



VRIJE
UNIVERSITEIT
BRUSSEL



Master thesis submitted in order to be awarded the degree of
Master of Science in Engineering Technology:
Electromechanical Engineering - aeronautics

STANDALONE ON-SITE LIGHTING MEASUREMENTS FOR INSECT STUDIES

Zakaria El Gharbaoui

2024-2025

promoter: prof. dr. V.A. Jacobs
supervisor: Arjen Mentens

ENGINEERING SCIENCES

Acknowledgements

The completion of this thesis would not have been possible without the help and support of several individuals.

First of all, I would like to express my sincere gratitude to my supervisor, Arjen Mentens, for his continuous guidance throughout the development of this work. His technical expertise and constructive feedback were invaluable in shaping the standalone on-site monitoring unit and in refining the broader research approach. I am particularly thankful for his availability, patience, and clear direction. I would also like to thank my promotor, Prof. Dr. V.A. Jacobs, for giving me the opportunity to pursue this thesis within the MERLIN research group at VUB. Her encouragement and trust allowed me to explore an interdisciplinary topic at the intersection of technology and ecology.

My appreciation extends to the MERLIN team for providing a stimulating environment that enriched this thesis. Furthermore, I thank the jury members for taking the time to evaluate my work and for their thoughtful feedback.

Last but not least, I would like to thank my family and friends for their unwavering support and encouragement throughout this journey.

My sincerest gratitude,
Zakaria El Gharbaoui

Ongoing urbanization and the global shift toward energy-efficient LED lighting are intensifying the exposure to artificial light at night, raising concern about its effects on the communities of nocturnal insects. Most existing field studies are short in duration and limited in spatial coverage because of the labor demands of traditional sampling. This thesis addresses this gap by designing, deploying, and validating a low-cost camera-based remote monitoring unit for season-long operation in diverse outdoor environments. The system records weather variables (temperature, relative humidity), location (GPS), lighting parameters (correlated color temperature), and time-stamped images of attracted insects.

To evaluate classification strategies without full-scale deployment, a pilot case study tested two general-purpose vision–language models on curated insect images with authoritative ObsIdentify determinations. The results position vision–language models as effective advisory tools for provisional tagging, triage, and quality assurance, while authoritative labels should come from specialists or dedicated species-recognition services.

Future work should first determine the minimally sufficient taxonomic depth for ALAN impact studies, then adapt recognition tools to meet this requirement. Subsequent steps include extending the pipeline to counting and abundance metrics, developing a robust containerized deployment architecture, migrating the database to secure cloud or NAS infrastructure, and implementing strong cybersecurity measures for remote units. These advances will enable standardized, season-long, multi-site campaigns capable of delivering robust assessments of ALAN impacts under LED-dominated lighting conditions.

keywords: artificial light at night; nocturnal insects; automated monitoring; low-cost sensors; vision–language models; taxonomy; field ecology; AI

Voortgaande verstedelijking en de wereldwijde overstap naar energiezuinige LED-verlichting vergroten de blootstelling aan kunstlicht 's nachts, wat bezorgdheid wekt over de effecten op gemeenschappen van nachtactieve insecten. De meeste bestaande veldstudies zijn kort van duur en beperkt in ruimtelijke dekking vanwege de arbeidsintensieve aard van traditionele bemonstering. Dit proefschrift pakt deze lacune aan door het ontwerpen, implementeren en valideren van een goedkope, op camera's gebaseerde, op afstand werkende monitoringsunit voor seizoenslange inzet in diverse buitenomgevingen. Het systeem registreert weerparameters (temperatuur, relatieve vochtigheid), locatie (GPS), verlichtingsparameters (gecorrleerde kleurtemperatuur) en tijdgestempelde beelden van aangetrokken insecten.

Om classificatiestrategieën te evalueren zonder grootschalige inzet, werd in een kleinschalige casestudy het gebruik van twee algemene vision–language modellen getest op zorgvuldig geselecteerde insectenbeelden met gezaghebbende ObsIdentify-bepalingen. De resultaten positioneren vision–language modellen als effectieve hulpmiddelen voor voorlopige tagging, triage en kwaliteitscontrole, terwijl gezaghebbende labels afkomstig moeten zijn van specialisten of gespecialiseerde soortenherkenningsdiensten.

Toekomstig werk dient eerst de minimaal vereiste taxonomische diepte voor ALAN-impactstudies vast te stellen en vervolgens herkenningstools aan te passen om aan deze eis te voldoen. Verdere stappen omvatten het uitbreiden van de pijplijn naar tellingen en abundantiematen, het ontwikkelen van een robuuste containergebaseerde implementatie-architectuur, het migreren van de database naar een beveiligde cloud- of NAS-infrastructuur en het implementeren van sterke cyberbeveiligingsmaatregelen voor externe units. Deze verbeteringen zullen gestandaardiseerde, seizoenslange, multisitecampagnes mogelijk maken die robuuste beoordelingen van ALAN-effecten onder door LED gedomineerde lichtomstandigheden opleveren.

keywords: kunstlicht 's nachts; nachtactieve insecten; geautomatiseerde monitoring; goedkope sensoren; vision–language modellen; taxonomie; veldecologie; AI

L'urbanisation croissante et la transition mondiale vers un éclairage LED économe en énergie intensifient l'exposition à la lumière artificielle nocturne, suscitant des préoccupations quant à ses effets sur les communautés d'insectes nocturnes. La plupart des études de terrain existantes sont de courte durée et limitées en couverture spatiale en raison des contraintes de main-d'œuvre liées aux méthodes d'échantillonnage traditionnelles. Cette thèse comble cette lacune en concevant, déployant et validant une unité de surveillance à distance, peu coûteuse et basée sur caméra, pour un fonctionnement saisonnier dans divers environnements extérieurs. Le système enregistre les variables météorologiques (température, humidité relative), la localisation (GPS), les paramètres d'éclairage (température de couleur corrélée) et des images horodatées des insectes attirés.

Pour évaluer les stratégies de classification sans déploiement à grande échelle, une étude pilote a testé deux modèles vision-langage génériques sur des images d'insectes sélectionnées avec déterminations de référence issues d'ObsIdentify. Les résultats montrent que ces modèles constituent des outils efficaces pour l'étiquetage provisoire, la hiérarchisation et l'assurance qualité, tandis que les identifications faisant autorité doivent provenir de spécialistes ou de services dédiés à la reconnaissance des espèces.

