# PERSONAL LLM AND AI AGENTS ON LOCAL EQUIPMENT: ARCHITECTURE, BENCHMARKING AND IMPLEMENTATION FEASIBILITY ASSESSMENT

**Smidovych L.**
*[0000-0000-0000-0000]*
**Uzun D.**
*[0000-0001-5574-550X]*
*Department of Computer Systems, Networks and Cybersecurity, National Aerospace University "Kharkiv Aviation Institute",*
*17, Vadym Manko Str., 61070, Kharkiv, Ukraine*

# ПЕРСОНАЛЬНІ LLM ТА ШІ-АГЕНТИ НА ЛОКАЛЬНОМУ ОБЛАДНАННІ: АРХІТЕКТУРА, БЕНЧМАРКІНГ ТА ОЦІНКА ДОЦІЛЬНОСТІ ВПРОВАДЖЕННЯ

**Смідович Л.**
*[0000-0000-0000-0000]*
**Узун Д.**
*[0000-0001-5574-550X]*
*Кафедра комп'ютерних систем, мереж та кібербезпеки, Національний аерокосмічний університет «Харківський авіаційний інститут»,*
*Україна, 61070, м. Харків, вул. Вадима Манька, 17*
https://doi.org/10.5281/zenodo.20448109

**Abstract**

Most systems based on large language models operate via cloud servers, and this is unacceptable for organizations with sensitive data. At the same time, open models (LLaMA, Qwen, Mistral) already allow you to run billions of parameters on a regular video card thanks to quantization. The article describes a prototype of a desktop AI agent that runs locally via Ollama. The system has its own ReAct loop instead of LangChain, two-layer memory with background classification on a separate small model, a set of tools for working with notes and files, and built-in benchmarking on 29 test prompts. We compared four models on a single GPU (RTX 3090, 24 GB). Qwen 2.5 14B turned out to be the best in terms of speed and quality, and Mistral Nemo 12B was unable to correctly call the tools at all. For an ordinary user, a cloud subscription is still more profitable, but for a company with 50+ employees, a local solution pays for itself in a year.

**Анотація**

Більшість систем на базі великих мовних моделей працює через хмарні сервери, і для організацій з чутливими даними це неприйнятно. Водночас відкриті моделі (LLaMA, Qwen, Mistral) вже дозволяють запускати мільярди параметрів на звичайній відеокарті завдяки квантизації. У статті описано прототип десктопного ШІ-агента, який працює локально через Ollama. Система має власний цикл ReAct замість LangChain, двошарову пам'ять з фоновою класифікацією на окремій малій моделі, набір інструментів для роботи з нотатками і файлами, та вбудований бенчмаркінг на 29 тестових промптах. Ми порівняли чотири моделі на одному GPU (RTX 3090, 24 ГБ). Qwen 2.5 14B виявився найкращим за балансом швидкості і якості, а Mistral Nemo 12B взагалі не зміг коректно викликати інструменти. Для звичайного користувача хмарна підписка поки що вигідніша, але для компанії з 50+ працівниками локальне рішення окупається за рік.

**Keywords:** Large Language Models, AI agent, quantization, Ollama, benchmarking, RAG, ChromaDB, local deployment.

**Ключові слова:** Великі мовні моделі, ШІ-агент, квантування, Ollama, бенчмаркінг, RAG, ChromaDB, локальне розгортання.

## 1 Вступ

За три роки з моменту запуску ChatGPT мовні моделі з експериментальної технології перетворились на робочий інструмент. За різними оцінками, ChatGPT має близько мільярда активних користувачів на місяць [1, 2]. Claude, Gemini та десятки менших сервісів додають ще стільки ж. Люди пишуть листи, аналізують документи, генерують код, і рідко замислюються над тим, куди йдуть їхні дані.

А йдуть вони на сервери OpenAI, Google чи Anthropic. Для особистого листування це прийнятно. Для юридичної фірми, яка завантажує договори клієнтів, або лікарні, яка передає медичні записи, це може бути порушенням законодавства про захист персональних даних. Навіть якщо провайдер обіцяє не використовувати дані для тренування, юридичний ризик залишається, бо дані фізично знаходяться на чужому сервері.

