УДК 004.89:617.3

В.Ю. СТАРЖИНСЬКИЙ, О.В. БІСІКАЛО

МЕТОД АВТОМАТИЗОВАНОЇ СТАНДАРТИЗАЦІЇ НАЗВ МЕТИЗІВ НА ОСНОВІ LLM-МОДЕЛІ

Вінницький національний технічний університет, Хмельницьке шосе 95, 21021, Вінниця, Україна, e-mail: 3372292@gmail.com

Анотація. У статті представлено метод автоматизованої стандартизації неструктурованих технічних назв метизної продукції на основі великих мовних моделей (LLM). Розглянуто архітектуру системи, що базується на локальному інференсі моделі Mistral-7B через сервер LM Studio, що забезпечує конфіденційність промислових даних. Проведено порівняльний аналіз методу «Instructor» із використанням Pydantic-валідації та авторського методу прямої JSON-серіалізації на основі Few-Shot Prompting. Результати експерименту демонструють, що прецизійне налаштування промптів та контекстне навчання дозволяють досягти 100% точності у формуванні назв згідно з міжнародними стандартами DIN/ISO та ДСТУ. Запропоноване рішення автоматизує процеси оновлення баз даних SQLite3, мінімізує «людський фактор» та забезпечує коректну багатомовну локалізацію технічної номенклатури.

Ключові слова: великі мовні моделі (LLM), стандартизація даних, метизи, інженерія промптів (Prompt Engineering), Few-Shot Learning, автоматизація баз даних.

Abstract. The article presents a method for the automated standardization of unstructured technical names for fasteners using Large Language Models (LLMs). The system architecture is based on the local inference of the Mistral-7B model via the LM Studio server, ensuring the confidentiality of industrial data. A comparative analysis is conducted between the «Instructor» method utilizing Pydantic validation and a proprietary direct JSON serialization method based on Few-Shot Prompting. Experimental results demonstrate that precise prompt engineering and in-context learning achieve 100% accuracy in generating names aligned with international DIN/ISO and DSTU standards. The proposed solution automates the updating of SQLite3 databases, minimizes human error, and provides correct multilingual localization of technical nomenclature. This approach significantly enhances data quality and operational efficiency within supply chain management systems.

Keywords: Large Language Models (LLM), data standardization, fasteners, Prompt Engineering, Few-Shot Learning, database automation.

DOI: 10.31649/1681-7893-2026-51-1-33-40

ВСТУП

Цифрова трансформація промислового сектора та перехід до концепції "Індустрія 4.0" висувають жорсткі вимоги до якості та структурованості даних, що використовуються в інформаційних системах підприємств (ERP, PDM, PLM). Одним із найбільш критичних та водночас проблемних сегментів є ведення довідників товарно-матеріальних цінностей, зокрема металевих виробів (метизів). Метизна продукція, яка включає болти, гвинти, гайки, шайби та інші елементи кріплення, характеризується величезною номенклатурою, що нараховує десятки тисяч найменувань. Проблема полягає в тому, що назви цих виробів у базах даних постачальників, виробників та споживачів часто формуються в довільній формі, з використанням різних скорочень, мовних варіацій та відхилень від державних чи міжнародних стандартів (ДСТУ, DIN, ISO).

Важливість стандартизації назв технічних виробів важко переоцінити. Відсутність єдиного підходу до іменування призводить до дублювання позицій у складських програмах, помилок при закупівлях та неможливості автоматизації аналітичних процесів. Як зазначають дослідники, некоректні майстер-дані можуть збільшувати операційні витрати підприємства на 10-15% через неефективну логістику та надлишкові запаси [1]. Стандартизація є фундаментом для створення "єдиного джерела істини" (Single Source of Truth), що дозволяє інтегрувати системи проектування з виробничими циклами без втрати інформації.