Les travaux futurs devraient d'abord déterminer la profondeur taxonomique minimale nécessaire aux études d'impact de l'ALAN, puis adapter les outils de reconnaissance pour répondre à cette exigence. Les étapes suivantes incluent l'extension de la chaîne de traitement au comptage et à des métriques d'abondance, le développement d'une architecture de déploiement robuste basée sur la conteneurisation, la migration de la base de données vers une infrastructure sécurisée de type cloud ou NAS, ainsi que la mise en place de mesures de cybersécurité renforcées pour les unités déployées sur le terrain. Ces avancées permettront de réaliser des campagnes standardisées, saisonnières et multi-sites, capables de fournir des évaluations fiables des effets de l'ALAN dans des environnements d'éclairage dominés par les LED.

keywords : lumière artificielle nocturne ; insectes nocturnes ; surveillance automatisée ; capteurs à faible coût ; modèles vision-langage ; taxonomie ; écologie de terrain ; IA

Disclosure on the use of AI

Declaration

During the preparation of this work, the author(s) used the following tool(s) and service(s):

name of tool/service	reason
<i>ChatGPT</i>	in order to: <i>Text revision, editorial support, assistance in coding</i>
<i>ChatGPT</i>	in order to: <i>Translate research queries for the different databases</i>

After using this tool(s) and/or service(s), the author(s) reviewed and edited the content as needed. The author(s) take(s) full responsibility for the content of this work.

Contents

Acknowledgements	i
Abstract	ii
Samenvatting	iii
Resumé	iv
Disclosure on the use of AI	v
Contents	vi
List of Figures	x
List of Tables	xii
Listings	xiii
Nomenclature	xiv
1 Introduction	1
1.1 Context	1
1.2 Background and significance	1
1.2.1 Artificial Light at Night	1
1.2.2 Lighting technologies	1
1.2.3 Nocturnal insects	2
1.2.4 The need for long-term, scalable insect monitoring	3
1.3 Research question	4
1.3.1 Sub-question	4
1.4 Objectives	5
1.5 Thesis structure	5
2 Literature review	7
2.1 Search methodology	7
2.1.1 Eligibility Criteria	7
2.1.2 Database Selection	8

2.1.3	Search Strategy	9
2.1.4	Database-Specific Adaptations	9
2.1.5	Paper Selection and Final Dataset	10
2.2	Synthesis and conclusions	11
2.2.1	Review synthesis methodology	11
2.2.2	Research scope	11
2.2.3	Key LED parameters examined	12
2.2.4	Effects on insect abundance	12
2.2.5	Effects on community composition	13
2.2.6	Evidence and knowledge gaps	13
2.2.7	Conclusion	13
3	Materials and Methods	15
3.1	System Architecture and Overview	15
3.2	Field layer: On-site monitoring unit	16
3.2.1	Functional objectives and requirements	16
3.2.2	Physical setup	17
3.2.3	Electronic design details	18
3.2.3.1	Power delivery and conversion	18
3.2.3.2	Arduino and sensor circuit	18
3.2.4	Connectivity	21
3.2.5	Sensing, data acquisition and transmission	23
3.2.5.1	Sensing	23
3.2.5.2	Data acquisition	24
3.2.5.3	Data transmission	25
3.2.5.4	Automation	26
3.2.6	Setup Manual	27
3.3	Central Layer: Central Computing Unit	37
3.3.1	Functional objectives and requirements	37
3.3.2	Physical and software setup	37
3.3.3	Database	38
3.3.3.1	DBMS: Database Management System	39
3.3.3.2	Layout, structure and relations of the database	40
3.3.4	Data flow	41
3.3.4.1	Raw data reception and storage	41
3.4	Cloud-layer: Recognition API	43
3.4.1	Architecture and function	43
3.4.2	Recognition APIs Used	43
3.4.3	Data Flow and Logging Strategy	44
3.4.4	Limitations and motivation for AI integration	44
3.4.5	Case study: Evaluating general-purpose AI for insect recognition	44

4	Case Study	47
4.1	Comparative performance of general-purpose vision models for insect recognition	47
4.1.1	Objective	47
4.1.2	Dataset	47
4.1.3	Scoring and metrics	48
4.1.4	Results	48
4.1.4.1	Full-resolution: 1000×1000 px	48
4.1.4.2	Effect of downscaling	49
4.1.4.3	Limitations	49
4.1.4.4	Statistical significance: RM-ANOVA test	51
5	Discussion	52
5.1	What the results mean for the project	52
5.1.1	Capability envelope	52
5.1.2	Resolution matters	52
5.1.3	Balancing AI assistance and expert validation	52
5.1.4	Auto-critical reflections	52
5.1.4.1	Case-study scope	53
5.1.4.2	Prompting assumptions	53
5.1.4.3	API instability	53
5.1.4.4	Strategic pivot	53
5.1.4.5	Unknown necessary taxonomic depth	54
5.1.5	Low-cost design: components and costs	54
6	Future work	55
6.1	Future work	55
6.1.1	Clarifying the required taxonomic depth	55
6.1.1.1	Targeting appropriate recognition tools	55
6.1.1.2	Extending recognition to counting	55
6.1.1.3	Developing an abundance metric	55
6.1.1.4	Production-ready deployment architecture	56
6.1.1.5	Secure and accessible data infrastructure	56
7	Conclusions	57
7.1	Summary of contributions	57
7.1.1	A low-cost, field-ready platform	57
7.1.2	Operational workflow and governance	58
7.1.3	Imaging resolution and VLM performance in context	58
7.2	Answer to the sub-question	58
7.3	Progress toward the main question	59
7.4	Objective-by-objective reflection	59

<i>Contents</i>	ix
A RM-ANOVA test results	61
A LTE modem	63
A.1 Modem boot and network access workflow	63
A Database structure	66
B Codes	69
Bibliography	91

List of Figures

1.1	Relative spectral power distributions of HPS and LED lamps across the visible spectrum (380 nm–780 nm, 5 nm increments), reproduced from CIE 015:2018 (HPS: Table 11; LED: Tables 12.1/12.2) [14, 13]. The HPS source displays a narrow emission band peaking in the yellow–orange region, while the LED illuminant exhibits a broader profile characterized by a dominant blue spike and extended phosphor tail.	2
3.1	Schematic overview of the implemented system showing major components and data flow.	15
3.2	Schematic and actual setup of the on-site monitoring unit. The left panel shows the functional layout, while the right panel illustrates the physical deployment.	16
3.3	Schematic of the PCB powering and connecting the Arduino Uno to the temperature and humidity sensor.	18
3.4	Designed PCB layout.	19
3.5	Power distribution tree of the Arduino Uno R3. The 3.3 V line is generated by a dedicated LDO regulator powered by the main 5 V rail. This configuration reduces voltage ripple and noise, improving stability for sensors [2].	20
3.6	Typical application circuit for the AHT20 sensor, including pull-up resistors on the I ² C lines (4.7 k Ω), a decoupling capacitor (10 nF) to ground, and a series resistor (390 Ω) between the VDD line and the sensor. These elements collectively form a basic RC low-pass filter to mitigate power line noise and enhance signal stability. Adapted from the AHT20 datasheet [1].	20
3.7	Schematic representation of data flow from the SBC to the internet. The CPU generates a digital bitstream, which is modulated by the cellular radio chip into an analog signal. This signal is transmitted via the antenna, received by the cell tower, and routed through the carrier’s infrastructure to reach the internet.	21
3.8	CLI command output showing the virtual serial interfaces forwarded by the LTE modem over USB to the SBC.	22
3.9	Flashing the operating system using Raspberry Pi Imager.	28
3.10	Optional pre-configuration and confirmation steps (part 1).	28

