BANGOR UNIVERSITY

School of Computer Science and Engineering

An LLM-Based Thermal Lab Agent

A dissertation submitted in partial fulfilment of the

requirements for the degree of MSc Computing

Xin Liu

Supervisor: Prof. Zengbo Wang

1 September 2026

Declaration

I hereby declare that this dissertation is the result of my own work, except where otherwise stated and acknowledged in the text. It has not been submitted, either wholly or in part, for any degree at this or any other university.

Signature: Xin Liu Date: 1 September 2026

Declaration of Generative AI Use

Generative AI tools (e.g., Claude) were used to assist with drafting, editing, structuring, and formatting this dissertation, as well as producing select figures and providing coding assistance (such as syntax debugging and refactoring). All research design, experimental work, data collection and analysis, core system architecture, and hardware integration are entirely my own work. I have reviewed and verified all AI-assisted content and take full responsibility for the final submission.

Abstract

This dissertation presents the Thermal Lab Agent, a platform that combines natural-language agent control with multimodal visual reading of laboratory instruments. The main application is deployed on a Mac Mini and comprises a Python FastAPI hardware backend, a Node.js coding-agent service and a Vue 3 web interface. Visual interpretation is provided by a locally deployed Qwen3.6-35B-A3B multimodal large language model running through vLLM on an NVIDIA DGX Spark. Two cameras observe the displays of a contact thermometer and thermal imager, allowing temperature readings to be extracted without requiring a direct digital data interface to those instruments.

Three autonomous heat–cool rounds ran to completion without intervention, each producing a clean dataset with no outliers. Against manual readings of the same displays, the Qwen3.6-35B-A3B thermometer readings agreed closely: the mean bias was +0.006 °C and the 95 % limits of agreement were ±0.12 °C — within the thermometer's 0.1 °C display resolution — with Lin's concordance correlation coefficient 0.9995. One failure mode was identified and resolved: digit-transition ghosting caused by reduced camera capture resolution, eliminated by reverting to the original resolution. Under the tested experimental conditions, the Qwen3.6-35B-A3B readings agreed sufficiently closely with manual readings to replace manual display transcription for this thermal experiment.

Acknowledgements

I would like to thank Prof. Zengbo Wang for his supervision and guidance throughout the project. Special thanks go to Ciaran Brown and Dr. Simya Olavil Karayi for their hands-on guidance during the assembly of the experimental setup and device testing phases. I am also grateful to the maintainers of the open-source software this work builds upon, in particular the Pi Coding Agent project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Thermal experiments sit at the heart of a large part of laboratory science. Laser processing, materials characterisation, heat-transfer measurement and a good deal of physics and chemistry all depend on heating or cooling a sample and watching how its temperature responds. Yet, for all the automation that has reached the rest of the modern laboratory, the routine of running these experiments has changed remarkably little. An operator still sets the power supply and the laser by hand, keeps an eye on the thermal imager, and copies temperature readings into a log at fixed intervals. This thesis sets out to remove that operator from the loop.

Three problems follow directly from the manual arrangement. The first is cost. Continuous supervision consumes operator time that could be spent elsewhere and puts a hard limit on how many experiments can run in parallel. The second is reliability. A human log is fragile: a digit is misread, a value lands in the wrong column, a sample is skipped, and the resulting dataset quietly inherits those faults without any record that they happened. The third is resolution. A person can only read and write so quickly, so the sampling interval is, at best, measured in minutes. The fast thermal transients that a heating or cooling experiment is often designed to capture fall between the sample points and are lost entirely.

Two recent developments make it realistic to close this gap. Recent multimodal large language models extend conventional LLM capabilities beyond text by incorporating visual understanding. When such a multimodal LLM is provided with images together with language instructions, its visual capability can be used as a vision–language model (VLM). In this project, this capability is used to read temperature values directly from images of laboratory instrument displays. The system therefore combines two capabilities: an LLM-based coding agent for natural-language reasoning and tool use, and multimodal visual understanding for instrument reading. These two capabilities are combined in a single platform, the Thermal Lab Agent, that supervises a thermal experiment from start to finish.

1.2 Aim and Objectives

Aim. To design a platform for autonomous thermal experiment monitoring that integrates an agent, built around VLM-based reading of thermal instrument displays.

Four objectives follow from this aim:

1. Build a full-stack platform that unifies hardware control (a DC power supply, a laser and a camera rig), telemetry persistence and a web interface.

2. Integrate the Pi Coding Agent as the agent framework and provide it with a domain-specific tool layer, so that the hardware can be driven end to end from a natural-language request.

3. Read the displays of thermal instruments (the thermal imager's on-screen display and a seven-segment LCD) with a prompt-engineered VLM.

4. Run a multi-round heat–cool experiment loop autonomously and evaluate its output against a manual-measurement baseline.

1.3 Research Questions

The thesis addresses four research questions.

RQ1. Can a prompt-engineered multimodal LLM reliably read temperature values from thermal instrument displays, and what failure modes remain?

RQ2. Can a coding-agent architecture — the Pi Coding Agent extended with a domain-specific tool layer — drive the full experiment pipeline end to end, from hardware control and synchronised multi-camera capture through to telemetry persistence, on the strength of natural-language instructions?

RQ3. Can the platform execute a multi-round heat–cool experiment loop autonomously, triggered by temperature and time, and produce a valid thermal dataset without human intervention?

RQ4. Under an identical heat–cool protocol (laser heating 10 min, cooling 15 min, 1 min sampling interval), does the automated platform achieve measurement accuracy comparable to manual measurement by a human operator?

1.4 Contributions

The principal contribution is not any single algorithm but the integration of an existing coding agent, an off-the-shelf multimodal model and commodity laboratory hardware into one working system. Concretely, the thesis contributes:

- A working LLM-based laboratory agent that translates natural-language experimental instructions into authorised hardware-control operations;
- A multimodal visual measurement approach using Qwen3.6-35B-A3B to read temperatures directly from instrument displays;
- A distributed deployment architecture in which the Thermal Lab Agent runs on a Mac Mini while multimodal inference is served locally from an NVIDIA DGX Spark;
- Synchronised multi-camera measurement, telemetry persistence and autonomous heat–cool orchestration; and
- Experimental validation against manual readings over repeated heat–cool cycles.

The project was developed and maintained in a version-controlled repository at `https://github.com/au971/Lab-agent` (private); Appendix A records its commit history.

1.5 Thesis Structure

The remainder of this document is organised as follows. Chapter 2 reviews the relevant prior work — AI agents, self-driving laboratories, VLM-based instrument reading and coding agents — and states the research gap they leave open. Chapter 3 derives the requirements and presents the physical experimental setup and the system architecture together with the reasoning behind each design decision. Chapter 4 describes the implementation, including the hardware control layer, the VLM monitoring loop, the agent tool layer and the front end. Chapter 5 reports the results of the evaluation methodology and the VLM prompt-engineering experiments, and Chapter 6 evaluates and discusses each research question in turn against that evidence. Chapter 7 concludes and outlines future work. Appendix A records the evidence of continued development across the project.

Chapter 2

Background and Related Work

This chapter first defines what an AI agent is and how LLM-based agents differ from their rule-based predecessors (Section 2.1). It then reviews self-driving laboratories (Section 2.2), VLM-based instrument reading (Section 2.3), VLMs in thermal and infrared settings (Section 2.4), and coding agents (Section 2.5), and closes with the traditional, scripted automation this work departs from (Section 2.6).

2.1 AI Agents: from Rules to Language Models

An AI agent is a system that perceives its environment and acts within it to achieve a goal. Most definitions identify four components: a *perception* interface through which the agent observes the state of the world; a *memory* that carries what it has seen and done; a *reasoning and planning* core that decides the next action; and an *action* (or tool) interface through which it changes the world. Agents differ mainly in how the reasoning core is implemented.

For most of the field's history that core was written by hand: production rules, state machines and planners that could only handle situations the designer had anticipated. Such agents are precise and predictable, but brittle — they cannot follow vague instructions, and extending them means writing new rules. Large language models change this division of labour. In an LLM-based agent the language model *is* the reasoning core: it turns a natural-language goal into a sequence of tool calls, reasons about the results it observes, and re-plans when the world does not match its expectation. The hand-written rules shrink to a thin guardrail, and the capabilities the model brings — understanding natural language, planning without a fixed procedure, and acting through external tools — are exactly what a human operator used to provide. This project is built on such an LLM-based agent, which is why this distinction matters throughout.

