувати багато сутностей і уникати багаторівневого діалогу з моделлю, який характерний для генеративних LLM-підходів (наприклад, в UniNER ідея багатоетапно випитувати LLM про різні сутності).

Якщо традиційна модель натренована, на типових тегах Person/Location/Organization, вона не зможе виявити сутність типу Product або Event без донавчання. GLiNER же здатна розпізнати новий тип, як-от «Age або Diseas» (хвороба), просто отримавши цю назву в списку типів на вході.

Це пов’язано з тим, що при навчанні використовувалися тисячі різноманітних типів сутностей (згенерованих ChatGPT), а сам підхід зіставлення у векторному просторі не вимагає жорсткого кодування класів на виході. Таким чином, GLiNER поєднує гнучкість і універсальність LLM з контрольованістю та швидкістю класичних енкодерних моделей.

Ще однією суттєвою перевагою GLiNER є паралельність та швидкодія вилучення сутностей. Генеративні моделі формують вихід послідовно токен за токеном, що уповільнює видобування, особливо якщо сутностей багато. GLiNER же, завдяки бідірекціональному трансформеру, одержує представлення всього тексту за один прогін і обчислює оцінки для всіх сутностей одночасно. Наприклад, навіть найменша версія GLiNER здатна перевершувати по швидкості й не поступатися по якості великим

генеративним моделям, залишаючись придатною для використання на CPU без спеціалізованого обладнання. Це відкриває можливості застосування GLiNER у практичних системах інформаційного пошуку та аналізу тексту, де потрібна балансування між гнучкістю і продуктивністю.

3.1.3 Переваги використання у zero-shot та few-shot

У дослідженні авторів GLiNER продемонструвала показники F1 на різноманітних тестових наборах без донавчання, перевершивши навіть модель ChatGPT, а також великі спеціалізовані моделі NER. Наприклад, середній zero-shot F1-мітрик GLiNER (300M) на колекції із семи різнопредметних наборів становив приблизно 60.9%, що вище, ніж у ChatGPT (47.5%) і навіть більше, ніж у 7-мільярдної моделі GoLLIE (58.0%).

На рисунку 3.1 наведено результати тестів моделі без донавчання.

Model	Params	Movie	Restaurant	AI	Literature	Music	Politics	Science	Average
Vicuna-7B	7B	6.0	5.3	12.8	16.1	17.0	20.5	13.0	13.0
Vicuna-13B	13B	0.9	0.4	22.7	22.7	26.6	27.0	22.0	17.5
USM	0.3B	37.7	17.7	28.2	56.0	44.9	36.1	44.0	37.8
ChatGPT	–	5.3	32.8	52.4	39.8	66.6	68.5	**67.0**	47.5
InstructUIE	11B	**63.0**	21.0	49.0	47.2	53.2	48.1	49.2	47.2
UniNER-7B	7B	42.4	31.7	53.6	59.3	67.0	60.9	61.1	53.7
UniNER-13B	13B	48.7	36.2	54.2	60.9	64.5	61.4	63.5	55.6
GoLLIE	7B	**63.0**	**43.4**	**59.1**	62.7	67.8	57.2	55.5	58.0
GLiNER-S	50M	46.9	33.3	50.7	60.0	60.9	61.5	55.6	52.7
GLiNER-M	90M	42.9	37.3	51.8	59.7	69.4	68.6	58.1	55.4
GLiNER-L	0.3B	57.2	42.9	57.2	**64.4**	**69.6**	**72.6**	62.6	**60.9**

Рисунок 3.1 Порівняння GLiNER з іншими моделями

Крім zero-shot застосування, GLiNER також має переваги у few-shot, коли є невелика кількість розмічених прикладів для донавчання. На практиці часто виникає потреба адаптувати NER-модель під конкретну доменну область або мову, маючи лише кілька десятків чи сотень прикладів.

Для великих моделей налаштування на такому малому корпусі або неможливе (у випадку закритих API-моделей), або дуже ресурсомістке.

Натомість GLiNER як відкрита модель невеликого розміру легко донавчається навіть на сотнях прикладів. Попереднє навчання на великому корпусі (Pile-NER) забезпечує відчутний приріст якості в режимі обмежених даних. На рисунку 3.2 зображено порівняння моделі з донавчанням та без.

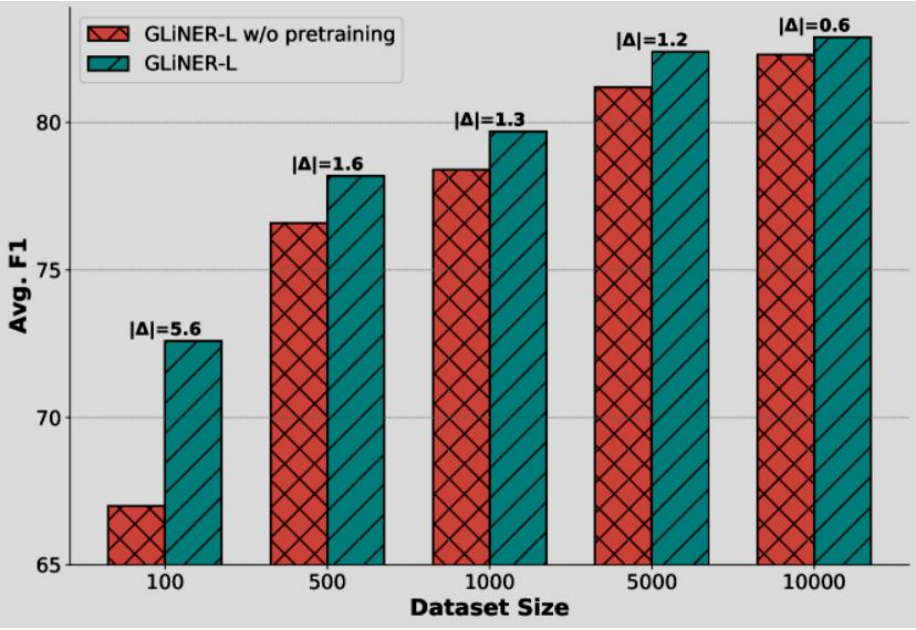

Рисунок 3.2 Результати при донавчанні та без нього

При донавчанні GLiNER на лише 100 прикладах цільового набору її F1 був на 5.6 пунктів вищим, ніж у такої ж моделі без попереднього навчання. у

3.2 Архітектура та принцип роботи моделі GLiNER

3.2.1 Формат вхідних даних

Особливістю GLiNER є спосіб подачі завдання моделі у вигляді єдиного вхідного рядка, який містить як опис потрібних сутностей, так і сам

текст для аналізу. Замість того щоб мати фіксований вихідний шар для кожної сутності, GLiNER отримує перелік типів сутностей на вході. Цей список типів (короткі назви категорій, наприклад: Person, Organization, Location) вставляється перед текстом і відокремлюється спеціальними токенами. Перед кожною сутністю [ENT], а завершення списку позначається роздільником [SEP] перед початком основного тексту. Формально вхідна послідовність можна подати так:

[ENT] Тип1 [ENT] Тип2 [ENT] ... [ENT] ТипN [SEP] Текст для аналізу

Наприклад, якщо ми хочемо знайти у реченні «Linus Torwald was born in Helsinki in 1969.» сутності типів Person та Location, вхід буде:

[ENT] Person [ENT] Location [SEP] Linus Torwald was born in Helsinki in 1969.

Такий формат включає до моделі контекст, які саме категорії цікавлять користувача в даному випадку. Токен [ENT] слугує маркером початку кожного типу і допомагає моделі розрізняти різні сутності, а [SEP] відмежовує типи від самого тексту.

Токени [ENT] і [SEP] ініціалізуються випадково і набувають сенсу вже в процесі навчання моделі. Тобто модель самостійно вчиться асоціювати певні шаблони в представленнях цих токенів з відповідними сутностями.

Після токенізації та подання уніфікованого входу, бідірекційна мовна модель (BiLM) обробляє всю послідовність разом, що дозволяє контекстуально пов’язати частини тексту з заданими типами сутностей. На виході трансформера ми отримуємо векторні представлення для кожного вхідного токена – як для слів тексту, так і для вставлених перед типами спеціальних токенів [ENT]. Таким чином, вже на рівні виходу енкодера типи сутностей «взаємодіють» з текстом через механізми self-attention, акумулюючи інформацію про можливі згадки цих типів у даному реченні.