3.11	Optional pre-configuration and confirmation steps.	29
3.12	Tailscale installation options on the Raspberry Pi.	31
3.13	Checking whether the 4G modem is detected by the Raspberry Pi.	33
3.14	Running basic AT commands inside a <code>minicom</code> session.	34
3.15	Accessing the repository content on the Raspberry Pi after cloning.	36
3.16	Relational structure of the local MySQL database.	40
4.1	Effect of image downscaling on AI recognition (nine test images). (a) Exact matches at Family and Genus levels for OpenAI and Gemini across five resolutions: both decline with lower resolution; OpenAI leads at Family for higher resolutions, but Genus accuracy drops to 0% below 50%. (b) Mean highest correctly identified rank (Order < Family < Genus < Species): accuracy peaks at full resolution and degrades with downscaling; OpenAI is stronger at high resolutions, Gemini more stable at lower ones.	50
7.1	Database structure and content in DBeaver SQL GUI tool: (Top) <code>TaxonomicData</code> table showing API responses for a sample image as well as an empty column for authority label. (Bottom left) <code>ImageData</code> table containing stored image BLOBs. (Bottom right) <code>RemoteMonitoringData</code> table containing environmental metadata linked to image IDs.	59
A.1	Schematic representation of data flow from the SBC to the internet. The CPU generates a digital bitstream, which is modulated by the cellular radio chip into an analog signal. This signal is transmitted via the antenna, received by the cell tower, and routed through the carrier's infrastructure to reach the internet.	64
A.1	MySQL terminal session showing schema structure of the project database.	67
A.2	Relational structure of the local MySQL database.	68

List of Tables

1.1	Overview of monitoring duration and methods in selected ALAN-insect studies.	3
4.1	Summary matrix of AI recognition performance across five image resolutions. The table reports the average taxonomic rank accuracy (mean correct rank position) and the average percentage of correct identifications at the Order, Family, and Genus levels for OpenAI and Gemini models.	48
4.2	Shapiro-Wilk Test Results: Tests indicated that the normality assumption was violated in more than half of the tested conditions ($p < 0.05$). Therefore, it was preferable to also consider the non-parametric Friedman tests as a more robust alternative.	51
5.1	Indicative bill of materials for the on-site unit.	54
A.1	Shapiro-Wilk test results	61
A.2	Chi-square test results	62
A.3	Pairwise Comparison Results (Gemini vs OpenAI)	62
A.4	Descriptive Statistics by Tool and Resolution	62

Listings

B.1	Allrun shell script to automate processes	69
B.2	Arduino code for AHT20 sensor data acquisition	71
B.3	Shell script for image data acquisition	72
B.4	Python script for geolocation	74
B.5	Python script for serial data logging	76
B.6	Shell script for data transmission	79
B.7	Python script for database logging	80
B.8	Python script for extraction and preprocessing	84
B.9	Python script for fetching images and performing insect recognition	87

ACRONYMS

ALAN	Artificial Light at Night
API	Application Programming Interface
CCU	Central Computer Unit
GPS	Global Positioning System
HAT	Hardware Attached on Top
HPS	High-Pressure Sodium
LED	Light-Emitting Diode
SBC	Single Board Computer
MCU	Microcontroller Unit
AI	Artificial Intelligence
CCT	Correlated Color Temperature
GPIO	General Purpose Input-Output
AC	Alternative Current
DC	Direct Current
RX	Receiver
TX	Transmitter
UART	Universal Asynchronous Receiver- Transmitter
LDO	Low Dropout
PCB	Printed Circuit Board
RC	Resistor-Capacitor
SCL	Serial Clock
SDA	Serial Data
GNSS	Global Navigation Satellite System
PPP	Point-to-Point Protocol
QMI	Qualcomm MSM Interface
FPS	Frames per Second
VPN	Virtual Private Network
VNC	Virtual Network Computing
CLI	Command Line Interface
DBMS	Database Management System

ROMAN SYMBOLS

<i>C</i>	Electric Capacity	F
<i>f</i>	frequency	Hz
<i>I</i>	electric current	A
<i>P</i>	power	W
<i>R</i>	electric resistance	Ω
<i>U</i>	voltage	V

1.1 CONTEXT

Artificial light at night has revolutionized human life by allowing extended hours of activity, especially in urban and suburban settings. Although outdoor lighting provides clear social benefits (e.g. enhanced visibility for road safety), mounting evidence shows that it significantly disrupts the natural behaviors and population dynamics of nocturnal insects [28]. Nocturnal insects perform numerous ecological functions, from pollination and seed dispersal to nutrient cycling and prey resource provisioning. Consequently, the introduction of ALAN disrupts these functions in multiple ways, from interference with nocturnal pollination to modifying predator-prey relationships. The negative impacts of ALAN reverberate throughout ecosystems, posing critical challenges to biodiversity and habitat health. Understanding and mitigating the effects of ALAN is a key concern for the conservation of these insects and the ecosystems that depend on them.

1.2 BACKGROUND AND SIGNIFICANCE

1.2.1 ARTIFICIAL LIGHT AT NIGHT

ALAN encompasses a wide range of artificial lighting sources that operate at night, including streetlights, residential and commercial lighting, vehicle headlights, and ornamental or monument lighting [9, 6, 22]. White LED lights often emit a relatively broad spectrum (Figure 1.1), including wavelengths in the blue range that can be especially disruptive to insects.