2.2 Self-Driving Laboratories

The idea of letting an AI agent run experiments has moved from proposal to practice over the last few years. The AILA framework pairs large language models with atomic-force microscopy hardware and reports an accompanying benchmark of task success across several models [1]. In the quantum domain, the k-agents framework organises laboratory knowledge and runs closed-loop experiments on a superconducting processor through a set of cooperating agents [2]. LabscriptAI applies a coder–reviewer loop to liquid-handling robotics for protein engineering [3].

Two features matter here. First, the field is focused overwhelmingly on chemistry, materials, biology and quantum hardware, where the experiment reduces to robot-driven liquid-handling or measurement; thermal-physics benches (a power supply, a laser, a thermal imager) appear only occasionally. Second, reported systems can adjust parameters but rarely revise the overall strategy without a human, so reliable end-to-end autonomy remains an open problem.

2.3 Vision–Language Models for Instrument Reading

Reading a physical instrument by looking at it is, on its face, exactly the kind of task a vision–language model should handle, but it proves surprisingly hard to get right in practice. MeasureBench frames the problem as one of fine-grained visual perception combined with text reading and reports that general-purpose models still struggle with precise numeric readings [4]. The systems that do work tend to fine-tune the model or pair it with classical computer vision. A staged parameter-efficient fine-tuning framework, for example, adapts a multimodal model to read industrial pointer gauges and reports large gains over its zero-shot counterpart, even under glare [5], an approach that works well but requires a labelled training set.

What is largely absent from this work is the middle path: a general-purpose, *un-fine-tuned* VLM steered only by careful prompting, applied specifically to the displays of thermal instruments. That is the route this thesis takes, and it is the subject of RQ1.

2.4 VLMs in Thermal and Infrared Contexts

Vision–language models have also been brought to bear on thermal and infrared imagery, but with a different aim. VLM-IRIS adapts VLMs to infrared data for scene monitoring in additive manufacturing, dealing with the modality gap between thermal and natural images [6], and work on active infrared thermography combines VLM encoders with a lightweight adapter for subsurface defect analysis in composites [7]. These are scene-understanding tasks: detecting a workpiece, localising a defect, diagnosing a hotspot. None of them reads a numeric temperature off an instrument display and uses it to close a control loop. That distinction separates this thesis from the existing thermal-VLM literature.

2.5 Coding Agents as Laboratory Operators

Most of the self-driving laboratories above are built on planner-style agent frameworks: the agent reasons over a plan and then dispatches commands. A coding agent is a different animal. Its native loop is read–edit–execute, and it is designed to be extended with tools and skills rather than to carry a fixed repertoire of actions. The Pi Coding Agent, the framework used here, exposes this extensibility directly: tools and skills live outside the core agent and are loaded at runtime, which makes it natural to graft on a set of domain-specific hardware tools [8].

2.6 Traditional Instrument Automation

The baseline this work departs from is worth naming. Laboratory instruments have been automated for decades without any language model. LabVIEW, the industry-standard tool, wires a dataflow program out of instrument drivers and virtual-instrument panels, while SCPI commands script the same instruments over GPIB or serial links. These systems are mature, reliable and repeatable, which is why they dominate test and measurement.

Their limitation is where the intelligence sits. A LabVIEW program executes exactly the procedure its engineer wrote in advance: every setpoint, trigger and measurement channel is fixed at design time, and the interface is a panel of buttons and dialogs rather than a conversation. When the experiment changes, a person must rewire the diagram or re-script the sequence. Such systems cannot take an instruction in natural language, cannot plan a novel procedure, and cannot adapt when an observation surprises them. These are precisely the abilities the LLM-based agent of Section 2.1 brings; the contrast between scripted and language-driven control is taken up again in the evaluation of Chapter 6.

2.7 Summary and Research Gap

The review leaves four gaps, each of which maps onto a research question. First, thermal-physics benches are under-represented in the self-driving-lab literature, which has concentrated on chemistry, materials and quantum hardware. Second, the prompt-only, zero-fine-tuning reading of thermal instrument displays has not been seriously tried; the working systems either fine-tune the model or lean on classical CV. Third, coding agents, despite being a natural fit for hardware control through a tool layer, are rarely the agent actually facing the instruments. Fourth, traditional instrument automation — LabVIEW and SCPI scripting — still dominates laboratory practice, yet it offers neither natural-language interaction nor planning, and whether an LLM-based agent can replace it at a thermal bench has not been shown. RQ1–RQ4, stated in Section 1.3, are designed to close these gaps.

Chapter 3

System Design and Architecture

This chapter turns the research questions of Section 1.3 into a concrete design. Section 3.1 derives the requirements. Section 3.2 describes the physical experiment and where it runs. Section 3.3 describes the software architecture, and Section 3.4 sets out and justifies the main design choices. Section 3.5 closes with the principal data flows.

3.1 Requirements Analysis

The four research questions imply a set of functional requirements (FR) that the platform must satisfy, together with a set of non-functional requirements (NFR) that constrain how it does so.

FR1 – Hardware control. Set and read back a programmable DC power supply and a laser over their native interfaces.

FR2 – Synchronised capture. Grab a temporally consistent frame from every connected camera on each sampling tick.

FR3 – VLM reading. Extract a structured temperature reading from the display of a thermal instrument with a vision–language model.

FR4 – Telemetry persistence. Write one row per sample (temperatures, voltage and current) to both a per-run CSV file and a queryable store.

FR5 – Autonomous loop. Run a multi-round heat–cool cycle whose transitions are triggered by temperature and by time.

FR6 – Agent access. Expose the hardware and the loop to a coding agent so that natural-language requests translate into the operations of FR1–FR5.

FR7 – Live UI. Stream system state and experiment progress to a web interface in near real time.

The non-functional requirements are dominated by safety and by the choice of inference. *NFR1* requires that the laser and power supply shut down automatically if a device disconnects or misbehaves. *NFR2* requires the sampling interval to be freely configurable (the runs reported here used 1 min, matching the manual protocol). *NFR3* requires the VLM provider to be swappable (self-hosted or an OpenAI-compatible API) without changing the calling code. *NFR4* requires the backend to boot cleanly without the rig attached, so that the software can be developed away from the laboratory.

3.2 Physical Experimental Setup

The experimental platform provides a simple physical testbed for autonomous laboratory operation. An 808 nm Class 3B diode laser heats an aqueous sample while its temperature response is monitored using two instruments. A contact thermometer provides the primary temperature measurement, while a thermal imager observes the sample region as an auxiliary non-contact measurement. Camera 1 observes the contact thermometer display and Camera 2 observes the thermal-imager display. The camera images are interpreted by the locally deployed multimodal language model, allowing the Thermal Lab Agent to obtain temperature readings through the same visual interfaces that would otherwise be read manually by an operator (Figure 3.1). The thermometer probe is immersed in the sample liquid, while the thermal imager observes the sample container region; the two instruments measure different physical quantities — direct contact versus non-contact surface radiation — so they do not necessarily report the same temperature, and small differences between them are not in themselves errors.

3.3 Architecture Overview

The platform is built as three cooperating processes, shown in Figure 3.2.

- **Hardware backend** (Python, FastAPI, port 8000). Owns the hardware drivers, runs the VLM monitoring loop and persists telemetry. Each device driver is a module-level singleton with a hot-plug monitor thread.
- **Agent service** (Node.js, port 3000). Hosts a Pi Coding Agent session extended with a domain tool layer, and streams agent events over SSE.
- **Web front end** (Vue 3, dev port 5173). A single-page interface that renders live telemetry and forwards the user's natural-language requests to the agent.