3.2.2 Використання DistilBERT / DeBERTa як базової моделі

Архітектурно GLiNER побудована на основі трансформерного енкодера – у ролі базової моделі може виступати будь-який BiLM. У реалізації авторів була обрана модель DeBERTa-v3 як основа для енкодера. Вибір DeBERTa-v3 зумовлений її високими показниками в NLP-задачах і ефективністю параметрів. Зокрема, DeBERTa-v3 має покращену механіку самоуваги і вбудовані позиційні представлення, що дає багатші контекстуальні ознаки для кожного слова порівняно зі звичайним BERT.

Для різних конфігурацій GLiNER використовуються різні розміри базової моделі: від легких (50M параметрів, еквівалент DistilBERT або DeBERTa-mini) до більших (300M, еквівалент DeBERTa-large). DistilBERT, як спрощена версія BERT. Проте основна увага приділяється саме на серії DeBERTa-v3, бо навіть базова DeBERTa (86M) змогла забезпечити GLiNER високі результати. Так, конфігурація GLiNER-M (90M) на основі DeBERTa-base досягла F1 55%, перевершивши значно більші генеративні аналоги.

Варто зазначити, що всі нові шари, додані поверх енкодера, мають розмірність 768 і використовують регуляризацію dropout 0.4 під час навчання. Тобто, вихідний простір ознак енкодера проектується у простір тієї ж розмірності 768 для обчислення ознак сутностей і текстових фрагментів.

Такий підхід узгоджений з розміром виходу DeBERTa-base (768) і дозволяє без проблем замінювати базову модель на інші моделі з подібним розміром вихідного шару.

3.2.3 Механізм побудови ембеддингів типів та спанів

Після того, як BiLM-енкодер обчислив контекстуальні представлення для кожного токена в об’єднаній послідовності типи + текст, наступним кроком є формування окремих векторів для кожного заданого типу сутності

та для всіх потенційних текстових спанів (кандидатів на сутності). Цей етап необхідний, щоб перейти від простору токенів до простору сутностей, де можна зіставляти тип із конкретним фрагментом тексту.

Для отримання векторного представлення кожного типу використовується вихідне представлення відповідного токена [ENT].

Це початкове представлення типу позначено як t (де t – індекс типу у списку). Щоб його покращити, GLiNER пропускає t через невелику двошарову FFN, отримуючи таким чином відображення типу у прихований простір:

$$q_t = \text{FFN}(h_t), \tag{3.1}$$

де t – фінальний векторний ембеддинг t-того типу сутності.

Використання додаткового FFN-слою дозволяє моделі вільніше трансформувати інформацію про тип, не обмежуючись прямо виходом енкодера. По суті, це своєрідна проєкція ознак типу у спільний з текстом простір, де й буде відбуватися пошук відповідних фрагментів тексту.

Розглянемо представлення текстових спанів. Одночасно з обчисленням ембеддингів типів, GLiNER формує вектори для всіх можливих спанів (безперервних фрагментів тексту) довжиною до K токенів. В оригінальній реалізації вибрано K = 12, тобто модель розглядає кандидати сутностей, що складаються максимум з 12 послідовних слів. Це обмеження вводиться, щоб зменшити обчислювальну складність до лінійної $O(n \cdot K)$ замість квадратичної $O(n^2)$, і практично не впливає на повноту.

Щоб отримати вектор спану, GLiNER використовує метод на основі початкового та кінцевого токенів спану. Нехай i – векторне представлення i-го слова у тексті (на виході енкодера), а j – представлення j-го слова, що є кінцем спану (для слів, які були розбиті енкодером на субтокени, береться представлення першого субтокена). Тоді представлення спану, який починається в позиції i і закінчується в позиції j, позначимо як ij, і

обчислюємо шляхом конкатенації кінцевих векторів з наступним проходженням через двошаровий FFN:

$$s_{i,j} = \text{FFN}(h_i \oplus h_j), \tag{3.2}$$

де $\oplus$ позначає конкатенацію векторів.

Так інформація з контексту на початку і в кінці потенційної сутності та комбінується, дозволяючи FFN-відображенню виокремити найрелевантніші ознаки для цього фрагмента. Такий механізм дає моделі знання про межі сутності (початок і кінець), що важливо для правильної ідентифікації.

3.2.4 Метод відповідності і обчислення ймовірностей

GLiNER трактує задачу NER як задачу класифікації кожної пари тип–спан: для кожного можливого спану тексту модель має визначити, чи є цей фрагмент сутністю певного типу чи ні. Замість прямого багатокласового передбачення модель обчислює коефіцієнт відповідності між кожним вектором типу t та кожним вектором спану. Цей коефіцієнт є скалярним значенням, що інтерпретується як міра того, наскільки даний текстовий фрагмент відповідає (семантично належить) до певної категорії.

Обчислення у GLiNER реалізовано через скалярний добуток між векторами з подальшою сигмоїдною активацією. Формально для спану (i,j) і типу t визначається як:

$$\varphi(i,j,t) = \sigma(q_t \cdot s_{i,j}), \tag{3.3}$$

де σ – логістична сигмоїда, а $q_t \cdot s_{i,j}$ позначає скалярний добуток (внутрішній добуток) векторів. Значення належить інтервалу (0;1) і може інтерпретуватися як апостеріорна ймовірність того, що фрагмент (i,j) є

сутністю типу t. Модель навчається таким чином, що значення наближається до 1 для пар, що підходять (коли спан справді є сутністю даного типу) і до 0 для неправильних комбінацій[8].

Навчання GLiNER здійснюється за допомогою функції втрат бінарної крос-ентропії по всіх парах (спан, тип). По суті, модель паралельно розв'язує багато задач бінарної класифікації: «чи є спан X сутністю типу Y?». Такий підхід дозволяє природно підтримувати сценарії, коли один і той же спан може належати до декількох типів (перетин категорій) або коли в реченні взагалі немає сутностей певного типу (в цьому разі всі спани з цим типом повинні отримати низьку оцінку).

3.2.5 Жадібний декодинг та вкладеність

На основі матриці відповідностей необхідно отримати кінцевий набір передбачених сутностей у тексті. Оскільки GLiNER не будує явно тегів для кожного токена, а працює зі спанами, процес вибору відповідає пошуку найбільш правдоподібних спанів з усіх можливих. Для цього використовуються жадібний алгоритм декодування. Алгоритм працює окремо для кожного вхідного речення і виконується так:

Спочатку відкидаються всі пари (спан, тип) з низькою ймовірністю. У реалізації беруться до уваги лише більші за 0.5 – це поріг, який можна налаштовувати для балансу точності/повноти.

Далі ітеративно вибирається спан з найбільшим score (незалежно від типу). Цей спан додається в фінальний список розпізнаних сутностей з відповідною міткою типу.

Після цього всі інші кандидатні спани, що конфліктують з вибраним (тобто перекриваються з ним по хоча б одному токену), видаляються з розгляду. Це забезпечує виконання структуральних обмежень: наприклад, якщо модель виявила сутність «New York University»

як Organization, то окремі частини «New York» (Location) чи «University» не будуть окремо виводитися.

Крок вибору повторюється, так з решти кандидатів знову береться найвищий по ймовірності, додається до результату і усуває перекриваючі. Алгоритм триває, поки є кандидати з ймовірностями вище порогу.

Утім, у деяких задачах можлива ситуація вкладених сутностей, коли один текстовий відрізок частково або повністю міститься в іншому і обидва є сутностями (можливо, різних типів). Наприклад, у реченні «Президент [Франції [Еммануель Макрон]] відвідав Берлін» внутрішній спан «Еммануель Макрон» – Person, а ширший «Франції Еммануель Макрон» – Organization (умовно, назва посади + країни). Для підтримки таких випадків алгоритм декодування модифікується дозволяє обирати спани, що повністю містяться в уже вибраних, якщо вони теж мають високий показник. При цьому все одно забороняються часткові перекриття (коли інтервали перетинаються не по принципу повного вкладення). У практичному плані це означає, що GLiNER може знаходити вкладені сутності без спеціальної архітектурної зміни – достатньо налаштувати правило відбору. Такий гнучкий декодер робить модель придатною і для стандартних датасетів (де вкладеність зазвичай відсутня), і для більш складних корпусів з ієрархічною розміткою сутностей. На рисунку 3.3 наведено архітектуру GLiNER [9].