1.2.2 LIGHTING TECHNOLOGIES

The rapid uptake of LED fixtures in urban and suburban areas is driven by economic and environmental reasons (reduced electricity costs, lower emissions). However, from an ecological perspective, the spectral composition of these lamps contributes to flight-to-light behaviors in insects and disrupts circadian patterns [28]. Unlike older lighting technologies, LED luminaires offer greater spectral and intensity control, as they can emit monochromatic light at specific wavelengths within or adjacent to the visible spectrum. This tunability allows for more engi-

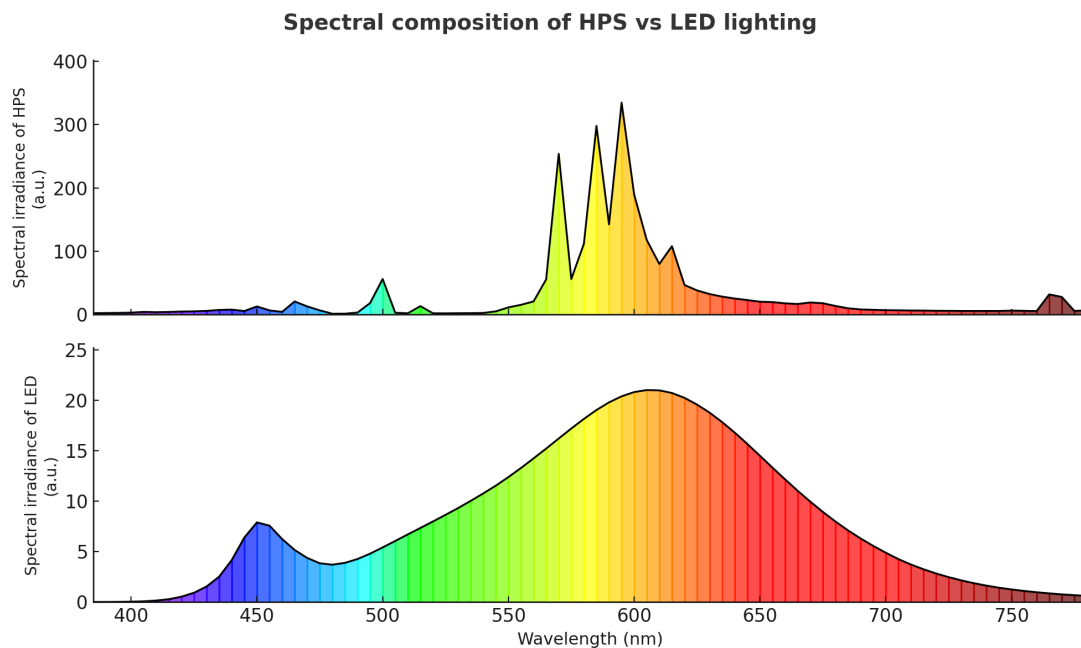


Figure 1.1: Relative spectral power distributions of HPS and LED lamps across the visible spectrum (380 nm–780 nm, 5 nm increments), reproduced from CIE 015:2018 (HPS: Table 11; LED: Tables 12.1/12.2) [14, 13]. The HPS source displays a narrow emission band peaking in the yellow–orange region, while the LED illuminant exhibits a broader profile characterized by a dominant blue spike and extended phosphor tail.

neering maneuverability for designing optimized lighting systems that can reduce ecological disturbances [28].

1.2.3 NOCTURNAL INSECTS

Nocturnal insects, including Lepidoptera (moths), Coleoptera (beetles), and Diptera (flies), are vital for pollination, decomposition, and the formation of the base of nocturnal food webs. Their activity supports plants that bloom at night and provides food for nocturnal predators such as Chiroptera (bats), and various amphibians. Hence, changes in insect populations under ALAN can trigger larger ecological imbalances, affecting plant reproduction, soil fertility, and predator–prey relationships [34].

Beyond ALAN, nocturnal insects face compounding threats, such as habitat loss, climate change, and pesticide use. ALAN often interacts in symbiosis with these other stressors, exacerbating insect declines [8]. In urbanized regions, shifting to energy-efficient but bright blue-rich LEDs can amplify these threats, particularly given the strong phototactic responses of many insect taxa [34].

1.2.4 THE NEED FOR LONG-TERM, SCALABLE INSECT MONITORING

Assessing the ecological impact of ALAN on nocturnal insects requires a sustained and geographically distributed data collection. However, existing studies, based on the preliminary literature reviewed in this thesis, rely largely on short-term monitoring campaigns due to logistical constraints and the labor-intensive nature of traditional sampling methods, such as light traps and manual retrieval, classification and count. As shown in Table 1.1, these studies typically span only a few nights to a few weeks and often offer limited spatial replication.

Table 1.1: Overview of monitoring duration and methods in selected ALAN-insect studies.

study	duration	technology	highlight
De Causmaecker et al. [8]	4 nights	camera with local storage	need for seasonal data integration
De Causmaecker et al. [7]	13 + 9 nights	camera, OpenCV, SD storage	high labor and use of killing traps
Lima et al. [17]	–	camera-based systems	cost barriers for wider deployment
Bolliger et al. [5]	32 nights	flight traps	influence of weather, site, and lighting
Bolliger et al. [4]	44 nights	flight traps	manual twice-daily maintenance; weather sensitivity noted

In addition, insect activity and abundance are strongly influenced by seasonal and meteorological variation. For example, Bolliger et al. [5] and Bolliger et al. [4] identified weather, precipitation, local site conditions, and lighting regime as the main drivers of the abundance of nocturnal insects. These observations reinforce the need for longer-term datasets to disentangle ecological signals from short-term variability.

In addition, long-term ecological trends, such as declines in insect populations or changes in community composition, typically emerge only after multiple seasons or years [11]. However, such time frames are rarely achieved because of the operational burden of current monitoring methods.

Together, these findings highlight a clear methodological gap: the absence of scalable, long-duration, and low-labor monitoring systems suitable for diverse field conditions. Addressing this gap is essential to improve the temporal and spatial resolution of ALAN impact studies. The present thesis addresses this

need by investigating the feasibility of an automated camera-based monitoring system capable of season-long operation with minimal human intervention.

1.3 RESEARCH QUESTION

The research question emerged after an initial literature exploration during which I became familiar with the field of study. Several potential research avenues were considered, including AI model training for insect recognition, camera system optimization, and remote monitoring system design. This exploratory phase helped identify both the ecological importance of ALAN impacts and the methodological gaps in existing insect monitoring approaches.

To ensure clarity and a well-defined scope, the research question was subsequently structured using the PICO framework, commonly employed in evidence-based research to distill the essential components of an investigation. The selection of each framework element was guided by the patterns and findings identified during the initial review of the relevant literature.

- **Population:** Positive-phototactic nocturnal insects.
- **Intervention:** Different setup of LED lighting
- **Comparison:** Darkness or alternative LED lighting conditions.
- **Outcome:** Insect abundance and community composition.

Based on these considerations, the final research question guiding this thesis is formulated as follows:

How do different LED lighting parameters affect the abundance and community composition of positive phototactic nocturnal insects in outdoor, suburban, rural, and semi-natural habitats under real-world field conditions?

The research question ensures that the focus remains on ecological studies conducted in field conditions, allowing for an assessment of ALAN's impact in environments that closely resemble natural ecosystems.