Паралельно з'явились відкриті моделі. Meta випустила LLaMA [3], Mistral AI зробили Mistral [4], Alibaba Cloud представила Qwen [5]. За якістю вони наближаються до закритих аналогів, а квантизація (GGUF, GPTQ, AWQ) дозволяє стиснути модель у 3-4 рази без помітної деградації [6, 7]. Мо-

дель з 14 мільярдами параметрів, яка у повній точності займає 28 ГБ, після квантизації поміщається у 9 ГБ відеопам'яті.

Ми вирішили перевірити, чи можна зібрати на цій базі повноцінного ШІ-агента, який працюватиме автономно. Не просто чатбот, а систему, яка запам'ятовує контекст, працює з файлами, шукає по документах, і при цьому нічого не відправляє в інтернет. А заодно побудувати інструмент для порівняння моделей, щоб зрозуміти, яка з них підходить для яких задач.

*Далі ми спершу оглянемо наявні рішення і покажемо, чого їм бракує (розділ 2), потім розберемо архітектуру самого агента (розділ 3) та методику тестування з результатами (розділ 4), а під кінець чесно порахуємо, кому таке локальне розгортання справді має сенс (розділ 5).*

**2** Аналіз існуючих рішень

Перш за все варто розмежувати чатбот та агента. Wang та співавтори у своєму огляді [8] виділяють чотири складові LLM-агента: профіль (хто він і що вміє), пам'ять (що він знає про користувача), планування (як розбиває задачу на кроки), та дії (що робить). Yao та співавтори [9] запропонували конкретний паттерн для організації дій, ReAct, де модель почергово думає і діє, перевіряючи себе через зовнішні джерела замість того, щоб покладатись на «внутрішні знання».

Для запуску моделей локально є кілька варіантів. Ollama [10] обгортає llama.cpp [11] у зручний інтерфейс: одна команда завантажує і запускає модель з GPUприскоренням. Він також підтримує native function calling, тобто модель може у стандартному форматі попросити викликати зовнішню функцію. vLLM потужніший для серверного використання з continuous batching, але для десктопного застосунку на Windows це надлишково.

Серед фреймворків для побудови агентів найвідоміший LangChain [12]. Ми спробували його на початку і відмовились. API змінюється між версіями, залежностей десятки, а коли щось ламається, дебажити через п'ять шарів абстракції складно. AutoGen від Microsoft цікавий для мультиагентних сценаріїв, але для одного помічника надлишковий.

Є і готові десктопні застосунки. Jan дає офлайн чат, але без пам'яті між сесіями, без інструментів, без RAG. По суті красивий інтерфейс до Ollama. LM Studio аналогічний, плюс закритий код. AnythingLLM найближчий до того, що ми робимо: є RAG, є щось схоже на агентний цикл, є Docker. Але бенчмаркінгу немає, автоматичної класифікації пам'яті немає, профілів з різними наборами інструментів теж.

У роботі [15] Zheng та співавтори показали, що велика мовна модель може оцінювати відповіді інших моделей з якістю, порівнянною з людською оцінкою. Ми використали цей підхід у системі бенчмаркінгу: Qwen 2.5 32B виступає суддею для менших моделей.

**3** Архітектура

### 3.1 Що ми обрали і чому

Рантайм Ollama, мова Python, інтерфейс PySide6 (Qt6), зберігання ChromaDB + SQLite [13, 14]. Агентний цикл написаний з нуля, приблизно 250 рядків. Це менше, ніж один файл конфігурації LangChain для аналогічної задачі.

Моделей у системі шість, кожна зі своєю роллю (табл. 1). Основна робоча модель, Qwen 2.5 14B, займає 9 ГБ відеопам'яті і залишає достатньо місця для KV-cache навіть при довгих діалогах. Qwen 2.5 3B працює у фоні як класифікатор, вирішуючи чи є у розмові щось варте запам'ятовування. nomic-embed-text перетворює текст у вектори для семантичного пошуку. А Qwen 2.5 32B ми завантажуємо окремо, коли треба оцінити якість відповідей менших моделей.

Таблиця 1.