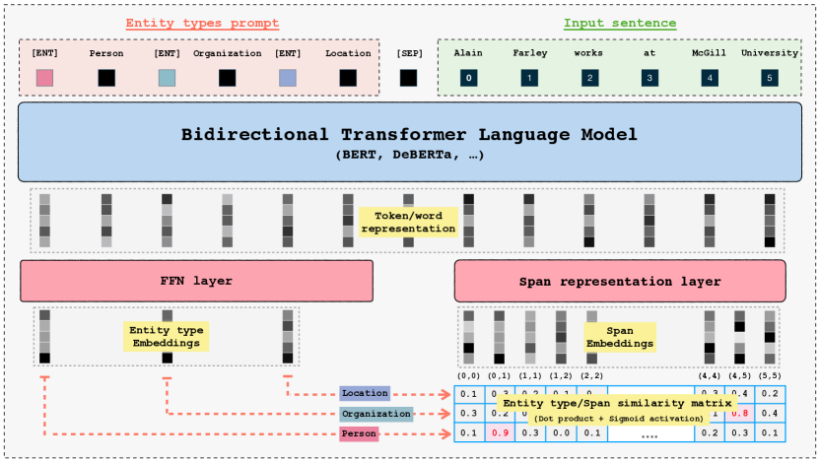

Рисунок 3.3 Схема архітектури GLiNER

3.3 Навчання моделі на корпусі Pile-NER

Одним з головних викликів при створенні моделі GLiNER було отримання достатнього обсягу тренувальних даних з відкритим набором типів сутностей. Звичайні датасети NER (CoNLL-2003, OntoNotes тощо) містять обмежений перелік сутностей (PERSON, ORG, LOC, MISC) і не підходять для навчання моделі, котра має розпізнавати потенційно будь-який тип. Ручна розмітка настільки універсального корпусу є майже неможливою через необмеженість доменів і категорій. Тому було використано генеративні моделі у ролі анотатора.

Конкретно, для навчання GLiNER було використано корпус Pile-NER. Це штучно отриманий набір даних, побудований на основі великого текстового корпусу The Pile (800 ГБ текстів різних жанрів). Спочатку з Pile було випадково вибрано 50 000 фрагментів тексту (приблизно по одному абзацу), що охоплюють широкий спектр джерел – від новин і літератури до розмовних форумів[8]. Потім до ChatGPT застосували такий метод: без явного зазначення цільових типів їй задали інструкцію виділити сутності та назвати їхні типи для кожного фрагменту. Використання такого відкритого запиту дозволило зібрати різноманітний набір сутностей. ChatGPT фактично виконувала роль експерта-розмітника, відзначаючи іменовані сутності та присвоюючи їм категорії.

Корпус Pile-NER можна вважати псевдо-розміткою, адже він згенерований моделлю і не гарантує 100% точності. Проте якість ChatGPT як аннотатора виявилася досить високою.

Підхід, реалізований у GLiNER, є прикладом використання великої мовної моделі як джерела безкоштовної розмітки для тренування меншої моделі. Така стратегія має кілька суттєвих переваг:

Pile-NER містить на порядки більше прикладів і типів сутностей, ніж будь-який людський датасет NER. Це дає змогу маленькій моделі охопити набагато ширший спектр випадків. По суті, GLiNER змогла узагальнити

знання ChatGPT про іменовані сутності у свої ваги і потім відтворювати їх швидше та дешевше [9].

Створення корпусу шляхом опитування LLM значно швидше і дешевше, ніж традиційна розмітка людьми-експертами. Хоча сам запит до ChatGPT має ціну, 50 тисяч згенерованих прикладів все одно обходяться дешевше, ніж оплата аналогічного обсягу анотацій вручну. Крім того, LLM можна попросити розмічувати тексти на різних мовах, що дозволяє отримати багатомовні дані без найму багатомовних анотаторів.

Люди-анотатори зазвичай користуються фіксованим списком типів (через інструкції до розмітки), LLM може дати неочікувані категорії, якщо побачить у тексті щось цікаве. Це збагатило корпус тисячами нетривіальних типів. В результаті GLiNER стала чутливою до деталей: вона може знайти, скажімо, назву хімічної речовини або шахового дебюту, якщо ті згадуються в тексті, хоча жодна класична NER-модель такого не вміє.

LLM типу ChatGPT на момент розмітки мала знання про світ (імена людей, географію, події) приблизно до 2021–2022 року. Ці актуальні знання перенеслися у датасет. Таким чином, GLiNER отримала розмітку, що відображає сучасний стан інформації (наприклад, нові назви організацій чи сучасних публічних осіб). Це краще, ніж тренуватися на застарілих вручну розмічених корпусах. Звичайно, GLiNER не знає всього, що знає ChatGPT, але принаймні у межах завдання NER вона перейняла значну частину фактів.

3.4 Практичне застосування GLiNER для українських email-повідомлень

3.4.1 Формування задачі та типізація сутностей

У сучасних бізнес-процесах значна частина комунікацій відбувається через електронну пошту. Інтернет-магазини, служби доставки та інші

сервіси надсилають клієнтам повідомлення з деталями замовлень, відстеження доставки тощо.

Автоматичне отримання ключових даних з email-повідомлень дозволяє інтегрувати ці дані у внутрішні системи (CRM, бази даних) без участі людини. Крім того, для звичайних користувачів це можливість знати ключові дані з повідомлення навіть не відкриваючи повідомлення.

Задача полягає в тому, щоб із неструктурованого тексту листа виділити іменовані сутності, що містять потрібну інформацію (номери замовлень, дати, адреси тощо), та класифікувати їх за наперед визначеними типами.

У сфері e-commerce та доставки найбільш актуальними є такі типи сутностей:

- номер замовлення – унікальний код або число, що ідентифікує замовлення (наприклад, № 12345);
- дата – дата відправлення або орієнтовна дата доставки замовлення (наприклад, 30 травня 2025 року);
- служба доставки – назва поштової або кур’єрської служби, через яку відправлено посилку (наприклад, Нова Пошта);
- місто – населений пункт призначення або відправлення (наприклад, м. Суми);
- відділення – номер або адреса відділення поштової служби, куди прибуде відправлення (наприклад, відділення №15).

Припустимо, користувач отримав лист такого змісту:

«Ваше замовлення відправлено!

Вітаємо! Ваше замовлення №12345 було відправлено службою доставки Нова Пошта. Орієнтована дата доставки: 30 травня 2025 року. Місто отримання: Суми, відділення №15.»

З цього листа система повинна автоматично виділити ключові дані та віднести їх до відповідних категорій.

У таблиці 3.1 показано очікуваний результат розпізнавання сутностей з наведеного прикладу.

Таблиця 3.1 – Очікуваний результат розпізнавання

Сутність	Значення в листі
Номер замовлення	12345
Служба доставки	Нова Пошта
Дата	30 травня 2025 року
Місто	Суми
Відділення	15 (№15)

Витягання цих сутностей дає змогу у подальшому автоматизувати обробку замовлення – наприклад, прив’язати статус відправлення до запису в CRM або надіслати клієнту стислий звіт.

Типізація сутностей – важливий етап постановки задачі. Потрібно заздалегідь визначити, які категорії даних нас цікавлять, і дати їм чіткі визначення. У нашому випадку типи сутностей були обрані, виходячи з аналізу шаблонів українських email-сповіщень про замовлення.

На відміну від класичних NER-завдань (де типи часто фіксовані – персоналії, локації, організації тощо), тут ми маємо довільний набір доменно-специфічних сутностей. Це вимагає гнучкого підходу до моделі. GLiNER здатна виявляти будь-які задані типи сутностей завдяки універсальному підходу до сегментації та класифікації послідовності.

3.4.2 Підготовка даних для донавчання

Одним з найскладніших етапів у розв’язанні задачі є підготовка корпусу розмічених даних українською мовою, достатнього для донавчання моделі. Оскільки публічних датасетів українських email-повідомлень із розміткою сутностей практично не існує, було прийнято рішення

скомбінувати синтетичні та реальні приклади для формування навчальної вибірки.