1.3.1 SUB-QUESTION

Although this umbrella question covers the entire causal pathway from ALAN to insect community responses, the present thesis addresses one crucial methodological step: the design, deployment, and validation of an automated image-based monitoring system capable of acquiring season-long datasets with minimal human labor. Consequently, the operational sub-question investigated here is:

Is it feasible to provide a low-cost, camera-based monitoring solution that can quantify the abundance and community composition of positive phototactic nocturnal insects across habitats with contrasting LED lighting parameters?

Implications: Successfully providing the solution described in the sub-question would yield a *low-cost, easily deployable, long-term, and accessible platform capable of amassing large data sets*, thereby laying the empirical and methodological foundation for wide-ranging ALAN assessments.

1.4 OBJECTIVES

1. Critically analyze and synthesize the existing scientific literature on ALAN and its effects on nocturnal insects.
2. Develop a low-cost system to collect reliable data on ALAN impact:
 - **Climate variables:** Temperature and relative humidity.
 - **Location:** Geographic coordinates of the monitoring unit.
 - **Lighting parameters:** CCT and illuminance.
 - **Image data:** Images of attracted insects.
 - **Taxonomic classification:** AI-driven taxonomic classifications.
3. Design and implement a monitoring system on site for the observation of nocturnal insects.
4. Describe potential future improvements and directions for continued research.

1.5 THESIS STRUCTURE

The remainder of this thesis is organized into five main chapters.

- **Chapter 2: Literature Review** presents the search methodology and the synthesis of existing research on the impact of artificial light at night on nocturnal insects, with a focus on relevant lighting technologies, measured parameters and identified knowledge gaps.
- **Chapter 3: Materials and Methods** details the design and implementation of the monitoring system. It covers the field layer (on-site unit), central computing layer, and cloud-based recognition layer, including hardware specifications, data acquisition workflows, and automation strategies.

- **Chapter 4: Case Study** evaluates the performance of general-purpose vision models for insect recognition, describing the dataset, metrics, and comparative results at different image resolutions.
- **Chapter 5: Discussion** interprets the case study results in the context of the project’s objectives.
- **Chapter 6: Future work** discusses system-level implications and outlines potential future developments.
- **Chapter 7: Conclusions** summarize the main contributions of the thesis, answer the research subquestion, and reflect on the progress toward the main research question.

A literature review serves as a structured process for evaluating and synthesizing existing research in a given field. In general, it comprises four interrelated phases:

1. **Search process:** Identify and collect relevant scholarly works using systematic search strategies to ensure comprehensive coverage of the topic.
2. **Relevance and eligibility:** Screen the collected material to exclude studies that do not meet the predefined criteria, retaining only those directly applicable to the research objectives.
3. **Data extraction and analysis:** Examine the content of selected studies, extract pertinent data, and analyze key findings or themes to build a cohesive picture of the existing knowledge.
4. **Synthesis and conclusions:** Synthesize the analyzed data into a coherent narrative, highlight gaps in the literature, and propose directions for future research.

2.1 SEARCH METHODOLOGY

Before conducting the literature review, it was essential to systematically identify and select relevant academic publications that address the research question. The following section outlines the methodological approach taken to establish eligibility criteria, select databases, construct search strategies, and refine the final data set.

2.1.1 ELIGIBILITY CRITERIA

To refine the search and ensure the inclusion of studies that are relevant and comparable, a set of strict eligibility criteria was established. These criteria were developed to systematically exclude studies that fall outside the scope of this review while ensuring consistency in the selection process.

Geographic scope:

- Studies conducted in **suburban, rural gardens, and parks** were included to maintain ecological relevance.
- Studies conducted in **fully urban environments** or **completely natural ecosystems** were excluded to ensure comparability with semi-controlled conditions.

Study design:

- **Included:** Systematic reviews, meta-analyses, analytical studies, and ecological field studies that investigate the effects of ALAN on nocturnal insect populations.
- **Excluded:** Laboratory-based studies, as they lack environmental variability and do not represent real-world field conditions.

Language restrictions:

- Only studies written in **English, French, and Dutch** were considered due to accessibility and comprehension constraints.

Outcome measures:

- The studies had to report changes in **abundance and/or community composition** of nocturnal insects under different LED lighting conditions.
- Studies focusing exclusively on physiological, molecular, or behavioral responses without population-level metrics were excluded.

2.1.2 DATABASE SELECTION

To ensure a comprehensive and interdisciplinary review, multiple academic databases were consulted. The selected databases were chosen based on their relevance to ecological and environmental sciences, as well as on their accessibility:

- **Web of Science:** Provides a broad coverage of environmental sciences, ecology, and biology.
- **Scopus:** A multidisciplinary database with ecological and environmental research.
- **GreenFILE:** Specializes in life sciences, ecology, and environmental sustainability.

2.1.3 SEARCH STRATEGY

A structured search strategy was developed to systematically retrieve relevant articles. The search terms were carefully designed to ensure a comprehensive coverage of studies focused on **population, intervention, geographic context, and outcome measures**. Boolean (advanced search options) was used to maximize retrievals. The final search string logic was formulated as follows:

```
TS=(((("LED" OR "light-emitting diode*")
AND ("artificial light at night" OR "ALAN"))
AND ("nocturnal insect*" AND (phototax* OR "positive phototaxis"
OR "positively phototactic"))
AND ("field stud*" OR "field experiment*" OR "outdoor" OR "garden*"
OR "semi-natural" OR "rural" OR "suburban")
AND ("abundance" OR "richness" OR "diversity" OR "communit* composition"
OR "species assemblage" OR "population dynamic*"))
AND LA=(English OR French OR Dutch)
```

2.1.4 DATABASE-SPECIFIC ADAPTATIONS

Since each database has its own specific syntax and search functionalities, the general search string was translated accordingly using an AI tool (see "**Disclosure on the use of AI**"). The searches were carried out on **10 December 2024**, giving the following results:

Web of Science:

```
TS=(((LED OR "light-emitting diod*")
AND (ALAN OR "artificial light at night"))
AND ("nocturnal insect*")
AND (abundance))
```

Results: 8 papers retrieved. The following articles were recovered: Bolliger et al. [4], Willmott et al. [38], Bolliger et al. [5, 3], Méndez et al. [22], Seymoure et al. [35], Deichmann et al. [9], and Boyes et al. [6].

GreenFILE:

```
TX ( (LED OR "light-emitting diod*") )
AND TX ( (ALAN OR "artificial light at night") )
AND TX (abundance)
AND TX (insect*)
```

Results: 5 papers retrieved. The following articles were recovered: Russo et al. [34], Lockett et al. [18], McNaughton et al. [20], Perrin et al. [29], and Luo et al. [19].