The Thermal Lab Agent application is deployed on a Mac Mini, while computationally intensive multimodal inference is performed separately on an NVIDIA DGX Spark. The Mac Mini hosts the hardware backend, agent service and web interface. The DGX Spark hosts the

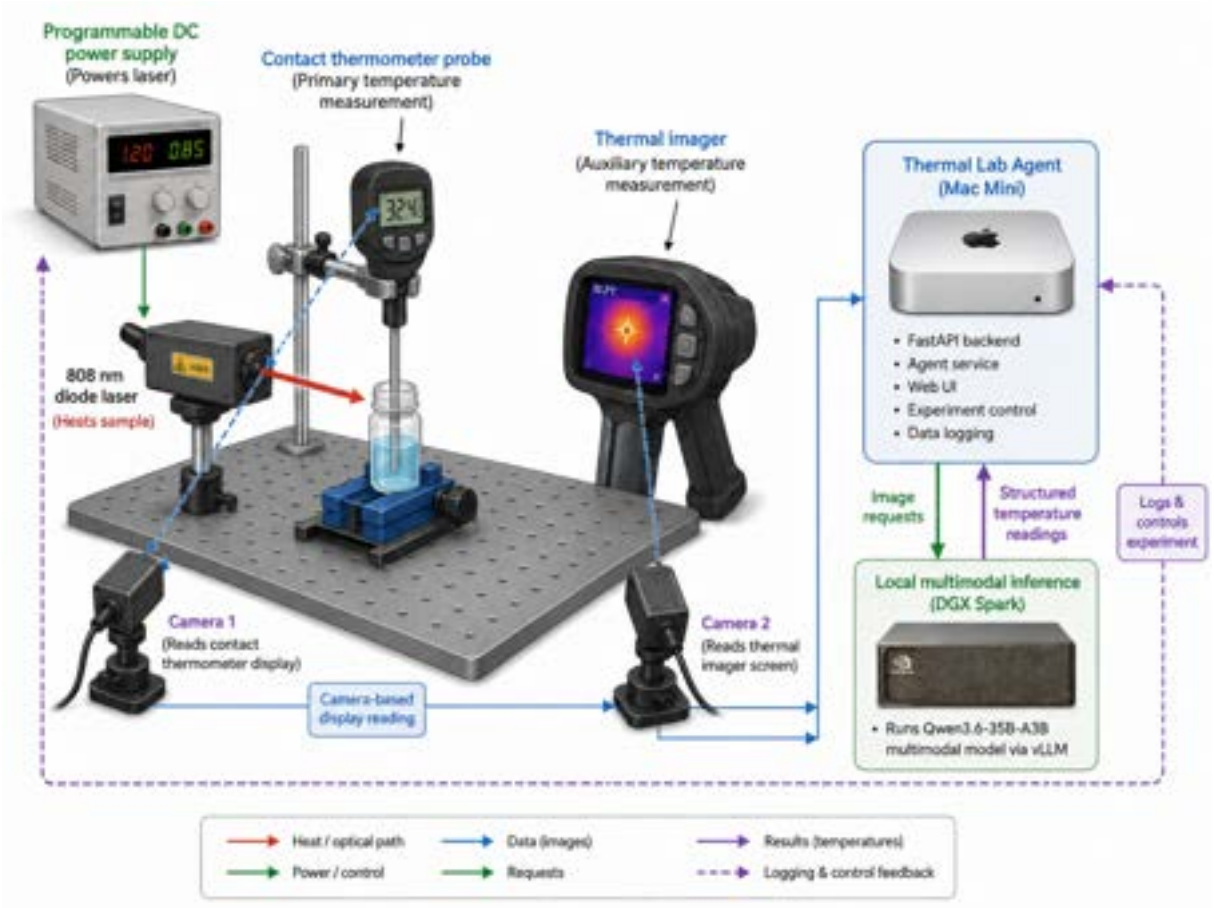


Figure 3.1: The physical experimental setup and its deployment. The power supply drives the laser, which heats the sample; the contact thermometer and the thermal imager measure it; two cameras read their displays and send images to the Thermal Lab Agent on a Mac Mini, which calls a Qwen3.6-35B-A3B model served by vLLM on an NVIDIA DGX Spark.

Qwen3.6-35B-A3B multimodal large language model through vLLM. The two systems communicate over the local network through an OpenAI-compatible API.

The three are joined by a UI bridge. Every line the backend prints is captured by a stdout redirector and pushed as a `system_log` event to connected SSE clients, which is how agent actions and hardware state reach the front end without the front end needing to know about the agent or the hardware directly.

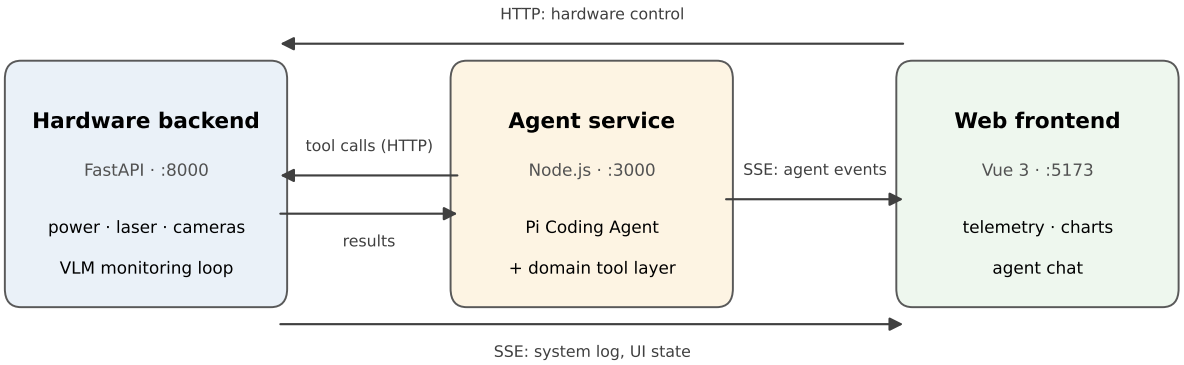


Figure 3.2: Overall three-process architecture of the Thermal Lab Agent.

3.4 Design Decisions

Each choice below records the alternatives considered and the reason for the one adopted. Table 3.1 summarises them.

3.4.1 Frontend: from a Streamlit prototype to a custom full-stack UI

The first working version of the platform was a Streamlit proof of concept, chosen because it gets a live data dashboard running in minutes. It became the wrong tool once the interface needed several things at once: six live camera panels, a temperature chart, hardware control forms and a chat pane, all refreshing independently. Streamlit reruns the whole script on each interaction, which made fine-grained control of that layout awkward, and its server-side Python execution contended with the hardware loop. The interface was therefore rebuilt as a Vue 3 single-page application. Vue keeps the rendering on the client, gives full control over the layout, and, with ECharts, handles the real-time charts that Streamlit could not drive smoothly.

3.4.2 Real-time transport: SSE versus WebSocket

The system has a strongly one-way traffic pattern for state: the backend and agent produce events, and the front end consumes them. Server-sent events fit this pattern exactly. They run over plain

Table 3.1: Summary of key design decisions and their justifications.

Decision	**Alternatives**	**Chosen**	**Key Rationale**
Frontend	Streamlit, Gradio	Vue 3 + Vite	Real-time multi-panel UI, SSE streaming, full layout control
Real-time transport	WebSocket, polling	SSE	One-way telemetry push, simple reconnection, HTTP-native
VLM inference	Cloud-only APIs, fine-tuned VLM	Self-hosted vLLM + commercial APIs	OpenAI-compatible interface makes providers swappable for model comparison
Agent runtime	Planner agents (LangChain-style)	Pi Coding Agent + domain tools	Minimal extensible loop, extension/SKILL mechanism
Persistence	DB only / files only	SQLite + CSV	Human-readable exports plus queryable store

HTTP, reconnect automatically, and need no broker or framing layer, whereas WebSocket is bidirectional and brings complexity the system does not need. The one genuinely bidirectional surface, the agent chat, is not a live socket at all but a POST whose response is streamed, because the agent endpoint is a request rather than a persistent channel. SSE was therefore used for telemetry, logs and UI state.

3.4.3 VLM inference: self-hosted vLLM with swappable providers

The reading step sends an image of an instrument display to a multimodal model and expects a temperature back. Two considerations shaped the choice of inference. First, the model should not need fine-tuning: the point of the project is to see what prompting alone can achieve with a general-purpose, off-the-shelf model [5, 4]. Second, the model should be easy to swap, because part of the evaluation is to compare several un-fine-tuned models on the same reading task. The platform therefore self-hosts Qwen3.6-35B-A3B through vLLM on an NVIDIA DGX Spark. The same prompt and calling code can be pointed at a third-party vision–language API or a local Ollama instance without change, which turns the choice of provider into an experimental variable rather than a fixed dependency.