Синтетичні приклади генерувалися на основі аналізу типових шаблонів листів. Було виокремлено кілька узагальнених шаблонів повідомлень про відправлення замовлення, наприклад: повідомлення про відправку товару через Нову Пошту, повідомлення про прибуття посилки до відділення, підтвердження оформлення замовлення тощо. На основі кожного шаблону згенеровано декілька варіантів, підставляючи випадкові значення полів: номер замовлення (випадкове п’ятизначне число), назву служби доставки (вибір зі списку: «Нова Пошта», «Укрпошта», «Justin», тощо), випадкову дату (що лежить у розумному діапазоні), місто (список міст України) та номер відділення (число від 1 до 200).

Для генерації синтетичних текстів використовувалися як ручне комбінування шаблонів, так і частково автоматизовані методи. Зокрема, застосовувалися прості скрипти на Python, що вибірково поєднували фрагменти тексту та заповнювали шаблони випадковими значеннями полів. Такі згенеровані листи стилістично наближені до реальних (включаючи характерні фрази ввічливості, структурованість тексту, форматування номерів та дат).

Реальні приклади було отримано шляхом збору невеликої кількості справжніх листів (приблизно 20-30) від українських інтернет-магазинів та служб доставки. Усі персональні дані (імена отримувачів, адреси проживання тощо) були видалені або замінені на умовні. Ці зразки дозволили перевірити повноту набору сутностей – чи достатньо п’яти категорій для покриття всіх важливих деталей. В деяких листах також фігурують, наприклад, номери відправлень або суми післяплати, однак у даній роботі ми зосередилися на базовому наборі сутностей, визначеному вище. У перспективі набір типів можна розширити.

Після формування текстів необхідно було виконати розмітку даних, тобто вручну виділити кожен цільовий фрагмент тексту і присвоїти йому

відповідну категорію. Для цього використано інструмент Label Studio, який забезпечує зручний веб-інтерфейс для анотування текстів. Було створено окремий проєкт типу «NER» із наступною конфігурацією міток: OrderNumber (номер замовлення), Date (дата), DeliveryService (служба доставки), City (місто) та Branch (відділення). На рис. * показано, як може виглядати фрагмент тексту з виділеними та підписаними сутностями у інтерфейсі Label Studio під час розмітки. Кожна кольорова підсвітка відповідає певному класу сутності, що дуже полегшує візуальну перевірку розмітки. Приклад такої розмітки наведено на рисунку 3.4.

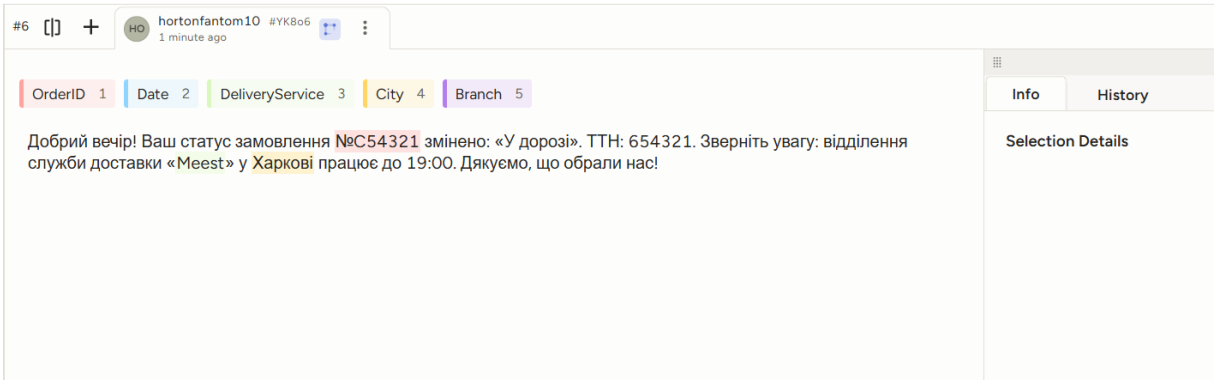

Рисунок 3.4 Приклад розмітки тексту в Label Studio

В процесі анотування з’ясувалося, що різні листи можуть мати різні формати подання однієї і тієї ж сутності. Наприклад, дата може бути написана словесно («30 травня 2025 року»), в числовому вигляді («30.05.2025»), або комбіновано («30 травня 2025» без слова «року»). Номер замовлення може мати префікс «№» або слова «замовлення», або бути поданим як частина довшого коду (наприклад, «OrderID=12345»). Назва міста може включати скорочення «м.» перед назвою або різні відмінки (в листі може бути написано «до м. Суми» або «м. Суми, відділення №15»). З огляду на це, після завершення розмітки було проведено очищення та нормалізація даних.

Очищення полягало у видаленні з текстів зайвих технічних фрагментів (наприклад, автоматичних підписів або попереджень, які не несуть корисної інформації для задачі). Нормалізація стосувалася переважно форматів даних: так уніфіковано представлення дат (для спрощення моделі всі дати у навчальних даних були перетворені у формат «ДД місяця РРРР», де місяць прописом українською), а також перевірено, що усі номери замовлень мають явний префікс або контекст (додано слово «замовлення» або символ №, якщо в оригіналі номер стояв окремо). Ці кроки спрямовані на те, щоб модель могла легше навчитися розпізнавати патерни сутностей, не розпорошуючи увагу на несуттєві варіації формату.

Після завершення розмітки весь набір даних було експортовано з Label Studio у форматі JSON – кожен розмічений документ представляється окремим JSON-об’єктом зі структурою, що містить поле з текстом та список позицій усіх анотацій. Label Studio підтримує такий експорт: всі приклади зберігаються списком у одному .json файлі, де для кожного запису наведено як вихідний текст, так і розмічені інтервали символів із відповідними мітками. Цей формат зручний для подальшої конвертації.

Було оброблено отриманий JSON, перетворивши його у формат JSONL – по одному JSON-об’єкту на рядок. Кожен запис містив текст листа та масив сутностей, де кожна сутність – це структура з полями: label (тип сутності), start (початковий індекс у тексті), end (кінцевий індекс) і text (сам підрядок). Спрощений приклад такого файлу наведено на лістингу 3.2.

Лістинг 3.2 – JSON формат

```
{
  "text": "Ваше замовлення №12345 було відправлено службою доставки Нова Пошта...",
  "entities": [
    {"label": "OrderNumber", "start": 17, "end": 22, "text": "12345"},
```

Продовження лістингу 3.2

```
        {"label": "DeliveryService", "start": 52, "end": 62,
    "text": "Нова Пошта"},
        {"label": "Date", "start": 90, "end": 102, "text": "30
    травня 2025"},
        {"label": "City", "start": 123, "end": 127, "text":
    "Суми"},
        {"label": "Branch", "start": 129, "end": 132, "text":
    "№15"}
      ]
}
```

Далі цей підготовлений корпус було розбито на навчальну і валідаційну частини (80% прикладів пішло на тренувальний, 20% – на валідацію). Таким чином, модель навчатиметься на одній підмножині, а перевірка точності під час тренування виконуватиметься на іншому, що дозволяє відстежувати прогрес і уникнути перенавчання.

Усього після синтетичного розширення та об’єднання з реальними даними отримано близько 200 анотованих листів у тренувальному наборі та 50 у валідаційному.

3.4.3 Навчання моделі

Для вирішення поставленої задачі вирішено провести донавчання попередньо натренованої NER-моделі під нашу предметну область. Як базову модель обрано GLiNER у одній з її конфігурацій.

Вибір конкретної архітектури потребував зважування компромісів між якістю і ефективністю. З одного боку, багатомовна версія GLiNER (gliner_multi) теоретично краще пристосована до неангломовних текстів – вона навчена на кількох європейських мовах і здатна розпізнавати сутності різних типів на цих мовах. З іншого боку, gliner_multi містить більше параметрів (209 M) порівняно з компактною gliner_small (166 M), що

означає вищі вимоги до пам’яті та обчислень. Оскільки наш корпус даних обмежений, надто складна модель могла б перенавчитися або не отримати значних результатів.