Scopus:

```
TITLE-ABS-KEY(((LED OR "light-emitting diod*")
AND (ALAN OR "artificial light at night")
AND ("nocturnal insect*")
AND (abundance)))
```

Results: 5 papers retrieved. The following articles were recovered: Bolliger et al. [4], Willmott et al. [38], Bolliger et al. [5, 3], and Méndez et al. [22]. [4]

2.1.5 PAPER SELECTION AND FINAL DATASET

After retrieving papers from all databases, duplicate studies were identified and removed (automatic duplicate detection within the zotero application). Of the **18 studies initially recovered, 5 were duplicates**, leaving a total of **13 unique studies**:

1. Russo et al. [34]
2. Luo et al. [19]
3. Méndez et al. [22]
4. Seymoure et al. [35]
5. Bolliger et al. [3]
6. Bolliger et al. [5]
7. Bolliger et al. [4]
8. Willmott et al. [38]
9. Deichmann et al. [9]
10. Perrin et al. [29]
11. McNaughton et al. [20]
12. Boyes et al. [6]
13. Lockett et al. [18]

However, one of these studies required special access permissions, which could not be obtained within the time frame of this review [38]. Two other studies were, after reading, considered irrelevant ([29], [19]). As a result, the final data set consists of **10 studies**, which provide the most up-to-date information on the effects of ALAN on nocturnal insect abundance and community composition.

These selected articles form the foundation of the following literature review, allowing for a structured assessment of the existing scientific knowledge before conducting the primary experimental investigation.

2.2 SYNTHESIS AND CONCLUSIONS

2.2.1 REVIEW SYNTHESIS METHODOLOGY

To structure the literature review in a systematic way, I created a comparative matrix that captured key parameters in all studies. The matrix included the following fields:

- Citation and year
- Research objective
- Population studied (geographic location, and season)
- Intervention/exposure details (light source, spectrum, intensity, duration)
- Comparison
- Results
- Limitations or sources of bias

This tabular synthesis allowed for a clear comparison of apples-to-apples between studies and revealed patterns or inconsistencies in methodology. It also made it easier to identify gaps, which are later highlighted in the synthesis. Based on this structured overview, the following sections were organized thematically: abundance, community composition, and knowledge gaps.

2.2.2 RESEARCH SCOPE

How do LED lighting parameters affect the abundance of nocturnal insects and the composition of the community in outdoor habitats? Ten field studies (2019-2024) in Europe, South America, China and Australia form the evidence base.

2.2.3 KEY LED PARAMETERS EXAMINED

A cross-comparison of the studies reveals five main lighting parameters that influence insect responses:

- **Illuminance and dimming:** Traffic-adaptive dimming that reduced the mean illuminance by $\approx 35\%$ decreased insect captures compared to fully lit controls [5], confirming that illuminance is an important driver of attraction.
- **Spectral composition (CCT):** Across the available range (1750 K–6500 K) insect attraction increased with the correlated color temperature, at comparable luminous flux ([3], [5], [4], [9], [18]).
- **Spectrum filtering:** Commercial amber and yellow filters (CW8, CW9) that removed most of the blue light reduced the abundance by up to 40% compared to unfiltered 3000 K lamps [9].
- **Luminaire geometry and light distribution:** Diffusers that expand the spatial light distribution attracted 16% more insects than standard cut-off luminaires at equal luminous flux [3]. This suggests that the distribution and beam govern the size of the visual catchment and, therefore, the scale of ecological impact.
- **Legacy lamp comparison:** Where LEDs replaced high pressure sodium (HPS) lamps of comparable road class illuminance, the number of insects increased markedly under the LED regime [6]. This shift is attributed to the higher blue content and the larger spectral bandwidth of the white LEDs (Figure 1.1). Importantly, the HPS–LED comparison underscores that any transition to energy-efficient lighting must consider ecological consequences on biodiversity.

2.2.4 EFFECTS ON INSECT ABUNDANCE

In every study, the abundance of insects increased with the amount of light or the proportion of blue wavelengths. In a study performed in Switzerland, capture rates rose with the percentage of time that lights operated at full power, even after accounting for warm, dry weather [5]. A sister study found no abundance difference between 2700 K and 6500 K LEDs at the same luminous flux, but temperature again dominated the model [4]. Where CCTs spanned a wider range (1750 K)–4000 K, 1750 K lights attracted fewer insects than 3000 K or 4000 K lights. However, the overall abundance did not differ significantly between 3000 K and 4000 K, except for Lepidoptera, which were more numerous at 4000 K. The 4000 K diffuser combination still yielded the highest counts [3].

In the English lowlands, the negative impacts were more pronounced under white LED street lights than under conventional yellow sodium lamps, demonstrating

that the ongoing transition from narrow-spectrum HPS to broad-spectrum LEDs (Figure 1.1) can exacerbate the declines in nocturnal insect populations [6]. In Melbourne, the abundance of airborne invertebrates doubled under street lights relative to adjacent dark gaps, with the strongest effect under higher CCT fittings [18]. Filtering blue wavelengths in a tropical forest reduced captures by one third compared to unfiltered white LEDs [9]. Collectively, these results show that intensity and spectrum are co-determinants, with the temperature of attraction.

2.2.5 EFFECTS ON COMMUNITY COMPOSITION

Community-level responses were evident in four studies. Lepidopteran captures were particularly sensitive to spectral and geometric treatments: catches in 3000 K and 4000 K with diffusers exceeded those at 1750 K [3]. Warm white LEDs (2700 K) shifted the family-level composition toward larger moths [4]. In Melbourne, airborne assemblages under lights diverged significantly from those in darker intervals, while ground-dwelling communities showed indirect shifts, suggesting cascading trophic effects [18]. Deichmann et al. [9] recorded greater variety of insect types under white light compared to amber filtered light, although this difference in variety was small compared to the much larger difference in the number of insects observed. In general, ALAN not only inflates local insect numbers, but also reshapes community structure, with blue-rich spectra exerting the strongest selection pressure.

2.2.6 EVIDENCE AND KNOWLEDGE GAPS

- **Reliance on lethal sampling:** All studies used light trap capture methods that kill insects and require manual identification ([5], [4], [3]). Sensor-based, non-lethal techniques remain underexplored.
- **Labour-intensive workflows:** Bulk samples require expert sorting and counting, limiting temporal and spatial coverage [22].
- **High equipment costs:** Integrated monitoring rigs are often prohibitively expensive, restricting deployment (<https://www.ceh.ac.uk/solutions/equipment/automated-monitoring-insects-trap>)
- **Sparse spatio-temporal resolution.** Most experiments span fewer than eight weeks at a handful of locations, yielding snapshots rather than long-term trends [22].