3.4.4 Agent Runtime: Lightweight Coding Agent vs. Planner Architecture

Most self-driving laboratories are built on planner-style frameworks, in which the agent first forms a plan and then dispatches it. A lightweight coding agent was chosen instead. The runtime is deliberately minimal — there is no planning module, no state machine and no prompt-orchestration layer, just the agent's read–execute loop and its tools — and two properties of that loop motivated the choice. First, the loop (read a tool's output, then decide the next call) is precisely what hardware control needs, where each step depends on the observed state rather than on a fixed schedule. Second, the Pi Coding Agent's extension mechanism keeps tools and skills outside the agent core and loads them at runtime, which let the hardware-facing tools live in the project rather than in a fork of the agent [8]. The trade-off is that Pi ships with no built-in permission system, so the dangerous tools (`bash`, `edit`, `write`) are disabled and the agent is restricted to a whitelist of domain tools.

The LLM operates primarily at the supervisory level, translating natural-language experimental goals into authorised tool calls. Once an experimental cycle is initiated, timing-critical operations, data acquisition, threshold checks and hardware safety functions are executed deterministically by the backend software.

3.4.5 Multi-camera capture

The design supports up to six synchronised camera streams, although the current rig exercises two (a Logitech C930e and a generic 2K QHD USB webcam). The key requirement, regardless of count, is temporal consistency: a sample must contain one frame from every camera taken at the same moment, otherwise the temperature and the image evidence drift apart. The backend therefore grabs all connected cameras atomically in a single call per sampling tick.

3.4.6 Persistence: queryable SQLite plus a per-run CSV

Telemetry is recorded in two places per sample. A SQLite table is the queryable store, one row per sample, and a per-run CSV file carries the same rows as a plain, portable record a researcher can open directly. The two are written together within the sampling loop, so they stay consistent; the CSV is a live companion to the database rather than a later batch export of it.

3.4.7 Safety: hot-plug monitoring and shutdown

Because the equipment includes a Class 3B laser, safety is a design constraint rather than an afterthought. Each driver runs a hot-plug monitor that watches for its device, and a disconnect triggers an orderly shutdown (laser first, then power) before anything is re-initialised. This is the mechanism behind NFR1. The software shutdown mechanisms and the agent-tool restrictions supplement, rather than replace, the laboratory's physical and procedural laser-safety controls.

3.5 Data Flow

Three flows recur throughout the system. In the *agent flow*, a natural-language request reaches the agent service, which invokes one or more domain tools; each tool calls the backend, optionally pushes a UI event, and returns a text result that the agent weaves into its reply. In the *monitoring flow*, the sampling loop grabs the camera frames, fires the VLM requests concurrently, and writes one telemetry row. In the *UI flow*, anything the backend prints is redirected through the UI bridge and streamed to the front end as a log event. These three flows are all one-way at the point of delivery, which is what makes the SSE choice of Section 3.4.2 hold.

Chapter 4

Implementation

This chapter describes how the design of Chapter 3 was put together. It covers the hardware control layer, the VLM monitoring loop, the agent tool layer and the front end, and closes with the version history that traces the system from prototype to its present form.

4.1 Hardware Control Layer

The hardware layer wraps three kinds of device: a programmable power supply, a laser, and the camera rig. Each is owned by a module-level singleton driver with its own hot-plug monitor thread.

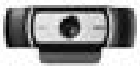
The power supply is an RS PRO RSPD 3303C, a three-channel programmable DC supply, addressed over VISA with PyVISA. The laser is an 808 nm Class 3B diode module mounted on a B71-REV:1.3 driver board. The board provides a TTL/PWM modulation input and is interfaced to the control computer through a CP2102 USB-to-serial adaptor. The camera rig uses the OpenCV DirectShow backend (`CAP_DSHOW`), the Windows-native capture path, and currently consists of two webcams: a Logitech C930e and a generic 2K QHD USB webcam. The design supports up to six streams; the present rig exercises two. Table 4.1 lists the hardware used in the experiments, with a photograph of each device; the instruments themselves are what the vision model must read.

Because the laser is a Class 3B source, the drivers treat disconnection as a safety event. Each hot-plug monitor watches for its device, and a disconnect triggers an orderly shutdown (laser first, then power) before anything is re-initialised. The backend is written to start cleanly with the hardware absent, which is what lets the software be developed away from the rig.

4.2 VLM Monitoring Loop

The monitoring loop is a background thread that fires on a configurable sample interval. Each tick performs three steps. First, it grabs one frame from every connected camera atomically, so

Table 4.1: Hardware inventory of the platform.

Photo	Device	Model	Role
	DC power supply	RS PRO RSPD 3303C	Three-channel programmable supply, set and read over VISA
	Diode laser	808 nm Class 3B diode module (B71-REV:1.3 driver)	Class 3B source, driven over USB-serial
	Thermal imager	Thermal Master (handheld)	Auxiliary reference; its screen is read by the VLM
	Contact thermometer	Seven-segment LCD type	Primary reference; its display is read by the VLM
	Camera 1	Logitech C930e	Faces the contact thermometer
	Camera 2	Generic 2K QHD USB webcam	Faces the thermal imager screen
	USB-serial adaptor	CP2102	Computer interface for the laser driver board

that the temperatures and the image evidence belong to the same instant. Second, it sends each frame to the VLM concurrently, asking for a structured temperature reading. Third, it writes one row — a temperature per camera view, plus the voltage and current from the power supply — to both the per-run CSV file and the SQLite telemetry table.

The VLM requests are routed according to the configured provider. The primary path posts to an OpenAI-compatible `/chat/completions` endpoint served by vLLM from an on-site NVIDIA DGX Spark, with the image embedded as a data URI; the same call can be pointed at a commercial provider or at a local Ollama instance without changing the calling code. The locally served model is Qwen3.6-35B-A3B, deployed through vLLM following the procedure in [9]. The prompt itself is the substance of the reading. Three rules, shown in Listing 4.1, carry the load: the model is told which device it is looking at, is instructed to ignore unit symbols and reflections, and is constrained to a single-decimal numeric output.

Listing 4.1: Structured VLM prompt rules (simplified).

```
# Structured VLM prompt rules (simplified).
#
# [Type-A: Thermal Imager Screen]
#   - Extract ONLY the main temperature at the crosshair.
#   - The screen may be rotated; locate the readout regardless of
   orientation.
#   - Ignore the 'Max' and 'Min' values shown on the left.
#
# [Type-B: Seven-Segment LCD Thermometer]
#   - Extract the main 7-segment number in the centre.
#   - The display may be side-placed or upside-down; read through
   any shadow
#     on the digits.
#   - Ignore unit symbols (°C, °F) and reflections.
#
# Output: a single floating-point number with one decimal place,
   and nothing
# else --- no words, units, spaces or line breaks.
# Expected output:
{"temperature": 39.5}
```

4.3 Agent Tool Layer

The agent service runs a Pi Coding Agent session inside a Node.js server and streams its progress as SSE events (`text_delta`, `tool_start`, `tool_end`, `done`, `error`). The agent is given a set of domain tools that live in the project's extension directory: power, laser, camera, plan, experiment

and the autonomous loop. Each tool follows the same shape: it calls the backend through a shared `callLabBackend()` helper, optionally pushes a UI event through `notifyFrontendUI()`, and returns a text result that the agent weaves into its reply. The server builds its allow-list by scanning the extension directory, and the general-purpose tools (`bash`, `edit`, `write`) are disabled so that the agent can touch the hardware but not the machine around it.

4.4 Frontend

The front end is a single Vue 3 component. Live telemetry and UI state arrive over two `EventSource` connections to the backend's stream bridge, while the agent chat is handled separately with `fetch` and a `ReadableStream` parser, because the agent endpoint is a POST rather than a persistent channel and therefore cannot use `EventSource` directly. The charts are ECharts panels resized through a `ResizeObserver`, so they track the window without the polling timers a manual resize would have needed.