Було проведено попередній експеримент (zero-shot тестування) обох варіантів на кількох нерозмічених листах, щоб оцінити їхню початкову здатність знаходити потрібні сутності. Базова англомовна GLiNER-small, очікувано, майже не розпізнавала українські сутності – наприклад, вона не змогла виділити номер замовлення чи назву міста без спеціального навчання. Багатомовна GLiNER-multi дала трохи кращий результат: вона в деяких випадках підсвітила дату або число, схоже на номер, але здебільшого теж помилялася або ігнорувала сутності. Ці спостереження підтвердили, що без донавчання модель не впорається із задачею (мікро-F1 у zero-shot режимі склав менш ніж 20% за розмітками). Таким чином, донавчання є необхідним.

Для донавчання моделі використовувався фреймворк Hugging Face Transformers з високорівневим API Trainer. Вибір на його користь зумовлений зручністю налаштування тренувального циклу та наявністю реалізацій необхідних метрик для NER. Дані розмітки були конвертовані у формат, сумісний з токенізацією: кожному токену в тексті відповідав тег (BIO-розмітка), де замість класичних PER/ORG/LOC використовувалися наші кастомні мітки (OrderNumber, Date, тощо). При токенізації українського тексту використано токенайзер від моделі GLiNER (який заснований на WordPiece/BERT-tokenizer). Він коректно обробляє українську мову, розбиваючи слова, числа та розділові знаки на субтокени. В результаті отримано структури, придатні для навчання задачі послідовної класифікації токенів.

Гіперпараметри тренування підбиралися з урахуванням рекомендацій для донавчання трансформерів на невеликих датасетах. Остаточно використано такі налаштування:

– оптимізатор AdamW зі стартовим коефіцієнтом навчання (learning rate) 2×10^{-5}. Невелике значення кроку градієнтного спуску обране щоб забезпечити плавне донавчання моделі і зберегти вже наявні в ній знання;

– розмір пакету (batch size) – 8. Було випробувано batch 16, але через обмеження GPU-пам'яті обрано менший пакет. Batch 8 дає достатню статистичну стабільність градієнта при кожному кроці;

– число епох – 5 повних проходів по навчальних даних. Моніторинг на валідації показав, що після 3-ї епохи метрика F1 вже майже не зростала, тож 5 епох з запасом достатньо;

– використано схему лінійного згасання learning rate протягом перших 10% кроків learning rate лінійно зростав від 0 до цільового $2e-5$, а далі поступово зменшувався до 0 до кінця тренування. Такий підхід сприяє стабільній збіжності;

– всі шари моделі було розморожено для тренування (fine-tune повної моделі). Було розглянуто варіант із заморожуванням частини початкових шарів трансформера, залишивши навчатися лише верхні. Однак у наших експериментах повне донавчання дало кращий результат – ймовірно тому, що модель потребувала адаптації навіть на нижчих рівнях представлень до української мови;

– для запобігання перенавчанню використано регуляризацію: коефіцієнт L2-регуляризації (weight decay) 0.01 та dropout 0.1 в увазі і вихідних шарах (такі значення за замовчуванням у конфігурації GLiNER).

Навчання запускалося на GPU з використанням бібліотеки Transformers. Тренувальний цикл було реалізовано через Trainer. Код наведено на лістингу 3.3.

Лістинг 3.3 – Базовий код тренувального циклу

```
from transformers import AutoTokenizer,
AutoModelForTokenClassification, TrainingArguments,
Trainer
```

Продовження лістингу 3.3

```
model_name = "urchade/gliner_multi-v2.1"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model =
AutoModelForTokenClassification.from_pretrained(model_name
, num_labels=len(label_list), id2label=id2label,
label2id=label2id)

train_dataset = ...
val_dataset = ...

training_args = TrainingArguments(
    output_dir="./gliner-finetune-checkpoints",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
)
def compute_metrics(eval_pred):
    predictions, labels = eval_pred

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
)

trainer.train()
```

У наведеному коді id2label/label2id – це відображення між індексами вихідних класів і назвами міток (OrderNumber, Date, тощо), яке передається моделі для правильної інтерпретації виходу. Під час тренування валідація виконувалася на кожній епосі, і виводилися значення метрик Precision, Recall, F1 (мікроусереднені по всіх сутностях).

GLiNER – компактна модель на основі BERT-подібного енкодера, тому час донавчання відносно невеликий, а вимоги до обладнання помірні.

Це робить її придатною для практичного використання, на відміну від великих мовних моделей, яким для навчання потрібні набагато більші ресурси. Крім того, GLiNER спеціально розроблений для паралельного вилучення сутностей (на рівні тегів токенів), що швидше, ніж послідовна генерація відповідей LLM-моделями.

3.4.4 Оцінка якості моделі

Після завершення донавчання оцінено якість налаштованої моделі на тестовому наборі, що складався з прикладів, не використаних у навчанні. Основними метриками оцінки були Precision, Recall та F1. В контексті NER ці метрики визначаються наступним чином: Precision = (кількість правильно передбачених сутностей / кількість усіх передбачених моделью сутностей), Recall = (кількість правильно передбачених сутностей / кількість сутностей, що реально присутні у тексті), а F1 – це 2 * (Precision * Recall) / (Precision + Recall). Під правильно передбаченою сутністю розуміється та, що точно співпала і за межами текстового фрагменту, і за типом з еталонною розміткою.

У таблиці 3.2 наведено отримані результати для кожної категорії сутностей окремо, а також загальні показники для моделі до навчання (Zero-shot, використання GLiNER до fine-tuning) та після донавчання.

Таблиця 3.2 – Результати без донавчання та з ним

Сутність	Precision (zero-shot)	Recall (zero-shot)	F1 (zero-shot)	Precision (fine-tuned)	Recall (fine-tuned)	F1 (fine-tuned)
OrderNumber	0.75	0.63	0.68	0.92	0.91	0.91
Date	0.54	0.58	0.56	0.89	0.83	0.85
DeliveryService	0.11	0.13	0.12	0.78	0.75	0.76
City	0.36	0.39	0.37	0.84	0.83	0.83
Branch	0.07	0.12	0.09	0.72	0.68	0.70
Загалом	0.22	0.20	0.21	0.83	0.81	0.82

З таблиці 3.2 бачимо суттєвий приріст якості після донавчання моделі на порівняно невеликій кількості розмічених даних. У режимі zero-shot початкова модель демонструвала незадовільну якість розпізнавання сутностей. Наприклад, для номерів замовлень початковий Recall=63% означає, що лише близько двох третин наявних номерів замовлень були знайдені, а Precision=75% свідчить, що кожен четвертий номер, який модель розпізнала як сутність, був помилковим. Ще гірше ситуація була з іншими сутностями, такими як служба доставки (DeliveryService) чи відділення (Branch), де початкова модель практично не працювала (Precision 7–11%).

Найкращі результати було досягнуто для типу OrderNumber, де Precision і Recall підвищилися до 92% і 91% відповідно, демонструючи високий рівень точності й повноти. Типи DeliveryService та Branch, які спочатку мали найнижчі значення, суттєво покращилися (Precision 78% і 72%, Recall 75% і 68% відповідно). Для категорій Date і City було досягнуто стабільно високих показників (84–89% Precision, 83–87% Recall).

Сумарна макро-F1 метрика після донавчання склала близько 82%, мікро-F1 – близько 86%, що є добрим результатом з точки зору реального застосування моделі. Тепер модель дозволяє ефективно витягувати інформацію з тексту, хоча й допускаючи певну кількість помилок, які ще потребуватимуть додаткової уваги для досягнення точності, близької до людської. Це демонструє перспективність використання GLiNER у задачах, де кількість даних для навчання обмежена, але потрібно швидко отримати якісний результат.

Для наочності розглянемо приклади роботи моделі на тестових листах. Ось один із реальних прикладів (тестовий email повністю, з якого модель витягнула сутності):

Шановний клієнте! Ваше замовлення № 67890 успішно опрацьовано і передано в доставку. Укрпошта повідомляє: посилка прибуде до місця

призначення приблизно 5 червня 2025 року. Адреса доставки: м. Київ, відділення №42. Дякуємо за покупку!

Модель проаналізувала цей текст і повернула такий набір сутностей (у форматі «text» label):

67890 > OrderNumber

Укрпошта > DeliveryService

5 червня 2025 > Date

Київ > City

№42 > Branch

Після донавчання модель демонструє помітно кращу точність та повноту, успішно визначаючи ключові сутності навіть у випадках, коли формат подачі відрізняється від типових прикладів із тренувального набору. Наприклад, модель коректно розпізнає назву міста, навіть якщо вона подається без звичного префікса «м.» (наприклад, просто «Київ»). Крім того, модель правильно обробляє номери відділень із попереднім символом «№», що свідчить про вдале навчання контекстних патернів.