2.2.7 CONCLUSION

Two principles emerge consistently from temperate and tropical landscapes:

1. nocturnal insect attraction rises with both light intensity and the proportion of blue wavelengths, as well as local weather temperature and,

2. design choices that mitigate but do not eliminate biological impacts. Luminance geometry can compound the attraction by enlarging the visual catchment.

However, current evidence is hampered by short study durations, lethal sampling, and limited geographic coverage. To close these gaps, low-cost and portable science-ready monitoring modules are essential. By empowering volunteers to deploy hundreds or dozens of devices, researchers can capture huge insect activity data across seasons, habitats, and continents. The result will be a richer empirical basis for designing lighting technologies that safeguard both human needs and ecological integrity.

3.1 SYSTEM ARCHITECTURE AND OVERVIEW

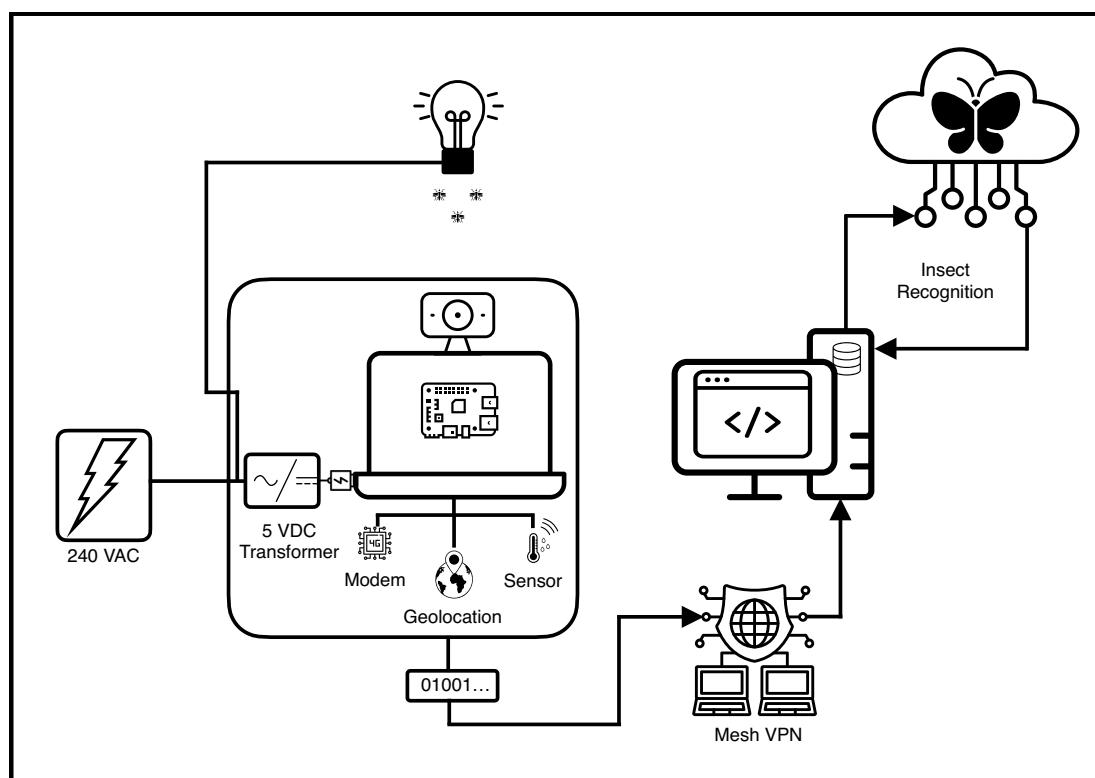


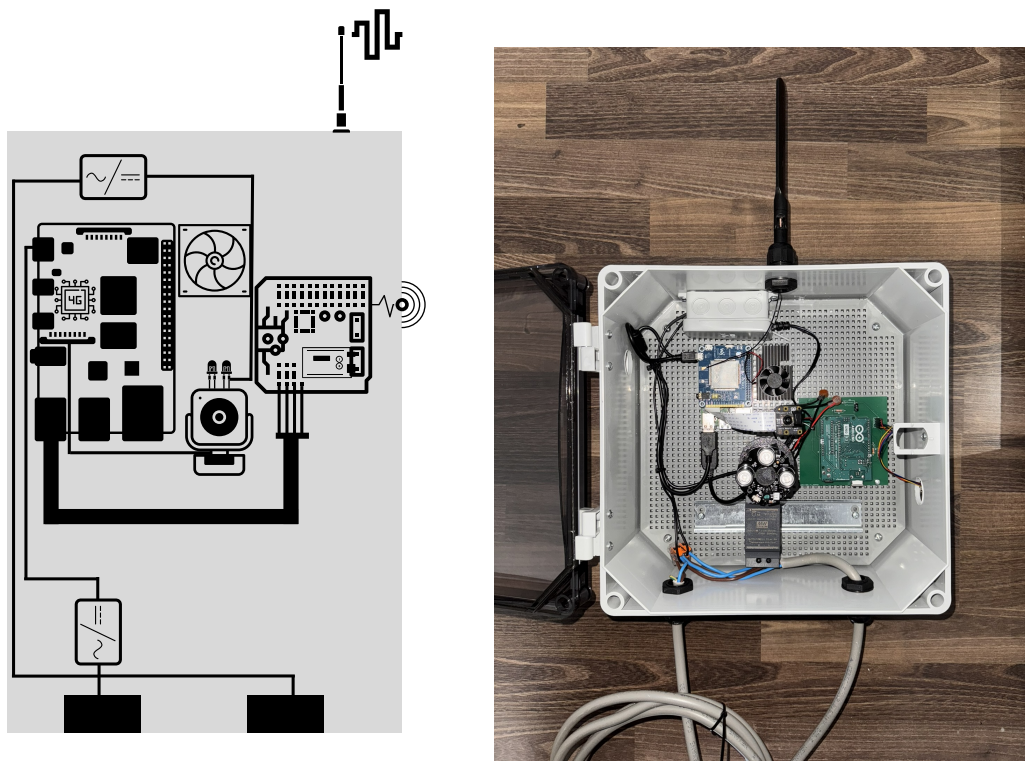
Figure 3.1: Schematic overview of the implemented system showing major components and data flow.

This thesis aims to develop a low-cost monitoring system capable of capturing images of nocturnal insects, recording local environmental parameters, and transmitting these data for remote analysis. The system is designed to be portable and operate in multiple locations with minimal human intervention, supporting citizen science and distributed ecological studies.

To achieve this, the system adopts a three-layer architecture. Each layer is responsible for a specific set of tasks, ranging from field data acquisition to centralized storage and cloud-based insect recognition, and classification. Figure 3.1 presents an overview of the system topology and the main components involved.