4.5 Iterative Development

The system grew in three stages, summarised in Table 4.2. The first was a Streamlit proof of concept that validated the core idea: a single thermal stream, a basic VLM prompt, and temperature logging. The second rebuilt the interface as the custom full-stack application of Section 3.4.1, added the hardware control panels and moved to automated CSV logging with live charts. The third brought the pieces together: synchronised multi-camera capture, the autonomous agent and its tool layer, and project planning.

Table 4.2: The three development stages of the Thermal Lab Agent.

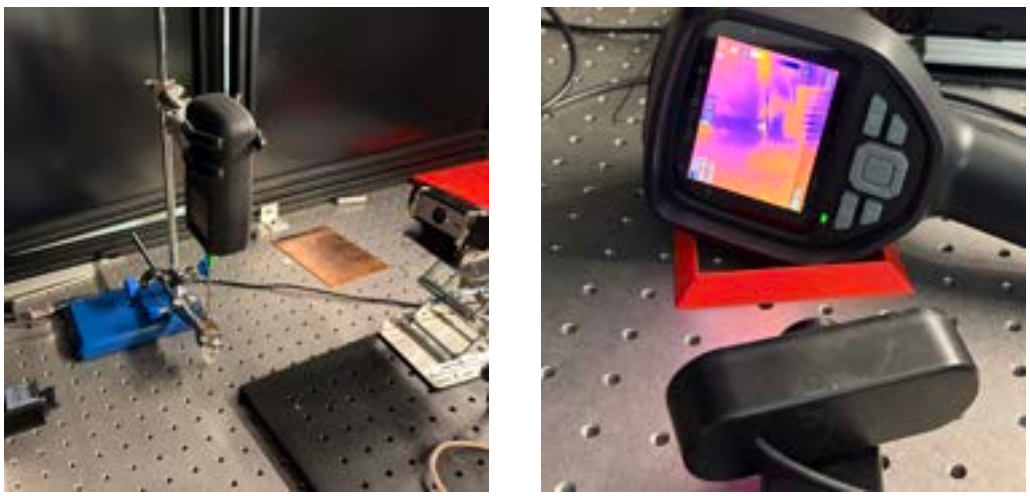
Stage	**Month**	**Capabilities**
Stage 1	June	Streamlit PoC; Single thermal stream; Basic VLM prompts; Telemetry logging
Stage 2	July	Custom full-stack UI; Hardware control panels; CSV logging; Live charts
Stage 3	August	Synchronised multi-camera capture; Autonomous agent; Project planning

Chapter 5

Results and Testing

5.1 Evaluation Methodology

The platform was exercised on a series of autonomous heat–cool runs, each repeating the same protocol: 10 min of laser heating followed by 15 min of cooling, sampled every 1 min. Two carbon nano-onion (CNO) aqueous dispersions were heated: a benzene-derived CNO (BENZ-CNO) sample in the single-cycle run of Section 5.2, and a boron/nitrogen co-doped CNO (B/N-CNO) sample in the three-round run of Section 5.4, whose different photothermal conversion produced a slower heating curve. For the accuracy comparison the VLM readings were checked against manual readings of the same displays, made once per minute by an operator; the room temperature (21 °C) was recorded once before each run. The figures below report the raw extracted temperatures without post-processing. The experimental rig is shown in Figure 5.1.



(a) Contact thermometer (b) Thermal imager

Figure 5.1: The experimental rig. (a) The stand holds the seven-segment contact thermometer and the sample solution, with a camera facing the readout and the laser positioned to heat the sample. (b) The thermal imager beside the stand measures the sample centre as an auxiliary reference, with a second camera reading its screen.

5.2 VLM Reading of Thermal Instrument Displays (RQ1)

The first question is whether the prompt-engineered VLM can read the displays reliably. Table 5.1 lists six representative samples from one round of the B/N-CNO run, with the VLM reading next to the manual reading of the same display, and Figure 5.2 shows the full 26-point traces of that round. The contact thermometer — a stable, high-contrast, contact-based display — was read without a single outlier: no dropped decimal points, no duplicated values and no spikes; the thermal imager, whose readout fluctuates as its screen digits change, still tracks the manual readings closely.

Table 5.1: Manual versus automated (VLM) temperature readings from one round of the B/N-CNO run.

	Thermal imager (°C)		Thermometer (°C)	
Time (min)	**Manual**	**VLM**	**Manual**	**VLM**
0	22.6	22.8	21.5	21.5
5	26.3	25.9	26.0	26.1
10	27.5	27.5	28.0	28.0
15	24.7	24.8	24.2	24.2
20	23.5	23.5	22.4	22.4
25	22.4	22.4	21.6	21.6

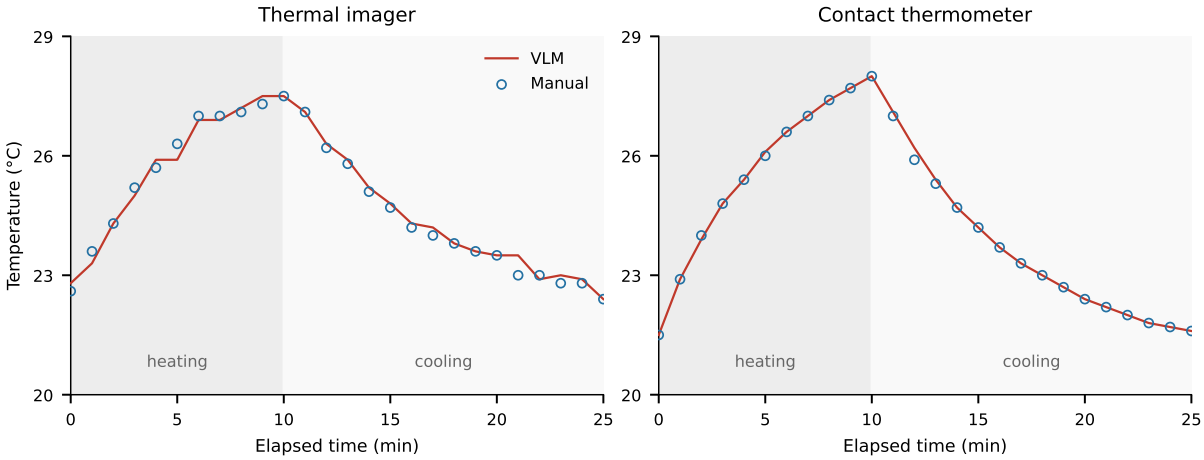


Figure 5.2: The full trace of the round summarised in Table 5.1: VLM readings (lines) and manual readings (circles) for the thermal imager (left) and the contact thermometer (right). The thermal imager's trace fluctuates as its display digits change, whereas the contact thermometer moves smoothly, yet both agree point for point.

One failure mode remained, and it originated in the capture pipeline rather than in the sample or the prompt. During one run the camera was switched to a reduced resolution, and because the thermal imager's readout changes from one digit to the next far more often than the contact thermometer's, the display was repeatedly captured mid-transition; at those instants the superimposed ghost of the previous digit caused a misread: 32.8 °C where the true value was 32.0 °C

at 7 min, and 26.8 °C where the true value was 26.3 °C at 19 min, so the curve appears to rise briefly instead of falling (Figure 5.3). The contact thermometer, whose display changes slowly, was unaffected, and restoring the original capture resolution eliminated the ghosting (Figure 5.4).

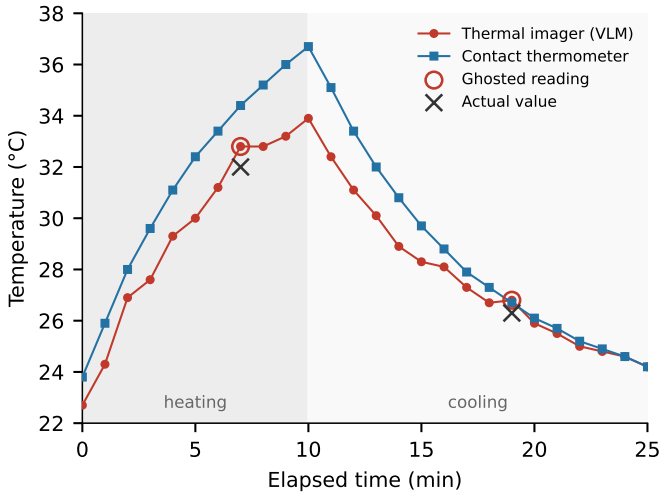


Figure 5.3: A heating–cooling cycle recorded on the BENZ-CNO sample over 10 min of heating and 15 min of cooling, sampled every 1 min. Open circles mark two ghosted thermal-imager readings (32.8 °C at 7 min and 26.8 °C at 19 min); the grey crosses mark the true values (32.0 °C and 26.3 °C) read by eye.

