Local LLM Evaluation Lab Report

1. Introduction

The goal is to install and evaluate local LLMs, why local models matter (privacy, cost, control), and experiment and compare the speed, size and output quality when given the same prompts. All experiments were conducted using CPU-only inference on a Windows laptop with 16 GB of system RAM. Due to hardware constraints and system stability considerations, testing was limited to quantized models in the ~3B–4B parameter range, which represent a practical balance between performance and usability on non-GPU systems.

---

2. Machine Setup

- **OS:** Windows 11
- **CPU**: 11th Gen Intel(R) Core(TM) i7 - 1165G7 @ 2.80 GHz
- **RAM**: 16GB
- **GPU**: Intel (R) Iris (R) Xe Graphics

After installing Ollama,

- Confirm Ollama works: `ollama --version` or `ollama help`
- To download a model: `ollama pull <model>`
- To run a model: `ollama run <model>`
- To exit: `/bye`

Ollama Version:

```
C:\Users\humag>ollama --version
ollama version is 0.13.5
```

Simple Experiment with phi3:mini:

```
C:\Users\humag>ollama run phi3:mini
>>> Explain what a hash table is in one paragraph.
A hash table, also known as a hash map, is an efficient data structure that allows for constant-time average complexity (O(1)) operations like search,
insertion and deletion when the hash function used to index into the array of buckets works well. It achrieses in providing quick access to its stored
values based on keys by computing indices using a hashing algorithm which maps each key to an integer value within a predefined range (the size of the
table). Collisions, where two or more keys map to the same index are handled through techniques like chaining and open addressing. These methods allow
for efficient storage in sparse tables when they occur but can lead to slower average complexities if not managed well. Hash tables serve as
fundamental building blocks within many modern computer algorithms due to their speedy access capabilities, making them a crucial tool across various
applications from databases management systems to caching mechanisms and more.
```

Simple Experiment with qwen2.5:3b:

```
C:\Users\humag>ollama run qwen2.5:3b
>>> Explain what a hash table is in one paragraph.
A hash table, also known as a hash map or hash array, is an efficient data structure that implements an associative array abstract data type, a
structure that can map keys to values. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can
be located. This allows for extremely fast access and retrieval times, making it highly effective for situations where quick lookups are essential.
```

3. Models Tested

Model	Parameters (Size)	Disk Size	Speed (Observed)	Architecture origin
phi-3:mini	3.8B	2.2 GB	fast, consistent	Microsoft
qwen2.5:3b	3B	2.0 GB	slower, variable (task-dependent)	Alibaba

4. Experiments

- **Prompt 1** - Text summarisation
- **Prompt 2** - Explain python function
- **Prompt 3** - Refactor code
- **Prompt 4** - Write unit tests
- **Prompt 5** - Suggest improvement for code snippet
- **Prompt 6** - Extract key details from text (JSON)
- **Prompt 7** - Convert requirements to test cases (JSON)

Exact prompts are saved in `prompts.md` and full raw outputs are saved in `outputs.md`.

---

5. Results & Observations

- **Speed differences:**
  - Both models performed quickly for summarisation and simple code explanation tasks.
  - For code-related tasks, overall speed was similar, though phi-3:mini generally responded more quickly.
  - qwen2.5:3b showed noticeable delays during structured JSON tasks, particularly when extracting entities from text (prompt 6).
- **Quality differences:**

  Summarisation:

  - `phi-3:mini` produced more technically dense summaries.
  - `qwen2.5:3b` produced clearer and more readable summaries.

  Code help:

  - `qwen2.5:3b` consistently provided clearer explanations and more pedagogical guidance.
  - `phi-3:mini` occasionally overengineered solutions, especially in unit testing tasks.

  Structured output:

  - Both models generated semantically correct JSON.
  - `phi-3:mini` was more concise and format-focused, while `qwen2.5:3b` often added additional explanation.
- **Formatting issues**:
  - `phi-3:mini` followed formatting instructions more strictly, especially when only JSON output was requested.
  - `qwen2.5:3b` frequently included extra explanatory text outside the requested format.
- **Hallucinations**: No major factual hallucinations were observed.
- **JSON reliability**:
  - Both models produced valid JSON structures.
  - `qwen2.5:3b` often wrapped JSON with explanatory text.

  - `phi-3:mini` more consistently returned pure JSON when explicitly instructed.

---

6. Comparison

Task	Winner	Reason
Summarisation	qwen2.5:3b	Clearer, more readable
Code help	qwen2.5:3b	Better pedagogy, less overengineering
Structured output	phi-3:mini	Faster, stricter formatting
Speed consistency	phi-3:mini	Less slowdown on structured tasks
Agent suitability	phi-3:mini	Predictability + format adherence

---

7. Reflection

What worked well?

- Running multiple local LLMs on constrained hardware was successful using quantized models.
- Both models handled summarisation and structure-related tasks reliably.

What failed or was limited?

- Larger or structured prompts caused noticeable slowdowns, especially for `qwen2.5:3b`.
- Some outputs (notably unit tests from `phi-3:mini`) were overengineered for the task requirements.
- CPU-only inference limited experimentation with larger models.

Model choice by task

- **Code review**: `qwen2.5:3b`
- **Summarisation**: `qwen2.5:3b` (clearer, more readable summaries)
- **Structured tasks**: `phi-3:mini`

What would you change next time?

- Test additional prompt constraints to further reduce verbose outputs.
- Evaluate one slightly larger model to better understand performance trade-offs.