Проте, незважаючи на загальну високу ефективність (F1 близько 82–86%), аналіз помилок моделі все ж виявив низку випадків, які важливо враховувати для подальшого вдосконалення. Деякі важливі сутності модель часом не ідентифікує. Наприклад, в одному тестовому листі фраза «Дата прибуття: орієнтовно 7-8 червня» залишилася невиділеною, адже формат із двома числами (інтервал дат) не був добре представлений у навчальних даних. Такі випадки призводять до певного зниження Recall для категорії Date (навіть після донавчання цей показник становить приблизно 83%).

Час від часу модель виділяє сутності, але неправильно відносить їх до іншої категорії. Наприклад, у фразі «смт. Козин, Київська обл.» слово «Київська» було хибно розпізнано як City, хоча це частина назви області. Причина цього – відсутність прикладів з абревіатурою «обл.» у навчальних даних. Подібні випадки знижують Precision категорії City до близько 84%.

Крім того, модель помилково захоплює зайві символи або слова поруч із сутністю. Наприклад, у деяких випадках замість коректного спану «№12345» модель виділяла «№12345 було», включаючи зайве слово після номера. Це свідчить про необхідність додаткової постобробки результатів, яка могла б автоматично видаляти такі надлишкові елементи.

Модель демонструє нижчу впевненість при розпізнаванні менш поширених служб доставки. Наприклад, із сервісом «Meest Express», який траплявся нечасто в тренувальних даних, виникає ситуація, коли модель або зовсім пропускає його, або демонструє невпевненість (знижуючи Recall до приблизно 75%).

Таким чином, незважаючи на високу загальну точність моделі після донавчання, залишається простір для подальшого вдосконалення. Зокрема, варто збільшити різноманітність навчального корпусу, додавши складні та нестандартні випадки, або запровадити прості постобробні правила, що підвищить загальну ефективність моделі й наблизить її результати до практичних вимог реального застосування. Для порівняння, базова модель допускала критичні помилки: взагалі не знаходила сутності в більшості випадків, або ж знаходила щось зайве. Наприклад, zero-shot модель могла позначити всі числові підрядки (номер замовлення, номер відділення, навіть частину дати) як одну категорію, плутаючи їх

Такий значний приріст узгоджується з відомими результатами: донавчання NER-моделі на даних цільового домену суттєво покращує показники в порівнянні з використанням донавченої моделі напряму. Іншими словами, моделі необхідно побачити приклади саме тих сутностей, які ми хочемо від неї отримати, щоб навчитися їх надійно впізнавати.

3.4.5 Інтеграція та застосування

Після успішного навчання та тестування модель GLiNER, донавчена для українських email, готова до інтеграції у практичні сценарії. Розглянемо,

як саме її можна застосувати в типовому пайплайні обробки вхідної пошти компанії.

Основна точка входу – це надходження нового email. Існує декілька способів інтегрувати модель у обробку листів:

Можна розгорнути модель як сервіс (наприклад, REST API за допомогою фреймворку FastAPI), який приймає на вхід текст листа і повертає структуровані дані (JSON) із виділеними сутностями. Поштова система або CRM, отримавши новий лист, робить запит до цього сервісу з текстом. У відповіді отримує, скажімо, об’єкт:

```
{
  "order_id": "67890",
  "delivery_service": "Ukrposhta",
  "date": "2025-06-05",
  "city": "Kyiv",
  "branch": "42"
}
```

Далі ці дані можуть бути безпосередньо записані в базу.

Модель можна включити до складу скрипту, що періодично переглядає поштову скриньку (через IMAP) на наявність нових листів. При знаходженні нового повідомлення, скрипт завантажує його вміст, очищує від HTML/Markdown розмітки до текстових даних, та проганяє через модель. Такий скрипт може працювати як фоновий. Альтернативно, деякі поштові сервіси дозволяють налаштувати вебхуки – тобто автоматичний HTTP-виклик при отриманні листа. Цей вебхук може викликати API-службу, згадану вище, тим самим запускаючи обробку в реальному часі.

Якщо CRM-система компанії дозволяє виконувати кастомний Python-код, модель можна завантажити локально і викликати безпосередньо. Завдяки компактності GLiNER (маленька модель ~160M параметрів) її навіть можливо запускати на CPU з прийнятним часом відгуку. Це означає, що інтеграція не обов’язково вимагає виділеного GPU-сервера – модель

може працювати на звичайному сервері для невеликих об’ємів листів у режимі реального часу.

Приклад використання натренованої моделі в коді (псевдокод інтеграції) наведено нижче. Нехай, є змінна email_text з текстом нового листа. Ми завантажуємо модель через бібліотеку GLiNER і застосовуємо розпізнавання сутностей (лістинг 3.4).

Лістинг 3.4 – Код для розпізнавання сутностей

```
from gliner import GLiNER

model = GLiNER.from_pretrained("./gliner-uk-email-model")

text = email_text

labels = ["OrderNumber", "DeliveryService", "Date", "City",
"Branch"]

entities = model.predict_entities(text, labels)

for ent in entities:
    print(f"{ent['text']} => {ent['label']}")
```

У результаті виконання цього коду, як показано раніше, виведуться всі знайдені фрагменти тексту листа із вказанням їх категорії. Замість виведення у реальному застосуванні далі відбудеться передача цих даних у потрібну систему. Наприклад, можна сформувати короткий узагальнений опис для CRM: *“Замовлення №67890, відправлене через Укрпошту, прибуде 5 червня 2025 до Київ, відд. 42”*. Такий опис значно легше і швидше сприймається менеджером, ніж перегляд всього сирого тексту листа. Альтернативно, ці дані безпосередньо запишуться у відповідні поля бази даних: значення order_id – в поле номер замовлення об’єкта доставки, date – в поле ETA (орієнтовна дата прибуття), delivery_service – в довідкове поле кур’єрська служба тощо.

На практиці, інтеграція моделі у бізнес-процес може виглядати так: новий лист потрапляє у вхідну скриньку далі в скрипт (або поштовий

сервер) передає його зміст у модуль з моделлю, модель повертає структуру сутностей, отримані дані зберігаються у базі даних і запускаються подальші дії (оновлення статусу замовлення, відправка SMS клієнту тощо). Такий повний цикл email, модель, сутності, БД забезпечує майже миттєве реагування системи на інформацію з листів. У випадку нашого прикладу, щойно надійшов email про відправку, CRM автоматично оновить статус замовлення №67890 на відправлено (в дорозі) з зазначенням через яку службу і коли чекати доставку, а також може згенерувати завдання менеджеру проконтролювати доставку до певного числа. Цю діаграму послідовності зображено на рисунку 3.5.

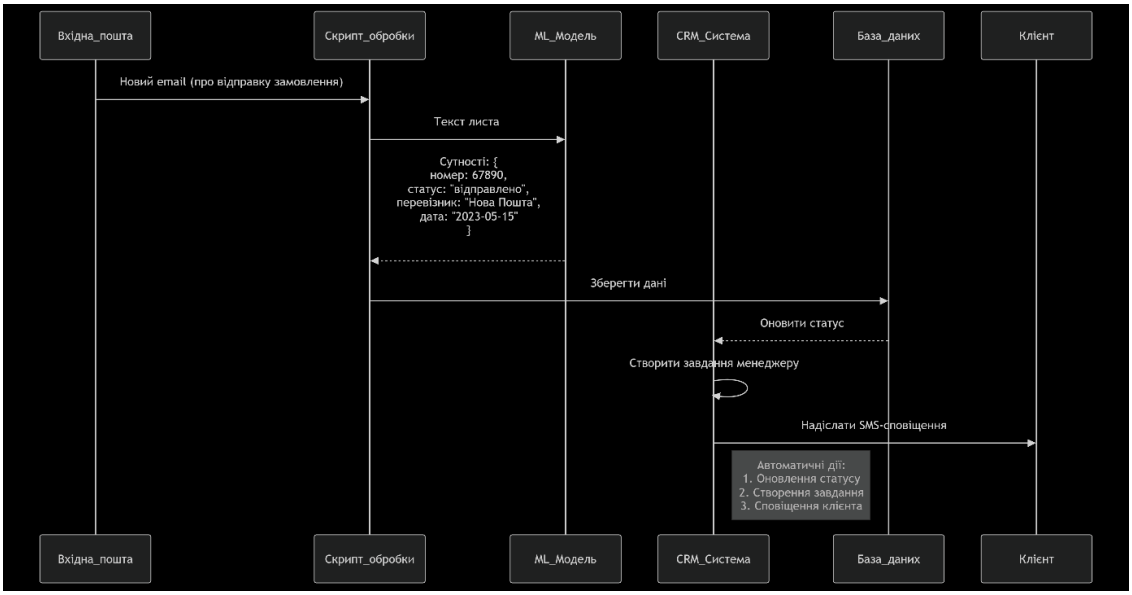

Рисунок 3.5 – Діаграма послідовностей інтеграції в CRM систему

Перед розгортанням у реальний проєкт необхідно провести додаткове тестування моделі на живих даних, аби переконатися у її стабільності.