(a) 7 min (ghosted) (b) 8 min (clean) (c) 18 min (clean) (d) 19 min (ghosted)

Figure 5.4: Camera frames underlying the ghosted readings in Figure 5.3.

5.3 End-to-End Coding-Agent Control (RQ2)

The experiments were not scripted by hand. The three-round run of Section 5.4 was started by a natural-language instruction to the agent — heat 10 min, cool 15 min, sample every 60 s, start the first round immediately and each following round once the sample is below 25 °C — and the agent translated it into the corresponding tool calls that drove the power supply, the laser and the sampling loop. Figure 5.5 shows the front end during the run, with the conversation on the right and the live telemetry on the left.

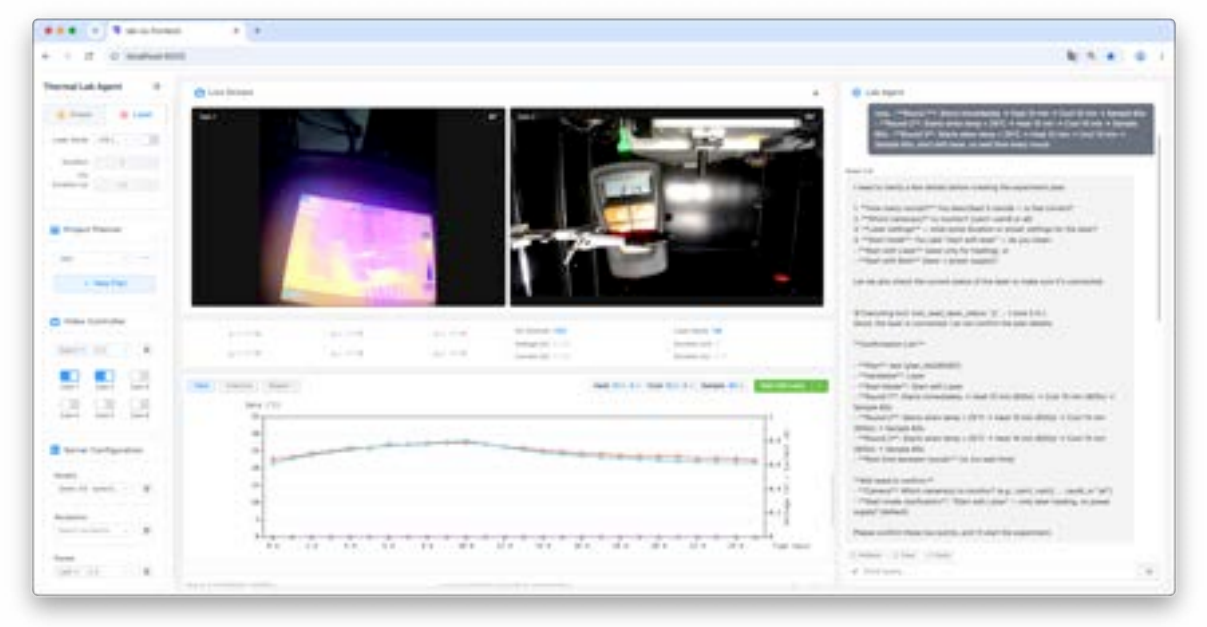


Figure 5.5: The front end of the runtime with the integrated agent: the natural-language conversation with the agent (right) sits alongside the live telemetry (left).

5.4 Autonomous Heat–Cool Loop (RQ3)

The loop then ran repeatedly without intervention: three consecutive rounds, the first starting immediately and the next two starting once the sample had cooled below 25 °C. The contact thermometer is used as the example here because its stable, high-contrast display provides the cleanest test of reading; the thermal imager's readout fluctuates (Section 5.2), so it is not a stable reference for this comparison. Figure 5.6 shows the contact thermometer trace of each round: all three follow the same rise-and-fall shape with no outliers, and the manual readings (circles) sit on the VLM trace.

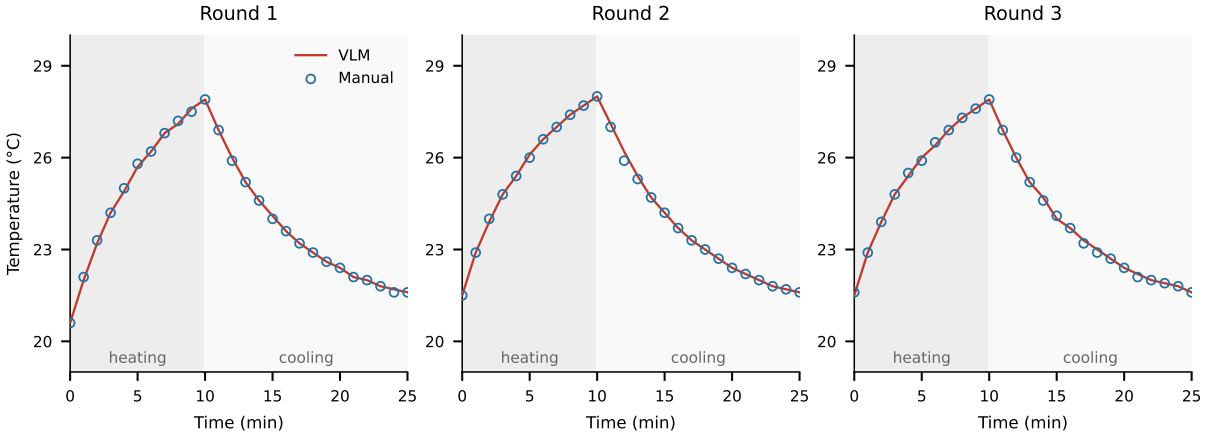


Figure 5.6: Three consecutive autonomous heat–cool rounds. Each panel overlays the VLM thermometer trace (line) with the manual readings (circles).

5.5 Automated versus Manual Measurement (RQ4)

Because the manual reading itself carries timing and parallax uncertainty, it is treated as a reference rather than as an error-free ground truth, and the agreement between the two methods is quantified with the Bland–Altman limits of agreement and Lin's concordance correlation coefficient. Writing V_i for the VLM reading and M_i for the manual reading at the i-th sample, the mean bias $\bar{d}$ and the 95 % limits of agreement (LoA) are

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n}(V_i - M_i), \qquad \text{LoA} = \bar{d} \pm 1.96\, S_d, \tag{5.1}$$

where S_d is the standard deviation of the differences, and Lin's concordance correlation coefficient is

$$\rho_c = \frac{2\,\sigma_{vm}}{\sigma_v^2 + \sigma_m^2 + (\mu_v - \mu_m)^2}, \tag{5.2}$$

where μ_v, μ_m are the means, σ_v^2, σ_m^2 the variances of the VLM and manual readings, and σ_{vm} their covariance. Across the 78 samples spanning the three rounds, $\bar{d} = +0.006$ °C and the 95 % LoA were ±0.12 °C — within the thermometer's 0.1 °C display resolution — and $\rho_c = 0.9995$, indicating almost perfect agreement (Figure 5.7). Under the tested experimental conditions, the multimodal-model readings agreed sufficiently closely with manual readings to replace manual display transcription for this thermal experiment.