Процес стандартизації технічної документації та найменувань традиційно базувався на використанні жорстких алгоритмів пошуку за ключовими словами або регулярних виразів (Regex). Проте такі методи демонструють низьку ефективність при роботі з природною мовою, де один і той самий об'єкт може бути описаний по різному згідно стандарту назви. Традиційні методи вимагають створення величезної кількості словників та правил, які складно підтримувати в актуальному стані при зміні асортименту.

Поява та стрімкий розвиток великих мовних моделей (Large Language Models, LLM) відкрили нову еру в автоматизації обробки технічних текстів. На відміну від класичних методів, LLM володіють семантичним розумінням контексту, що дозволяє їм ідентифікувати технічні параметри виробу (діаметр, довжину, клас міцності, тип покриття) навіть у неструктурованому тексті з великою кількістю помилок чи нестандартних скорочень. Застосування архітектур типу Transformer дозволяє ефективно вирішувати задачу Entity Extraction (видобування сутностей) та приводити їх до заданого канонічного вигляду згідно з ДСТУ чи внутрішніми регламентами підприємства [2].

Використання LLM-моделей для автоматизованої стандартизації назв метизів дозволяє мінімізувати "людський фактор", який є головним джерелом помилок при ручному введенні даних. Це створює передумови для повної автоматизації процесу оприбуткування товарів та синхронізації каталогів між різними суб'єктами господарювання. Отже, розробка методів, що поєднують глибокі знання предметної області (технічні стандарти кріплення) та потужність нейромережевих моделей, є стратегічно важливим завданням для сучасної прикладної науки.

1. ОГЛЯД МЕТОДІВ ОБРОБКИ ДАНИХ, МЕТОД «INSTRUCTOR»

Методи обробки даних є фундаментом сучасної цифрової економіки, оскільки саме вони перетворюють сирі, хаотичні масиви інформації на структурований актив, придатний для прийняття управлінських рішень. У контексті промислового виробництва та логістики, якість цих методів безпосередньо корелює з операційною ефективністю: некоректна обробка хоча б одного атрибута технічного виробу може призвести до зупинки конвеєра або помилкових закупівель на мільйони гривень. Ефективні алгоритми стандартизації дозволяють не лише усунути дублювання та помилки, а й забезпечують інтероперабельність — здатність різних інформаційних систем (від складського обліку до систем проектування) «розуміти» одна одну без перекладачів. Таким чином, методи обробки стають не просто технічним етапом, а критичним інструментом оптимізації витрат, що дозволяє підприємствам масштабувати свої процеси та впроваджувати передові технології автоматизації на базі штучного інтелекту.

Традиційні детерміновані методи базуються на використанні регулярних виразів (Regex) та пошуку за чітко визначеними словниками (Lookup tables). Це найбільш надійний підхід для даних, що суворо відповідають шаблонам. Наприклад, для видобування стандарту ДСТУ або DIN з тексту зазвичай використовується шаблонний пошук. Проте, як зазначають дослідники, головним недоліком таких систем є їхня «крихкість»: найменша друкарська помилка або нестандартне скорочення (наприклад, «гвинт» замість «болт») робить алгоритм непрацездатним без ручного оновлення бази правил [3].

Метод «Instructor» – це бібліотека з відкритим вихідним кодом, яка дозволяє отримувати структуровані дані з великих мовних моделей (LLM) замість звичайного неструктурованого тексту. Основна ідея методу була популяризована та розроблена Джейсоном Лю (Jason Liu). Він винайшов цей підхід як відповідь на складність отримання передбачуваних відповідей від нейромереж, використовуючи для цього потужний інструмент валідації даних у Python – бібліотеку Pydantic.

Особливість методу полягає в тому, що він «нав’язує» моделі певну схему даних (наприклад, JSON з чітко визначеними полями), використовуючи підказки (prompting) та механізми виклику функцій (Function Calling / Tool Use). Якщо модель повертає дані, що не відповідають схемі, Instructor автоматично надсилає повідомлення про помилку назад до LLM з запитом виправити формат [4].