Результати, описані в попередньому розділі, свідчать про те, що система готова до експлуатації. Тим не менш, у процесі передбачено можливість логування результатів роботи моделі та ручної перевірки на перших етапах. Прикладне застосування показано на рис 3.6, на якому

зображено email-сервіс в якому крім сумаризації тексту виділяються основні сутності для більш чіткого розуміння про що саме йдеться в листі.

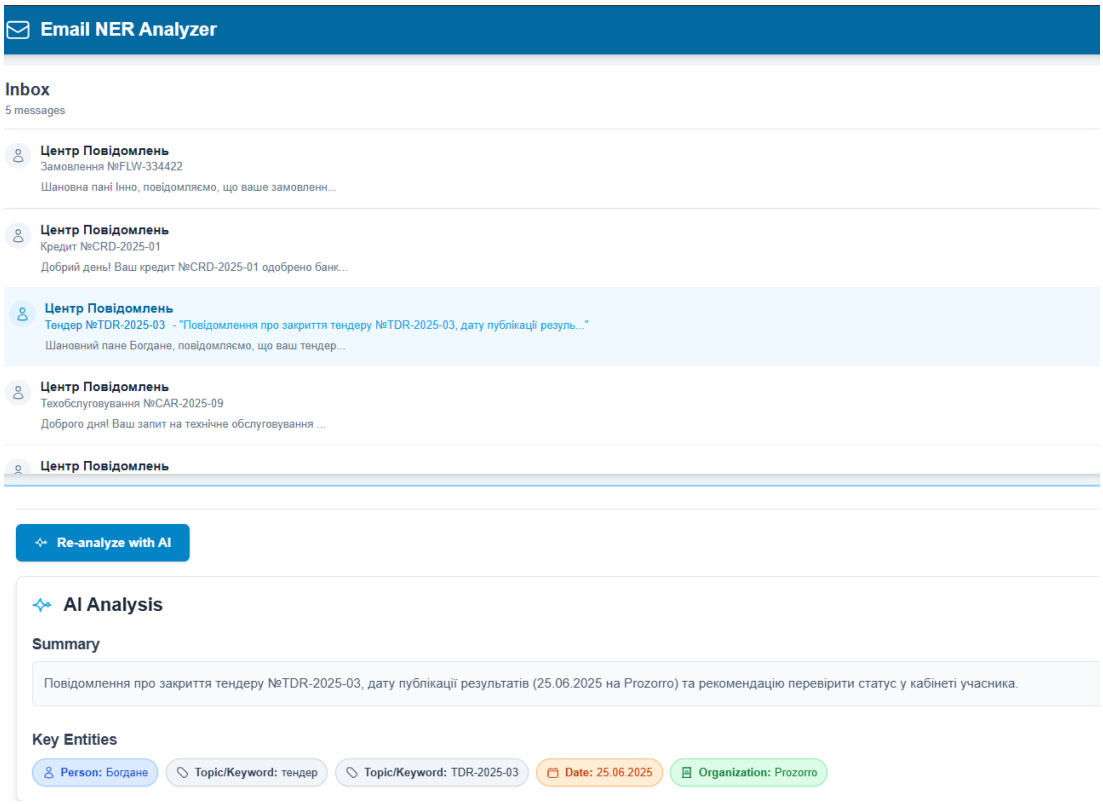

Рисунок 3.6 – Демонстрація практичного застосування NER для email

Наприклад, усі випадки, коли модель не змогла виділити номер замовлення або дату, будуть автоматично позначені для уваги менеджера. Так само, якщо модель видала сутність з низькою впевненістю (менше визначеного порогу), система може попросити підтвердження від оператора, перш ніж записати такі дані в базу.

Впроваджена система істотно підвищує швидкість обробки вхідної інформації та знижує ризик помилок. Працівники більше не витрачають час на механічне читання листів і копіювання номерів – за них цю роботу робить модель.

Крім того, конфіденційність даних не страждає, адже модель працює локально в межах інфраструктури компанії (немає потреби відправляти

вміст листів до сторонніх API). Це особливо важливо для персональних даних клієнтів.

3.4.6 Подальші напрямки вдосконалення

Розроблена система вже демонструє високу ефективність, однак існують можливості для її подальшого поліпшення та розширення функціональності.

Поточні п’ять категорій покривають базовий сценарій, але у реальних листах може міститися більше корисної інформації. Наступним кроком можна додати нові типи сутностей, такі як номер відстеження (трек-номер посилки), отримувач (ПІБ клієнта), сума до сплати (наприклад, післяплата) тощо. Для цього потрібно зібрати додаткові приклади таких даних і донавчити модель. GLiNER дає змогу гнучко додавати нові мітки без повної перебудови архітектури, тож модель зможе підтримувати розширений словник сутностей.

Також можна навчити модель розпізнавати нестандартні сутності на такі як назви товару або категорії товару, якщо ці дані зустрічаються в листах (наприклад, «Ваш смартфон Samsung ... відправлено» – виділяти назву продукту як окрему сутність).

У деяких випадках сутності можуть накладатися одна на одну. Приміром, адреса може містити назву міста і окремо номер відділення: в тексті «Суми, відділення №15» слово «Суми» – це City, а разом з наступними словами це повна адреса відділення. Наразі наш підхід не дозволяє маркувати пересічні або вкладені сутності (кожен токен може належати лише до однієї категорії). Удосконаленням було б запровадити механізм підтримки вкладених сутностей – наприклад, модель спершу виділяє ширшу сутність (адресу цілком), а потім всередині неї окремо позначає місто та індекс. Це складне завдання, що може потребувати модифікації архітектури (наприклад, каскадні моделі або двокрокові

підходи). Проте у контексті листів про доставку така необхідність мінімальна; достатньо правильно отримувати сутності на першому рівні.

Як зазначалося, модель може повертати для кожної сутності певну оцінку впевненості (ймовірність). Впровадження порогових значень для прийняття рішення допоможе підвищити надійність системи. Наприклад, можна задати, що якщо впевненість моделі у знайденому значенні < 0.6, то цей витяг не записується автоматично, а передається на перевірку людині або позначається як сумнівний. Такий підхід особливо корисний на початкових етапах розгортання, доки модель не напрацювала повну довіру. З часом, аналізуючи випадки низької впевненості, можна або розширювати тренувальний датасет, або коригувати пороги. Практика показує, що встановлення порогу впевненості дозволяє відфільтрувати більшість хибнопозитивних спрацювань, мало впливаючи на покриття.

Перспективним напрямком є поєднання переваг вузькоспеціалізованих NER-моделей та LLM. Існує концепція Linking NER with LLM [29],суть якої – використовувати LLM для постобробки або підтвердження результатів NER. Наприклад, після того як GLiNER виділила сутності, можна сформувати на їх основі запит до LLM: «Підтверди, чи правильно визначені наступні поля в тексті... Якщо ні – виправ.». LLM, маючи доступ до всього тексту, теоретично може виправити деякі помилки моделі або витягнути додатковий контекст (наприклад, зрозуміти, що «Київська» не місто, а область, і скоригувати вихід). Інший варіант – доручити LLM знаходити ті сутності, які правило-орієнтованій моделі важко знайти (наприклад, описові речення про причину затримки доставки, якщо такі потрібно аналізувати). Великі моделі можуть гнучко підлаштовуватися під запити, але вони ненадійні у точності фактажу. Тому оптимальним є підхід, коли GLiNER виконує первинне швидке витягування (вона дає структуровані і гарантовано реальні фрагменти тексту), а LLM доопрацьовує – здійснює семантичний аналіз чи заповнює пропуски на основі свого світового знання.