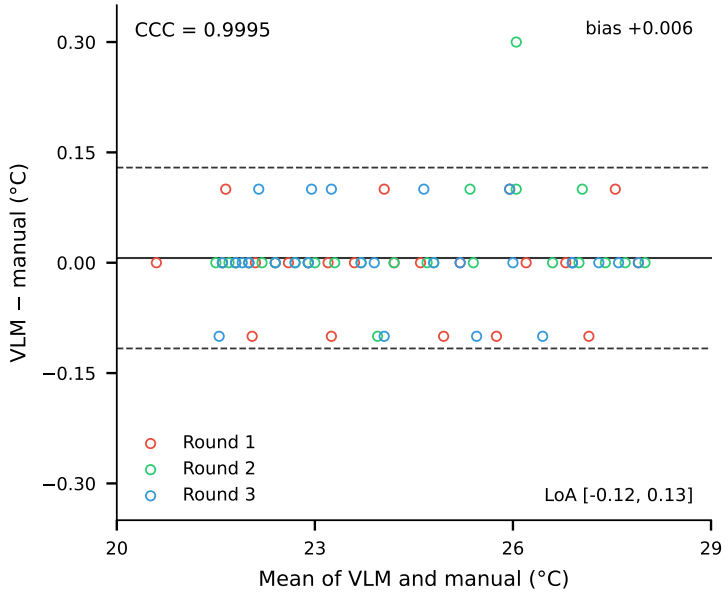


Figure 5.7: Bland–Altman plot of the difference (VLM minus manual) against the mean of the two readings, for all three rounds. The solid line is the mean bias and the dashed lines the 95 % limits of agreement.

Chapter 6

Evaluation and Discussion

6.1 RQ1: VLM Reading of Thermal Instrument Displays

The prompt-engineered VLM read the displays reliably: the seven-segment thermometer readings contained no outliers across the B/N-CNO run (Section 5.2) and agreed with the manual reference (Section 5.5). One failure mode did remain: when the camera capture was temporarily reduced to 720p, the thermal imager's display was captured mid-transition and the VLM misread the digit ghost (Section 5.2). This was traced to the capture resolution rather than to the prompt, and reverting to the original resolution eliminated it. RQ1 is therefore supported, with digit-transition ghosting under degraded capture as the one identified failure mode.

6.2 RQ2: End-to-End Coding-Agent Control

The three-round experiment was started by a natural-language instruction to the agent, which translated it into the tool calls that drove the power supply, the laser and the sampling loop (Section 5.3). The front-end record of the session (Figure 5.5) shows the instruction being issued and the run proceeding, with no hand-scripted commands in between. RQ2 is therefore supported: the agent drove the pipeline end to end from natural language.

6.3 RQ3: Autonomous Heat–Cool Loop

Three consecutive heat–cool rounds ran to completion without intervention (Figure 5.6): the first started immediately, and the next two started once the sample had cooled below 25 °C, each heating for 10 min and cooling for 15 min with 1 min sampling. Every round produced a complete, valid dataset, and the contact-thermometer traces were consistent across rounds. RQ3 is therefore supported.

6.4 RQ4: Automated versus Manual Measurement

The manual readings serve as a reference against which the VLM's stability is checked, rather than as an error-free ground truth: a human operator can only align readings to the sampling interval approximately and reads with some delay, so an exact match is not expected. Across the 78 readings spanning the three rounds, the VLM and manual thermometer readings agreed almost perfectly: the mean bias was +0.006 °C and the 95 % limits of agreement were ±0.12 °C, which lies within the thermometer's 0.1 °C display resolution, with Lin's concordance correlation coefficient 0.9995 (Section 5.5). Under the tested experimental conditions, the Qwen3.6-35B-A3B readings agreed sufficiently closely with manual readings to replace manual display transcription for this thermal experiment. The thermal imager, whose readout fluctuates, agreed to within 0.5 °C. RQ4 is therefore supported for measurement agreement.

6.5 Discussion

The contrast of Section 2.6 can now be settled. A LabVIEW or SCPI system would have run the same protocol only after an engineer scripted every setpoint in advance, offering a panel of controls rather than a conversation. Here, one natural-language instruction was translated into the tool calls that drove the supply, laser and sampling loop, and the loop planned its own transitions from the temperatures it observed. That is what a language model adds: reasoning between an instruction and the hardware, instead of a fixed procedure.

The trade-off is just as clear. A LabVIEW program is deterministic and repeatable, which is why test and measurement still relies on it, whereas an LLM-based agent is more flexible but less predictable, and its reading stays sensitive to capture quality, as the ghosting failure showed. Its value is therefore not that it replaces scripted automation everywhere, but that it closes the one gap where a person was indispensable: reading an instrument display and acting on it.

Several practical questions follow from the results. The most important is what happens when the model returns a plausible but wrong temperature: the digit-transition ghosting of Section 5.2 is exactly this, and the mitigation is not to trust a single reading. Repeated readings, cross-checking the two instruments and simple validation rules would catch most such errors; making the agent carry out this re-checking autonomously is the future direction developed in Section 6.7. A second concern is the network dependency between the Mac Mini and the DGX Spark: if that link fails, inference stops, so the sampling loop should fail safe and retain the last valid state rather than writing a default value. Inference latency also sets a lower bound on the practical sampling interval — a one-minute cycle is far above it, but a high-frequency experiment would need the latency measured and budgeted.

The reading is also sensitive to the camera view: angle, glare, illumination, resolution and display orientation all affect accuracy, and the ghosting failure showed how strongly resolution matters, so any new rig would need its cameras re-verified under its own lighting. The approach

generalises to other instruments in a straightforward way — a new device needs a camera aimed at its display and a short prompt describing how to read it — which is why the same tooling could cover oscilloscopes, PID controllers and spectrophotometers with little extra engineering. Whether the results transfer to those instruments, or to other models, remains open and is exactly what the swappable provider interface is designed to test.

6.6 Threats to Validity

The main threats concern the reference against which the readings are judged. The manual readings are themselves imperfect: a human operator can only align a reading to the sampling interval approximately and reads with a short delay, so the manual reference carries its own uncertainty (internal validity). The thermal imager's readout fluctuates even at steady state, so it is not an ideal ground truth; this is why the stable seven-segment thermometer is used as the primary reference (construct validity). The thermal-imager reading also depends on emissivity and viewing distance, which were not independently calibrated here, so its absolute values are indicative rather than traceable. The comparison is drawn from a single laboratory, a single rig and one model of each instrument, so the absolute figures may not transfer to other hardware (external validity). Finally, the digit-transition ghosting observed under 720p capture was resolved by reverting the resolution, but it shows that the reading is sensitive to capture quality and would need re-checking on any future hardware change.

6.7 Improving the Agentic Capability of the Thermal Lab Agent

The current Thermal Lab Agent demonstrates that a coding-agent architecture can translate natural-language instructions into physical laboratory actions through a restricted set of domain-specific tools. This is a strong foundation, but the present agent mainly acts as a supervisory interface and tool orchestrator. Once a heat–cool protocol has been specified, much of the actual execution is handled by deterministic backend logic rather than continuous scientific reasoning by the agent. The current system can therefore be viewed as agent-mediated laboratory automation, rather than a fully autonomous scientific agent.

A major direction for future work is to move towards closed-loop experimental reasoning. Instead of asking the user to provide a complete procedure such as "heat for 10 min, cool for 15 min and repeat three times", the user could provide a higher-level goal such as: "characterise the thermal response of this sample". The agent could then perform a short exploratory run, analyse the heating rate, decide whether to extend or shorten the heating period, determine when sufficient cooling has occurred, and select the number of repeat measurements required. This would allow the agent to construct and revise an experimental plan based on observations rather than simply executing a predefined protocol.

Another important improvement is observation-driven replanning and anomaly handling. The digit-transition ghosting identified in the present work provides a useful example. If the agent observes an isolated temperature value that is inconsistent with the local trend or disagrees strongly with the second instrument, it could automatically request another image, repeat the multimodal reading, compare the two sensors and decide whether the original value should be retained. For example, an unexpected sequence such as 31.8 °C → 32.0 °C → 32.8 °C → 32.1 °C could trigger a re-measurement rather than being accepted immediately. This would make the agent an active participant in measurement quality control.

The system could also benefit from experimental memory. Because measurements are already stored in SQLite and CSV files, future versions could retrieve previous runs when planning new experiments. For example, if earlier B/N-CNO samples consistently reached approximately the same peak temperature after 10 min, the agent could use this history to propose a more suitable protocol for the next run. This would allow the system to improve its decisions from accumulated experimental experience.

A further step would be to integrate automatic scientific analysis into the agent loop. After each run, the agent could calculate the peak temperature, temperature rise, heating and cooling rates, repeatability and agreement between the two temperature measurements. These results could then influence the next experiment. If two runs agree closely, the agent might conclude that sufficient repeatability has been achieved; if they differ significantly, it could automatically schedule an additional run.