В основі методу покладено концепцію «програмованих підказок», де структура очікуваного результату не просто описується словами, а передається моделі у вигляді суворого технічного опису – класу Python (Pydantic model). Під час формування запиту бібліотека автоматично ін'єктує в системний промпт детальні інструкції щодо форматування відповіді у стандарті JSON, що відповідає заданій схемі. Це перетворює LLM з генератора довільного тексту на структурований парсер, який здатний розпізнавати семантичні зв'язки між технічними характеристиками метизів. Якщо модель повертає дані, що порушують логіку схеми або формат (наприклад, невалідний JSON-об'єкт), «Instructor» активує механізм автоматичних повторних запитів (retries). У такому разі системі передається повідомлення про помилку валідації, і модель, аналізуючи свій попередній результат, самостійно виправляє структуру до моменту повної відповідності заданому класу [4].

Визначальною перевагою використання «Instructor» є безкомпромісний рівень машиночитаності отриманих даних, що критично важливо для інтеграції з промисловими базами даних та ERP-системами. Завдяки суворій типізації, модель фактично «змушена» адаптувати свої широкі знання про мову під конкретні програмні обмеження: наприклад, виділяти діаметр різьби «М10» як окремий рядок, а клас міцності «8.8» трансформувати у числове значення для подальших розрахунків. Такий підхід нівелює головний недолік генеративних моделей – схильність до «галюцинацій» та надмірного багатослів'я. Внаслідок, замість описового тексту користувач отримує валідований об'єкт, який можна миттєво використовувати в автоматизованих ланцюгах обробки без потреби в додатковому ручному коригуванні чи написанні складних регулярних виразів [5].

Для детальної візуалізації послідовності дій та автоматизованого робочого процесу (Workflow) структурування даних було розроблено Activity Diagram (діаграму діяльності), яка ілюструє застосування методу «Instructor». Відповідну Activity Diagram (Workflow) методу «Instructor» зображено на рис. 1.

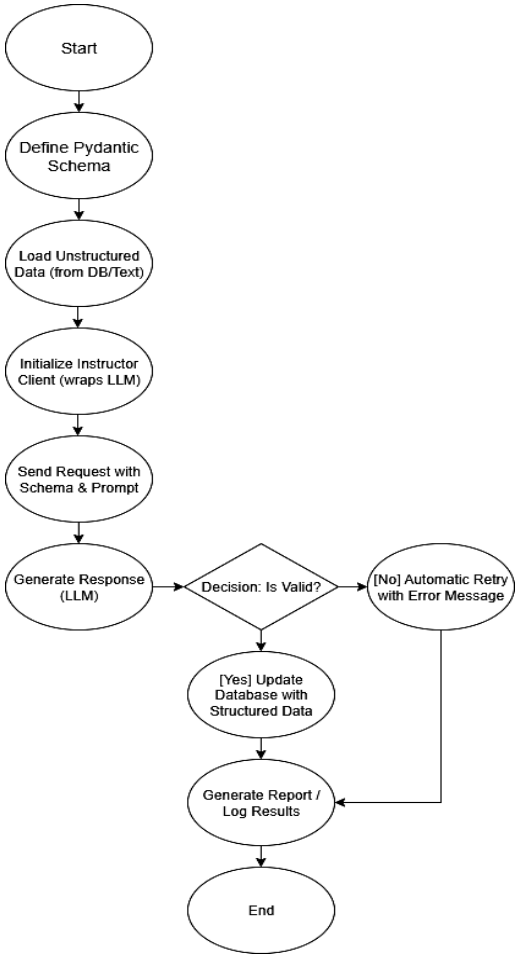

Рисунок 1 – Activity Diagram (Workflow) методу «Instructor»

Діаграма відображає повний цикл обробки даних – від завантаження неструктурованого тексту до отримання валідованого об'єкта Pydantic та оновлення бази даних. Ключовим елементом є цикл автоматичних повторних запитів (Retries), який активується у разі помилки валідації.