**Моделі та їх ролі у системі**

| Модель | VRAM | Роль |
|---|---|---|
| LLaMA 3.1 8B | 4.9 ГБ | Тестована (швидка) |
| Mistral Nemo 12B (Q4_K_M) | 7.5 ГБ | Тестована |
| Qwen 2.5 14B | 9 ГБ | Основна |
| Qwen 2.5 32B (Q4_K_M) | 19 ГБ | LLM-judge |
| Qwen 2.5 3B | 1.9 ГБ | Класифікатор |
| nomic-embed-text | 274 МБ | Ембедінги |

Все тестувалось на одній машині: RTX 3090 (24 ГБ), 32 ГБ DDR5, Windows 11.

### 3.2 Як працює агентний цикл

Користувач пише запит. Контролер збирає промпт: бере системне повідомлення профілю, шукає релевантні факти з пам'яті (до 3 глобальних і 3 профільних), додає фрагменти з бази знань якщо вона є, та історію розмови. Все це йде в Ollama через streaming.

Модель може відповісти текстом, а може вирішити що їй потрібен інструмент. Тоді вона формує tool_call прямо у стрімі. Контролер перехоплює його, виконує (наприклад створює нотатку або читає файл), повертає результат, і модель продовжує. Максимум 8 таких ітерацій, щоб не зациклитись.

Пам'ять працює на двох рівнях. Короткочасна, це просто останні повідомлення з SQLite з ковзним вікном, щоб не переповнити контекст моделі. Довгочасна зберігається у ChromaDB: окремо глобальні факти (ім'я, мова, алергії) і окремо факти конкретного профілю (деталі проекту, преференції). Після кожної пари повідомлень фоновий потік на Qwen 3B аналізує розмову і вирішує, чи є там щось варте збереження. Користувач нічого не робить і навіть не знає, що класифікація відбувається.

### 3.3 Інструменти і RAG

Інструментів сім: п'ять для нотаток (створити, список, прочитати, видалити, пошукати) і два для файлів (прочитати, переглянути директорію). Файлові операції обмежені робочою текою, дозволені тільки текстові формати, максимум 100 КБ на читання, таймаут 30 секунд.

Бази знань прив'язані до профілів. Markdown файли розбиваються на шматки по ~2000 символів з перекриттям і індексуються у ChromaDB. При запиті система знаходить три найбільш схожих фрагменти і додає їх у промпт. Якщо файл у теці змінився, переіндексовується тільки він, а не вся база.

## 4 Бенчмаркінг

### 4.1 Як ми тестували

Ми зібрали 29 тестових промптів у п'яти категоріях. Speed (5 штук) для чистого заміру швидкості. Quality UA (8 штук) для перевірки українською: переклади, ділові листи, граматика. Code (5 штук): Python, SQL, regex. Reasoning (6 штук): арифметика, логіка, послідовності. Tool Use (5 штук): чи модель викликає правильний інструмент з правильними аргументами.

Оцінювання трирівневе. Для Reasoning і Tool Use працює автоматична перевірка: правильна відповідь або ні. Для Quality UA і Code оцінює Qwen 2.5 32B як суддя [15], від 1 до 5. І якщо автоматична оцінка сумнівна, є ручна, вона має найвищий пріоритет. Перший запит кожної моделі відкидається, бо він включає час завантаження у VRAM і псує статистику. Між моделями VRAM очищається через keep_alive=0. **4.2 Що вийшло**

Результати зведено у табл. 2.

Таблиця 2.

**Результати порівняння моделей**

| Модель | tok/s | TTFT, мс | VRAM, ГБ | Quality UA | Tool Use |
|---|---|---|---|---|---|
| LLaMA 3.1 8B | 525 | 141 | 16.5 | 2.5-3.0 | часткова |
| Mistral Nemo 12B | 50 | 228 | 19.7 | 3.0-3.5 | 0/5 |
| Qwen 2.5 14B | 134 | 131 | 18.2 | 3.5-4.0 | найкраща |
| Qwen 2.5 3B (фонова) | 470 | 97 | 6.8 | — | — |

LLaMA 3.1 8B очікувано найшвидша, 525 токенів на секунду. Але з українською у неї проблеми: може перейти на російську посеред речення або зробити граматичну помилку, якої носій мови б не зробив. Оцінка LLM-judge 2.5-3.0 з 5.

Qwen 2.5 14B повільніший (134 tok/s), зате з українською працює краще за всіх: 3.5-4.0 з 5. Час до першого токена 131 мс, тобто відповідь починає з'являтись миттєво. Для інтерактивного діалогу це комфортно.