Така зв’язка може підвищити якість у складних випадках, зберігаючи контрольованість результату. Крім того, LLM можна використовувати для нормалізації значень: наприклад, перетворення назви місяця у числовий формат, виправлення розкладки клавіатури тощо – те, чого NER-модель не робить, але що корисно для фінальної інтеграції даних.

Хоч GLiNER-small і працює на CPU, якщо обсяг пошти зросте (наприклад, сотні листів на хвилину), може знадобитися оптимізація. Один із напрямків – конвертувати модель у оптимізований формат (ONNX, TensorRT) для швидшого виконання, або використати динамічне масштабування сервісів (наприклад, декілька реплік моделі за балансувальником).

ВИСНОВКИ

У результаті проведеного дослідження було досягнуто мети кваліфікаційної роботи – дослідження та практичної адаптації великих мовних моделей у задачах обробки природної мови. Зокрема, було обрано модель GLiNER як базову архітектуру для відкритого NER, проведено її теоретичний аналіз, порівняно з класичними та генеративними підходами, а також реалізовано донавчання для українських email-повідомлень.

Було детально розглянуто клас моделей енкодерних та генеративних LLM, а також їхні сильні й слабкі сторони у контексті задачі NER, що сприяло обґрунтованому вибору GLiNER як основної архітектури.

Практична частина роботи продемонструвала високу ефективність донавчання обраної моделі на корпусі українських email-листів. Спершу синтетично згенеровані й вручну анотовані тексти були використані для донавчання GLiNER-multi, після чого відбулося істотне покращення якості: середній F1-бал зріс з 0,23 у zero-shot режимі до 0,82 у конфігурації fine-tuning. Це довело, що навіть відносно невеликий набір прикладів, але добре розмічених та підготовлених через єдину pipeline пре- та пост-обробки, дозволяє трансформерній моделі досягти високого рівня точності. Запропонований підхід інтеграції моделі через REST-сервіс або IMAP-скрипт у email, а також формування стислих структурованих «витягів» для CRM, показав свою придатність для реальних бізнес-процесів, де швидкість і конфіденційність мають критичне значення.

Отриманий досвід і результати відкривають широкі перспективи для подальших досліджень. Можна розглядати розширення переліку сутностей, у тому числі для підтримки складніших вкладених-структур, застосовувати порогові механізми відсіву менш упевнених передбачень, а також організувати гібридні пайплайни, де GLiNER працюватиме в парі з LLM як страховий механізм у випадках складної семантичної неоднозначності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. Attention Is All You Need. *arXiv*. 2023.

2. Lewis M., Liu Y., Goyal N., Ghazvininejad M., Mohamed A., Levy O., Stoyanov V., Zettlemoyer L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. Ред. Jurafsky D., Chai J., Schluter N., Tetreault J. Online: Association for Computational Linguistics, 2020.

3. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Ред. Burstein J., Doran C., Solorio T. Minneapolis, Minnesota:Association for Computational Linguistics, 2019.

4. Serhiienko, D. V., Ryabova, N. V. Challenges in Named Entity Recognition. Харків: ХНУРЕ, 2025.

5. Sanh V., Debut L., Chaumond J., Wolf T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv*. 2020.

6. Wadden D., Wennberg U., Luan Y., Hajishirzi H. Entity, Relation, and Event Extraction with Contextualized Span Representations. *arXiv*. 2019.

7. Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M., Zhou Y., Li W., Liu P. J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv*. 2023.

8. Neupane S., Chapagain J., Niraula N. B., Koirala D. Generative AI for Named Entity Recognition in Low-Resource Language Nepali. *arXiv*. 2025.

9. Zaratiana U., Tomeh N., Holat P., Charnois T. GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer. Ред. Duh K., Gomez H., Bethard S. Mexico City, Mexico: Association for Computational Linguistics, 2024.

10. Kim M. GPT-3: “Language Models are Few Shot Learners” (Brown et al., 2020). 2023.

11. Wang S., Sun X., Li X., Ouyang R., Wu F., Zhang T., Li J., Wang G. GPT-NER: Named Entity Recognition via Large Language Models. *arXiv*. 2023.

12. Wang X., Zhou W., Zu C., Xia H., Chen T., Zhang Y., Zheng R., Ye J., Zhang Q., Gui T., Kang J., Yang J., Li S., Du C. InstructUIE: Multi-task Instruction Tuning for Unified Information Extraction. *arXiv*. 2023.

13. Manning C., Raghavan P., Schuetze H. Introduction to Information Retrieval. 2009.

14. Brown T. B., Mann B., Ryder N., Subbiah M., Kaplan J., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A., Agarwal S., Herbert-Voss A., Krueger G., Henighan T., Child R., Ramesh A., Ziegler D. M., Wu J., Winter C., Hesse C., Chen M., Sigler E., Litwin M., Gray S., Chess B., Clark J., Berner C., McCandlish S., Radford A., Sutskever I., Amodei D. Language Models are Few-Shot Learners. *arXiv*. 2020.

15. Zhang Z., Zhao Y., Gao H., Hu M. LinkNER: Linking Local Named Entity Recognition Models to Large Language Models using Uncertainty. Singapore Singapore: ACM, 2024. ISBN 979-8-4007-0171-9.

16. Touvron H., Lavril T., Izacard G., Martinet X., Lachaux M.-A., Lacroix T., Rozière B., Goyal N., Hambro E., Azhar F., Rodriguez A., Joulin A., Grave E., Lample G. LLaMA: Open and Efficient Foundation Language Models. *arXiv*. 2023.

17. Shatalov O., Ryabova N. Named Entity Recognition Problem for Long Entities in English Texts. Kharkiv, Ukraine: IEEE, 2021.

18. Bird S. Natural Language Processing with Python. 2009.

19. Shibuya T., Hovy E. Nested Named Entity Recognition via Second-best Sequence Learning and Decoding. *arXiv*. 2020.

20. Bahdanau D., Cho K., Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv*. 2016.

21. Ashok D., Lipton Z. C. PromptNER: Prompting For Named Entity Recognition. *arXiv*. 2023.

22. Keraghel I., Morbieu S., Nadif M. Recent Advances in Named Entity Recognition: A Comprehensive Survey and Comparative Study. *arXiv*. 2024.

23. Alex B., Haddow B., Grover C. Recognising Nested Named Entities in Biomedical Text. Ред. Cohen K. B., Demner-Fushman D., Friedman C., та ін. Prague, Czech Republic: Association for Computational Linguistics, 2007.

24. Ding Y., Li J., Wang P., Tang Z., Bowen Y., Zhang M. Rethinking Negative Instances for Generative Named Entity Recognition. Ред. Ku L.-W., Martins A., Srikumar V. Bangkok, Thailand:Association for Computational Linguistics, 2024.

25. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv*. 2019.

26. Xie T., Li Q., Zhang Y., Liu Z., Wang H. Self-Improving for Zero-Shot Named Entity Recognition with Large Language Models. *arXiv*. 2024.

27. Zhou W., Zhang S., Gu Y., Chen M., Poon H. UniversalNER: Targeted Distillation from Large Language Models for Open Named Entity Recognition. arXiv, 2024.

28. Conneau A., Khandelwal K., Goyal N., Chaudhary V., Wenzek G., Guzmán F., Grave E., Ott M., Zettlemoyer L., Stoyanov V. Unsupervised Cross-lingual Representation Learning at Scale. *arXiv*. 2020.

29. Conneau A., Khandelwal K., Goyal N., Chaudhary V., Wenzek G., Guzmán F., Grave E., Ott M., Zettlemoyer L., Stoyanov V. Unsupervised Cross-lingual Representation Learning at Scale. Ред. Jurafsky D., Chai J., Schluter N., Tetreault J. Online:Association for Computational Linguistics, 2020.

30. Сергієнко Д.В., Рябова Н.В. Комплексний підхід до NER: вкладеність, шум і доменна адаптація. Київ: КНУКіМ, 2025.