Опис ключових етапів діаграми:

- Start: Початок процеса;
- Define Pydantic Schema (Python Class): Визначення структури даних (схеми);
- Load Unstructured Data (from DB/Text): Завантаження не оброблених даних;
- Initialize Instructor Client (wraps LLM): Ініціалізація клієнта «Instructor»;
- Send Request with Schema & Prompt: Відправка запиту з promt та схемою;
- Generate Response (LLM): Генерація відповіді моделлю;
- Validate Response against Schema (Pydantic): Валідація відповіді через Pydantic;
- Decision: Is Valid?
  - [No] Automatic Retry with Error Message: Автоматичний повторний запит із повідомленням про помилку;
  - [Yes] Update Database with Structured Data: Оновлення бази даних структурованими даними;
- Generate Report / Log Results: Створення звіту або логування результатів;
- End: Завершення процесу.

Метод «Instructor» (Instructor + Pydantic) вимагає превентивного проектування структури даних безпосередньо на рівні програмного коду через створення класів Pydantic, де чітко детермінуються всі атрибути технічного виробу (наприклад, standard_name: str, size: str) та їхні типи. Такий підхід створює додатковий рівень абстракції, що забезпечує сувору типізацію виводу. Процес інтеграції передбачає обов’язкове розширення («патчинг») стандартного клієнта OpenAI за допомогою бібліотеки Instructor, що докорінно змінює механізм обробки відповідей: замість неструктурованого текстового рядка програма очікує на виході готовий екземпляр класу. Алгоритм запуску цього методу містить три критичні кроки: дефініцію схеми валідації в коді, ініціалізацію модифікованого клієнта та виклик методу create із обов’язковою передачею параметра response_model, що дозволяє моделі орієнтуватися на задану структуру під час генерації.

Метод «Instructor» впроваджує критично важливий шар синтаксичного контролю над вихідними даними мовної моделі, перетворюючи ймовірнісну генерацію тексту на детермінований процес структурування інформації. Технологічний цикл виконання в програмному коді розпочинається з декларації схеми через клас Standardized Metyz (Base Model), що виконує роль «цифрового трафарета» та жорстко регламентує структуру відповіді, зобов'язуючи модель повертати конкретні поля (наприклад, ukr_name та eng_name) у строго визначеному форматі рядків. Наступним етапом є «патчинг» клієнта за допомогою команди instructor.patch(), що модифікує стандартні методи запиту для забезпечення стабільної підтримки Pydantic-моделей. Під час інференсу програма не просто отримує текстовий рядок, а оперує готовим екземпляром класу, що значно спрощує подальшу інтеграцію з базами даних. Вирішальною перевагою методу є функція автоматичної перевірки: у випадку, якщо модель Mistral-7B генерує некоректний формат, бібліотека ініціює ітеративний цикл повторних запитів, надсилаючи моделі повідомлення про помилку валідації для самокорекції, що мінімізує ризик появи неструктурованих або помилкових даних у промислових системах [6].

2. АВТОРСЬКИЙ МЕТОД СТАНДАРТИЗАЦІЇ НАЗВ МЕТИЗІВ

Авторський метод стандартизації технічних найменувань базується на концепції семантичного перетворення через контекстне навчання (*in-context learning*) та стратегію прямого керування логікою формування назви. На відміну від жорстко детермінованих підходів, цей метод покладається на здатність великих мовних моделей до когнітивного аналізу структури тексту без потреби у попередній зміні програмного ядра бібліотек. Основу підходу складає використання розширеного промпту, який інтегрує в себе чітко визначену рольову модель («експерт з технічної стандартизації») та набір конкретних прикладів (*Few-shot*), що слугують еталонами для трансформації. Це дозволяє моделі вибудовувати внутрішні логічні зв’язки між неструктурованим входом та цільовим форматом, спираючись на надані прецеденти.