Mistral Nemo 12B здивував. Повільний (50 tok/s), займає найбільше VRAM (19.7 ГБ), і головне, взагалі не зміг викликати інструмент. Нуль з п'яти. Модель генерувала текстову відповідь замість того, щоб сформувати структурований виклик. Для нас це означає, що для агентних сценаріїв підтримка function calling у конкретній моделі важливіша, ніж кількість параметрів.

## 5 Кому це потрібно

Якщо чесно, звичайному користувачу поки що ні. Підписка на ChatGPT Plus коштує 20 доларів на місяць і дає кращу якість. Відеокарта RTX 3090 коштує 1000-1500 доларів на вторинному ринку, і навіть після цього локальна модель поступається за якістю.

Для бізнесу розрахунок інший. Компанія з 50 працівниками платить за ChatGPT Enterprise приблизно 36 000 доларів на рік. Сервер з GPU коштує одноразово. Після першого року різниця стає відчутною, а дані при цьому нікуди не йдуть. Ми визначили чотири конкретних сценарії: корпоративний асистент з RAG по документації, HRпомічник для онбордингу (покриває 70-80% типових питань), контроль якості на виробництві через мультимодальні моделі, та медичний попередній скринінг.

Є ще один аргумент на користь локальних рішень, і він пов'язаний з часом. Qwen 2.5 72B вже конкурує з LLaMA 3.1 405B при п'ятикратно меншому розмірі [5]. Якість відкритих моделей росте швидше, ніж вимоги до заліза. Те, що сьогодні дає GPT-4 [16], через два-три роки зможе модель на звичайній відеокарті. Залізо при цьому не змінюється.

## 6 Висновки

Ми побудували працюючого ШІ-агента, який запускається на одній споживчій відеокарті і працює без інтернету. У ньому є власний цикл ReAct замість LangChain, двошарова пам'ять з фоновою класифікацією, інструменти в пісочниці, профілі зі своїми базами знань і вбудований бенчмаркінг, що сам оцінює якість відповідей. Серед існуючих локальних рішень (Jan, LM Studio, AnythingLLM) такої комбінації немає. З чотирьох тестованих моделей Qwen 2.5 14B показав найкращий баланс. Mistral Nemo 12B не підтримує виклик інструментів, що робить його непридатним для агентних задач. Для бізнесу з чутливими даними рішення окупається менш ніж за рік. Далі планується мультимодальність, мультикористувацький режим і QLoRA дотренування.

**Список літератури:**

1. Vaswani A. et al. Attention Is All You Need // NeurIPS. — 2017. — arXiv:1706.03762.
2. Brown T. B. et al. Language Models are Few-Shot Learners // NeurIPS. — 2020. — arXiv:2005.14165.
3. Dubey A. et al. The Llama 3 Herd of Models // arXiv. — 2024. — arXiv:2407.21783.
4. Jiang A. Q. et al. Mistral 7B // arXiv. — 2023. — arXiv:2310.06825.

5. Yang A. et al. Qwen2.5 Technical Report // arXiv. — 2024. — arXiv:2412.15115.

6. Frantar E. et al. GPTQ: Accurate Post-Training Quantization // ICLR. — 2023. — arXiv:2210.17323.

7. Dettmers T. et al. QLoRA: Efficient Finetuning of Quantized LLMs // NeurIPS. — 2023. — arXiv:2305.14314.

8. Wang L. et al. A Survey on LLM based Autonomous Agents // Frontiers of CS. — 2024. — arXiv:2308.11432.

9. Yao S. et al. ReAct: Synergizing Reasoning and Acting // ICLR. — 2023. — arXiv:2210.03629.

10. Ollama Documentation. — 2024. — URL: https://github.com/ollama/ollama

11. llama.cpp — LLM inference in C/C++. — 2024. — URL: https://github.com/ggerganov/llama.cpp

12. LangChain Documentation. — 2024. — URL: https://docs.langchain.com

13. ChromaDB Documentation. — 2024. — URL: https://docs.trychroma.com

14. Lewis P. et al. Retrieval-Augmented Generation // NeurIPS. — 2020. — arXiv:2005.11401.

15. Zheng L. et al. Judging LLM-as-a-Judge // NeurIPS. — 2023. — arXiv:2306.05685.

16. OpenAI. GPT-4 Technical Report // arXiv. — 2023. — arXiv:2303.08774.