As autonomy increases, the system should retain the principle of flexible reasoning but constrained execution. The LLM may propose experimental actions, but deterministic control and safety layers should decide whether those actions are permitted. The existing restriction to approved domain tools is therefore an important design feature that should be retained. A future system could further separate planning, experiment execution, data analysis and safety verification, potentially using a hierarchical or multi-agent architecture when the experimental tasks become more complex.

A useful way to describe the progression is:

- **Level 1 – Scripted automation:** a fixed software procedure controls the experiment.
- **Level 2 – Agent-mediated automation:** the current system, where natural-language instructions are translated into approved tool calls.
- **Level 3 – Adaptive experimental agent:** the agent observes results, detects anomalies and modifies the ongoing procedure.
- **Level 4 – Goal-driven scientific agent:** the user specifies a scientific objective, while the agent plans experiments, analyses results, evaluates uncertainty and selects the next experiment.

The current project establishes much of the infrastructure required to progress towards Levels 3 and 4, including hardware control, multimodal instrument reading, persistent telemetry and a domain-specific agent tool layer. The most valuable future development would therefore be to add adaptive replanning, anomaly detection, experimental memory, automatic analysis and goal-driven experiment design. This would move the Thermal Lab Agent from a natural-language interface for laboratory automation towards a more capable self-driving experimental system.

Chapter 7

Conclusion and Future Work

7.1 Summary

This project set out to remove the human operator from the loop of thermal experiment monitoring. The result is the Thermal Lab Agent: a platform integrating an agent, in which a Python FastAPI backend owns the power supply, laser and camera rig, a Node.js service hosts the Pi Coding Agent extended with a domain tool layer, and a Vue 3 web front end presents the live telemetry. A prompt-engineered, self-hosted vision–language model reads the displays of thermal instruments (the thermal imager's on-screen readout and a seven-segment contact thermometer) without any fine-tuning.

The four research questions are answered as follows. RQ1 is supported: the Qwen3.6-35B-A3B model read the displays reliably, with digit-transition ghosting under degraded capture as the one identified failure mode, resolved by reverting the capture resolution. RQ2 is supported: the agent drove the experiment end to end from a natural-language instruction. RQ3 is supported: three consecutive heat–cool rounds ran to completion without intervention. RQ4 is supported: across 78 samples the Qwen3.6-35B-A3B and manual thermometer readings agreed sufficiently closely to replace manual display transcription for this thermal experiment (mean bias +0.006 °C, 95 % limits of agreement ±0.12 °C, Lin's concordance coefficient 0.9995).

7.2 Contributions Revisited

The principal contribution is the integration rather than any single algorithm: a working LLM-based laboratory agent that translates natural-language experimental instructions into authorised hardware-control operations; a multimodal visual measurement approach using Qwen3.6-35B-A3B to read temperatures directly from instrument displays; a distributed deployment in which the Thermal Lab Agent runs on a Mac Mini while multimodal inference is served locally from an NVIDIA DGX Spark; synchronised multi-camera measurement, telemetry persistence and autonomous heat–cool orchestration; and experimental validation against manual readings over

repeated heat–cool cycles.

7.3 Future Work

Three directions follow naturally. First, the reading could be hardened against capture-quality effects, for example by detecting digit-transition frames and re-reading them rather than relying on a fixed capture resolution. Second, the swappable provider interface invites a systematic comparison of several vision–language models on the same reading task, which would quantify how much the choice of model matters. Third, the platform could be extended to further instruments (oscilloscopes, PID controllers, spectrophotometers) whose displays present the same reading problem.

Bibliography

[1] I. Mandal, J. Soni, M. Zaki, M. M. Smedskjaer, K. Wondraczek, L. Wondraczek, N. N. Gosvami, and N. M. A. Krishnan, "Evaluating large language model agents for automation of atomic force microscopy," *Nature Communications*, vol. 16, p. 9104, 2025.

[2] S. Cao, Z. Zhang, M. Alghadeer, S. D. Fasciati, M. Piscitelli, M. Bakr, P. Leek, and A. Aspuru-Guzik, "Automating quantum computing laboratory experiments with an agent-based AI framework," *Patterns*, vol. 6, no. 10, p. 101372, 2025.

[3] Y. Gao, Y. Luo, Y. Lan, H. Jiang, L. Fu, and T. Si, "Autonomous liquid-handling robotics scripting through large language models enables accessible and safe protein engineering workflows," bioRxiv preprint, 2025.

[4] F. Lin, Y. Liu, H. Xu, Y. Chen, Z. He, M. Zhao, M. H. Chen, J. Liu, J. G. Yao, and X. Yang, "Do vision-language models measure up? benchmarking visual measurement reading with MeasureBench," arXiv preprint 2510.26865, 2025.

[5] J. Wang, X. Li, and W. Shen, "A staged PEFT framework for industrial pointer-gauge reading with multimodal large language models," *Applied Sciences*, vol. 16, no. 16, p. 7924, 2026.

[6] N. Mahjourian and V. Nguyen, "Vision-language models for infrared industrial sensing in additive manufacturing scene description," arXiv preprint 2512.11098, 2025.

[7] M. Salah, E. Ouda, G. Dell'Avvocato, F. Sarasini, E. D'Accardi, J. Dias, D. Svetinovic, S. Sfarra, and Y. Abdulrahman, "Towards cognitive defect analysis in active infrared thermography with vision-text cues," arXiv preprint 2603.10549, 2026.

[8] Earendil Works, "Pi coding agent," 2026. [Online]. Available: https://github.com/earendil-works/pi

[9] Z. Wang, "DGX spark vLLM deployment for Qwen3.6-35B-A3B-DFlash," GitHub repository, 2026. [Online]. Available: https://github.com/ZengboJamesWang/dgx-spark-vllm-qwen3.6-35b-a3b-dflash

Appendix A

Continued Development Evidence

A.1 Git Commit Timeline

The project was developed and maintained in a version-controlled repository at `https://github.com/au971/Lab-agent` (a private repository; the complete source code is provided with the project submission), which provides timestamped evidence of continued development: over eighty commits were made between June and August 2026. A representative selection spanning the project is listed in Table A.1.

Table A.1: Selected commits demonstrating continued development.

Date	Commit	Summary
2026-06-11	a8f5192	Project focus: laboratory agent
2026-06-14	e9c8d8c	Python control of a programmable DC power supply
2026-06-15	a74b180	Connect a locally deployed LLM to the front end
2026-06-17	be18b12	Embed mobile video streams into a web page
2026-06-22	5c773a1	Migrate from Streamlit to a frontend-backend architecture
2026-06-24	b0cb89e	Introduce a VLM to recognise thermal imaging data
2026-07-15	ca86466	Batch generation for chart and table views
2026-07-16	661946f	Align the VLM timeline with disk-based history
2026-07-18	bb1a534	Optimise multi-camera stream concurrency and timing
2026-07-25	5269395	Video stream toggling control
2026-07-29	93ba5e6	Laser control functionality
2026-07-31	39d74f4	Fix laser control logic and dashboard display
2026-08-11	f49b8d8	Fix timing drift in the VLM monitoring loop
2026-08-15	b8465a5	Node-based Lab Agent with UI state linkage
2026-08-19	33e6b27	Autonomous loop orchestrator, agent tools and skills
2026-08-19	77bc94a	Guard: restrict the agent from modifying loop core logic
2026-08-20	da4c2a2	Split experiment orchestration from VLM, fix sampling timing
2026-08-20	9bc5b2a	Revert to original camera capture (720p caused ghosting)
2026-08-21	92f26d3	Upside-down display type in the VLM prompt
2026-08-22	50bbb78	Per-camera 90° rotation for VLM input
2026-08-22	8c5bad5	Unify chart resizing through ResizeObserver

A.2 Supervisor Meetings Log

Table A.2: Supervision meetings and lab sessions across the project.

Date	**Session**
5 June 2026	Project meeting
11 June 2026	Project meeting
18 June 2026	Lab session
23 June 2026	Lab session
6 July 2026	Lab session
22 July 2026	Lab session
24 July 2026	Lab session
11 August 2026	Progress presentation