Принцип функціонування методу полягає у зміщенні акценту з програмного опису структури (як це реалізовано в методі «Instructor») на лінгвістичне моделювання шаблону безпосередньо в тілі запиту. Структура вихідних даних задається не кодом Python, а семантичним патерном («Назва + Стандарт +

Розмір»), що дозволяє гнучко адаптувати вивід під специфічні вимоги різних промислових стандартів. Процес обробки завершується отриманням текстової відповіді у форматі JSON, яка піддається десеріалізації стандартними засобами мови Python. Такий підхід мінімізує залежність від зовнішніх бібліотек-валідаторів, перекладаючи відповідальність за структурну цілісність на якість контекстного наповнення промпту.

Ключовою особливістю та стратегічною перевагою авторського методу є його висока адаптивність та орієнтація на специфіку міжнародних і державних технічних стандартів (DIN, ISO, ДСТУ). Метод фокусується не лише на видобуванні атрибутів, а й на інтелектуальній трансформації та перекладі технічних термінів. Модель самостійно ідентифікує та класифікує логічні блоки складної назви метизу, використовуючи надані зразки як систему координат. Це забезпечує високу точність при роботі з двомовними каталогами або при переході між різними системами кодування виробів, де критично важливим є збереження технічної ідентичності деталі при зміні її лінгвістичного опису.

Авторський метод (пряма JSON-серіалізація) базується на принципах семантичної гнучкості та використанні стандартних REST-протоколів. Основна перевага підходу полягає у відсутності потреби в декларації жорстких класів у коді, оскільки структура даних задається безпосередньо в тілі запиту через інженерію промптів (*Prompt Engineering*). Ефективність такої передачі структури через Few-shot приклади обґрунтована здатністю моделей до контекстного навчання (*in-context learning*), що дозволяє їм ідентифікувати складні закономірності та формати виводу без додаткового донавчання ваг мережі [7].

Технічна реалізація методу є максимально автономною: додаток використовує бібліотеку requests для прямої взаємодії з API сервера (до прикладу, LM Studio), що робить програмний код незалежним від сторонніх валідаторів. Процес обробки включає формування JSON-корисного навантаження (*payload*), відправку POST-запиту та десеріалізацію відповіді за допомогою стандартної функції json.loads(). Такий підхід забезпечує універсальну сумісність із будь-яким програмним середовищем, оскільки JSON є фундаментальним стандартом обміну даними, що гарантує цілісність структури при передачі між незалежними вузлами системи [8].

За допомогою мови програмування Python було сформовано реляційну базу даних, що складається з єдиної структурованої таблиці для зберігання технічних номенклатур. Програмна логіка реалізує ітераційний цикл обробки: спеціалізований SQL-запит здійснює селекцію цільових записів, для кожного з яких динамічно формується інженерний промпт (завдання), що містить контекстні вказівки та Few-shot приклади. Взаємодія з інтелектуальним ядром відбувається шляхом відправки POST-запитів на локальний сервер LM Studio, який забезпечує роботу мовної моделі в ізольованому середовищі. Після завершення інференсу сервер повертає структуровану відповідь у форматі JSON. Програма на Python виконує десеріалізацію отриманого об'єкта, видобуває стандартизовані назви та переклади, після чого ініціює операцію UPDATE для синхронізації значень у таблиці бази даних [9].

Для візуалізації архітектури рішення була розроблена Component / Deployment Diagram (рис. 2), яка відображає комплексну схему підключення локального середовища [10, 11, 12].

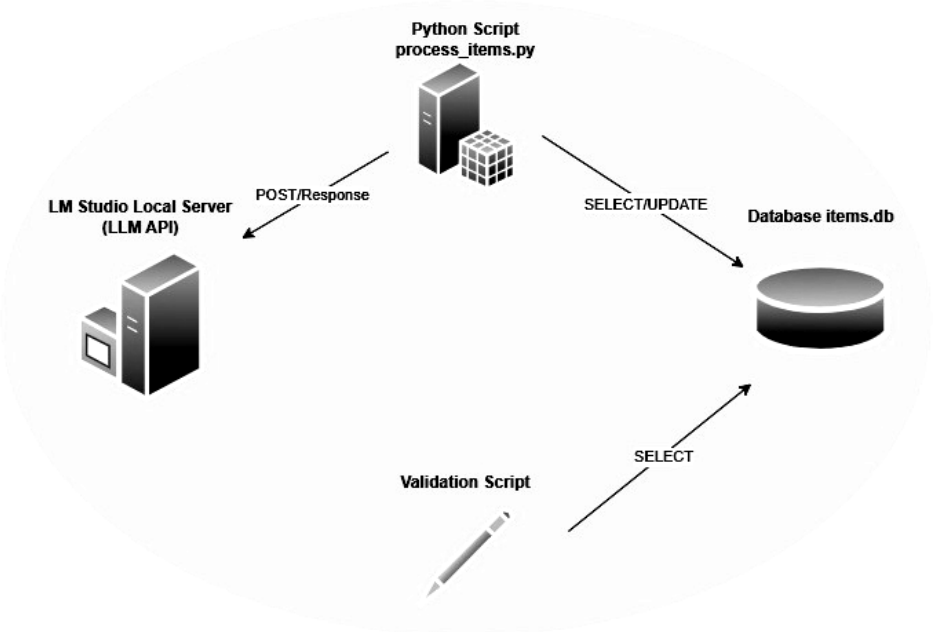

Рисунок 2 – Схема підключення авторського методу (Component / Deployment Diagram)

Діаграма демонструє логічний взаємозв'язок між компонентами (Python-клієнт, SQL-двигун, API-інтерфейс) та їх фізичне розгортання на локальних обчислювальних потужностях та деталізує внутрішню архітектуру системи та взаємодію її функціональних модулів:

- Прикладний модуль (process_items.py): Центральний керуючий скрипт на базі Python, що виконує роль оркестратора процесів. Він ініціює ітераційну обробку даних, здійснює динамічне формування семантичних запитів (промптів) для мовної моделі та керує транзакціями запису результатів у сховище.
- Локальне сховище даних (items.db): Реляційна база даних формату SQLite3, призначена для персистентного зберігання номенклатурних записів. Вона містить цільові поля для аналізу (raw_name) та верифіковані результати стандартизації (ukr_name, eng_name). Взаємодія з базою реалізована через оптимізовані SQL-запити вибору та оновлення даних.
- Сервер інференсу (LM Studio): Локальне обчислювальне середовище, що забезпечує функціонування мовної моделі mistral-7b-instruct-v0.2. Модуль виступає в ролі інтелектуального ядра, яке приймає структуровані HTTP POST-запити та повертає результати семантичного розбору у форматі JSON.
- Модуль аналітики та верифікації (validate_items.py): Спеціалізований інструментарій для проведення пост-обробки. Скрипт здійснює автоматичний аудит якості отриманих даних, перевіряє відповідність синтаксичним правилам та формує статистичні звіти щодо точності роботи системи.

Для детальної візуалізації послідовності операцій та логіки автоматизованого оброблення інформації було розроблено Activity Diagram (Workflow). Ця діаграма відображає повний життєвий цикл трансформації технічних найменувань метизів: від ініціації селекції неструктурованих записів із бази даних до фінальної верифікації результатів.

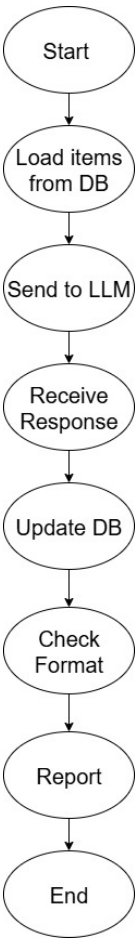

Рисунок 3 – Activity Diagram (Workflow) авторського методу

3. ПОРІВНЯННЯ МЕТОДІВ НА РЕЗУЛЬТАТАХ ЕКСПЕРИМЕНТУ

Для оцінки ефективності запропонованих рішень було проведено серію тестів на контрольному датасеті, що складався зі 100 одиниць номенклатури метизів. Вибірка включала вироби різної складності: від стандартних болтів до шайб та спеціальних кріплень із різними типами покриття.

Авторський метод стандартизації назв метизів на основі LLM-моделі пройшов через етап ітераційного вдосконалення інженерії промптів (Prompt Engineering). На початкових етапах показники точності були низькими через нездатність моделі стабільно дотримуватися заданого шаблону.

- Початкові ітерації: На перших спробах та проведенні експериментів з стандартизації назв модель повертала досить не точні результати. Показники точності стандартизованих назв варіювалися від 12% до 33%. Типовими проблемами були «ефект дзеркала» (модель просто дублювала вхідний рядок без змін) та порушення порядку елементів у макеті стандарту. Проведене порівняльне тестування продемонструвало критичну залежність точності моделі від якості вхідних інструкцій. Використання первинного неоптимізованого запиту (Prompt №1) призвело до того, що 67% згенерованих відповідей містили помилки або невідповідності. Це підтверджує гіпотезу про визначальну роль інженерії промптів (*Prompt Engineering*) та ретельної підготовки контекстних даних для коректної роботи великих мовних моделей у вузькоспеціалізованих доменах
- Фінальний результат: Після фінального доопрацювання структури промпту та впровадження конкретних Few-shot прикладів, авторський метод продемонстрував 100% точність (100 зі 100 записів). Модель навчилася коректно ідентифікувати семантичні межі між типом виробу, стандартом та розміром.

Незважаючи на сувору валідацію схеми Pydantic, метод «Instructor» продемонстрував нижчу якість семантичної обробки технічних даних. Із 100 записів лише 76 відповідають стандарту, тоді як 24 містять критичні помилки. Аналіз результатів дозволив класифікувати помилки методу «Instructor» за трьома основними категоріями:

- Галюцинації в назвах (7 випадків): Виникнення неіснуючих термінів через неправильну лематизацію. Приклад: Вхід: Шайба 8 DIN 125 Результат: Шайка DIN 125 M125. Заміна коректного терміна «Шайба» на помилковий «Шайка».
- Помилкова класифікація типу виробу (10 випадків): Найбільш критична помилка, при якій модель неправильно інтерпретує стандарт. Приклад: Вхід: Шайба 16 DIN 125 $\rightarrow$ Результат: Шпилька DIN 125 M16. Модель помилково асоціює стандарт DIN 125 із класом шпильок, ігноруючи пряму вказівку у вхідних даних.
- Конфляція (злиття) атрибутів (7 випадків): Помилкове об'єднання номера стандарту з розмірними характеристиками. Приклад: Вхід: Шайба 8 DIN 125 $\rightarrow$ Результат: ... M125 замість M8. Модель підставляє номер стандарту в поле розміру, що робить дані непридатними для використання в складських системах.

ВИСНОВКИ

Експеримент довів, що для складних технічних доменів, таких як номенклатура метизів, авторський метод на основі Few-shot prompting є надійнішим за метод «Інструктор». Хоча бібліотека Instructor гарантує технічну валідність JSON-файла, вона, як виявилося, обмежує «когнітивну гнучкість» моделі. Авторський метод, надаючи моделі приклади (еталони), дозволяє їй краще розуміти ієрархію технічних параметрів, що призвело до 100% точності проти 76% у конкурентного підходу.

Метод «Instructor» виявився схильним до специфічних дефектів – «злиття» номерів стандартів із розмірами (наприклад, перетворення DIN 125 на розмір M125). Це свідчить про те, що примусова типізація через Pydantic-схеми може змушувати модель «підганяти» дані під поля, втрачаючи логічний зв'язок. Авторський метод успішно нівелює цю проблему через чітке розмежування блоків у текстовому шаблоні.

Авторський метод демонструє вищу адаптивність: для зміни логіки стандартизації достатньо лише змінити текст промпту (Low-code). В той же час метод «Instructor» вимагає переписування програмної архітектури (класів Pydantic) при кожній зміні структури вхідних даних, що ускладнює його підтримку в динамічних виробничих умовах.

Висока точність авторського методу стандартизації назв метизів на основі LLM-моделі робить його готовим до інтеграції в реальні системи управління підприємством. Здатність методу не лише

нормалізувати назву, а й виконувати точний технічний переклад (локалізацію), забезпечує уніфікацію баз даних для міжнародної співпраці без залучення фахівців-лінгвістів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ / REFERENCES

1. Hazen, B. T., Boone, C. A., Ezell, J. D., & Jones-Farmer, L. A. (2014). Data quality for data science, predictive analytics, and business intelligence in supply chain management: An introduction to the problem and suggestions for research. *International Journal of Production Economics*, 154, 72-80.
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
3. Friedl, J. E. (2006). Mastering Regular Expressions. O'Reilly Media. https://www.oreilly.com/library/view/mastering-regular-expressions/0596528124/
4. Liu, J. (2023). *Instructor: Structured Extraction using LLMs*. GitHub Repository. https://github.com/567-labs/instructor
5. Weng, L. (2023). *LLM Powered Autonomous Agents*. OpenAI Blog / Lil'Log. https://lilianweng.github.io/posts/2023-06-23-agent/
6. Pydantic Team. (2024). *Validation Decorators and Models.* URL: https://docs.pydantic.dev/latest/concepts/models/#basic-model-usage
7. Brown, T. B., et al. (2020). *Language Models are Few-Shot Learners*. arXiv:2005.14165. URL: https://arxiv.org/abs/2005.14165
8. JSON.org. (2024). *The JSON Data Interchange Standard (ECMA-404).* URL: https://www.json.org/json-en.html
9. Starzhynskyi, V., Bisikalo, O. (2025). Using local LLM models for standardization and multilingual translation of technical product names. Measuring and computing devices in technological processes, 84(4), pp. 407–415. doi: 10.31891/2219-9365-2025-84-49.
10. Starzhynskyi, V. Bisikalo, O. Using local LLM models for standardization of hardware names. VNTKP VNTU. Faculty of Intellectual Information Technologies and Automation, Ukraine, Mar. 2026. Available at: <https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2026/paper/view/27436/22723>. Date accessed: 06 Mar. 2026.
11. Bisikalo, O.; Kharchenko, V.; Kovtun, V.; Krak, I.; Pavlov, S. Parameterization of the Stochastic Model for Evaluating Variable Small Data in the Shannon Entropy Basis. *Entropy* 2023, *25*, 184.
12. Intellectual technologies in medical diagnostics, treatment and rehabilitation: monograph / [S.V. Pavlov, O.G. Avrunin, S.M. Zlepko, E.V. Bodianskyi and others]; edited by S. Pavlov, O. Avrunin. – Vinnytsia: PP “TD “Edelweiss and K”, 2019. – 260 p.

Дата надходження: 05.02.2026
Дата прийняття до друку після рецензування: 20.04.2026
Дата публікації: 18.06.2026

СТАРЖИНСЬКИЙ ВАЛЕРІЙ ЮРІЙОВИЧ – аспірант групи 126-23а, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, Вінниця, ***e-mail: 3372292@gmail.com***, **https://orcid.org/0009-0009-3827-0122**

БІСІКАЛО ОЛЕГ ВОЛОДИМИРОВИЧ — доктор технічних наук, професор, завідувач кафедри Автоматизації та інтелектуальних інформаційних технологій, Вінницький національний технічний університет, м. Вінниця, ***e-mail: obisikalo@vntu.edu.ua,*** **https://orcid.org/0000-0002-7607-1943**

Valerii STARZHYNSKYIY, Oleg BISIKALO

METHOD OF AUTOMATED STANDARDIZATION OF METALLIC NAMES BASED ON THE LLM MODEL

Vinnytsia National Technical University