1. **On-site Monitoring Unit (field layer)**: a node deployed in the habitat itself. Capture images, record local weather, and upload data to the central computer unit. (sec 3.2)
2. **Central Computer Unit (central layer)**: a workstation that manages all communication with the remote unit, receives and stores image, weather and geolocation data, makes API requests. (3.3)
3. **Recognition API (cloud layer)**: web services that receive still images, execute insect recognition and classification, and return a full taxonomic label.

3.2 FIELD LAYER: ON-SITE MONITORING UNIT



(a) System schematic of the on-site unit. (b) Real-life setup of the on-site unit.

Figure 3.2: Schematic and actual setup of the on-site monitoring unit. The left panel shows the functional layout, while the right panel illustrates the physical deployment.

3.2.1 FUNCTIONAL OBJECTIVES AND REQUIREMENTS

The on-site monitoring unit is the subsystem in charge of monitoring insect activities, recording local weather data, setting up the light configuration, and transmitting data. It should withstand moderate outdoor environmental conditions. In order to meet these objectives, the physical setup was designed as follows.

3.2.2 PHYSICAL SETUP

The remote monitoring system is enclosed within a waterproof IP65 box to ensure protection against rain, dust, and harsh environmental conditions.

Internally, the system integrates a SBC connected to a camera module positioned to capture images through a transparent cover. [31, 32]

For remote data communication, 4G connectivity was enabled using the SIM7600G-H 4G HAT module [37].

MCU is used in conjunction with the SBC to manage sensor-specific tasks, such as reading and writing data from the temperature and humidity sensor. The microcontroller used for sensor polling is an Arduino Uno R3 [2]. Although the SBC features GPIO pins, they were not used for this task due to the lack of real-time guarantees in standard operating systems. As explained by Igoe [12], SBC operating systems introduce scheduling overhead and do not ensure deterministic timing for I/O operations. In contrast, a standalone microcontroller runs a single program without interruptions, allowing a precise and timely acquisition of physical input. The use of a dedicated MCU, therefore, ensures reliable sensor polling. Furthermore, the availability of an Arduino board made this approach practical and resource-efficient.

The system is powered by an external line 240 VAC, converted internally to 5 VDC using a Mean Well HDR-30-5 DIN rail power supply [21]. This unit delivers a regulated output 5 VDC with a typical voltage variation of $\pm 2\%$ and a maximum current output of 3 A, suitable for the Raspberry Pi 4B. According to its specifications, the Raspberry Pi 4B requires 5 VDC input via the USB-C connector with a minimum current of 3 A [31].

To support image acquisition at night, the system includes a 3 W infrared LED aligned with the camera's field of view. This addition enables the collection of visual data under naturally unlit conditions, allowing meaningful comparisons between scenarios with and without artificial lighting.

In short, we have an IP65-rated enclosure that houses the main components of the on-site monitoring unit:

- **Raspberry Pi 4B** as the SBC [31], connected to:
 - **Raspberry Pi Camera Module V3 NoIR Wide** [32], mounted inside and aligned with a transparent window.
 - **WaveShare SIM7600G-H 4G HAT** [37] for mobile data connectivity,
 - A GPS antenna bundled with the 4G HAT, used for geolocation.
- **Arduino Uno R3** as MCU [2], connected to:

- **Adafruit AHT20** digital temperature and humidity sensor [1].
- **Mean Well HDR-30-5** power supply module [21].
- **Infrared LED** positioned near the camera to enable nocturnal image capture. This addition ensures visual monitoring under unlit conditions and is used to provide for comparisons between lit and unlit.

3.2.3 ELECTRONIC DESIGN DETAILS

3.2.3.1 Power delivery and conversion

The system is powered directly from a main supply 240 VAC, which is converted internally to 5 VDC using a transformer module [21]. This module is rated for a maximum output of 3 A at 5 V, corresponding to a maximum power capacity of 15 W, which is sufficient to accommodate the simultaneous operation of the Raspberry Pi 4B, the 4G HAT, the Arduino Uno R3, and connected sensors and peripherals.

Retaining 240 VAC as the primary power source was a deliberate design decision. The external LED light bulb used to attract nocturnal insects is a standard AC-based unit mounted on 240 V sockets. Using a high-voltage AC supply simplifies wiring, ensures consistent illumination, and avoids the additional complexity and limited autonomy associated with battery-powered systems.

3.2.3.2 Arduino and sensor circuit

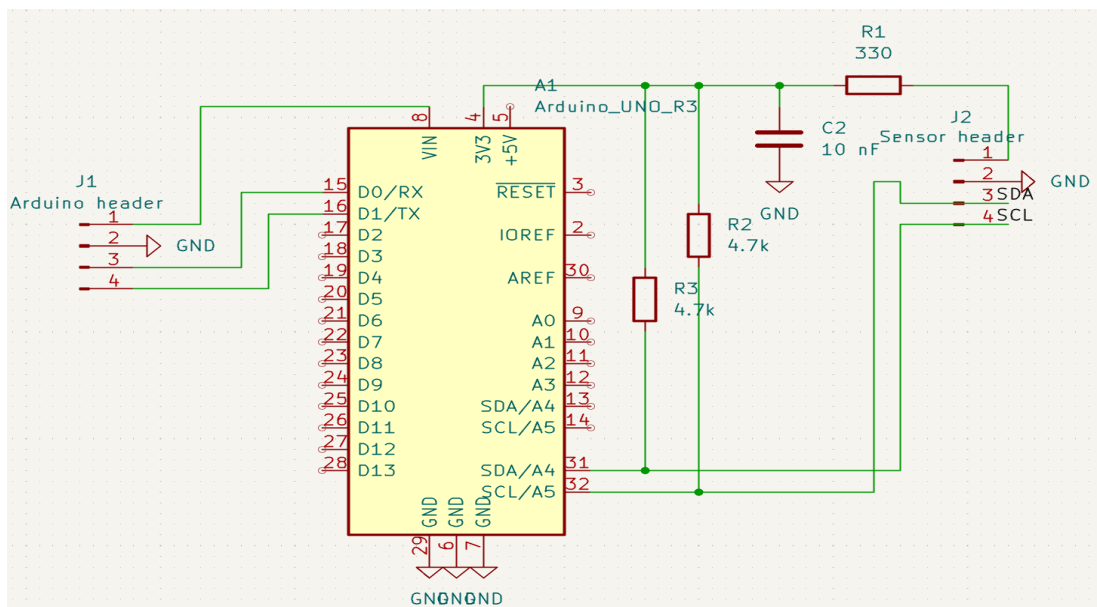


Figure 3.3: Schematic of the PCB powering and connecting the Arduino Uno to the temperature and humidity sensor.

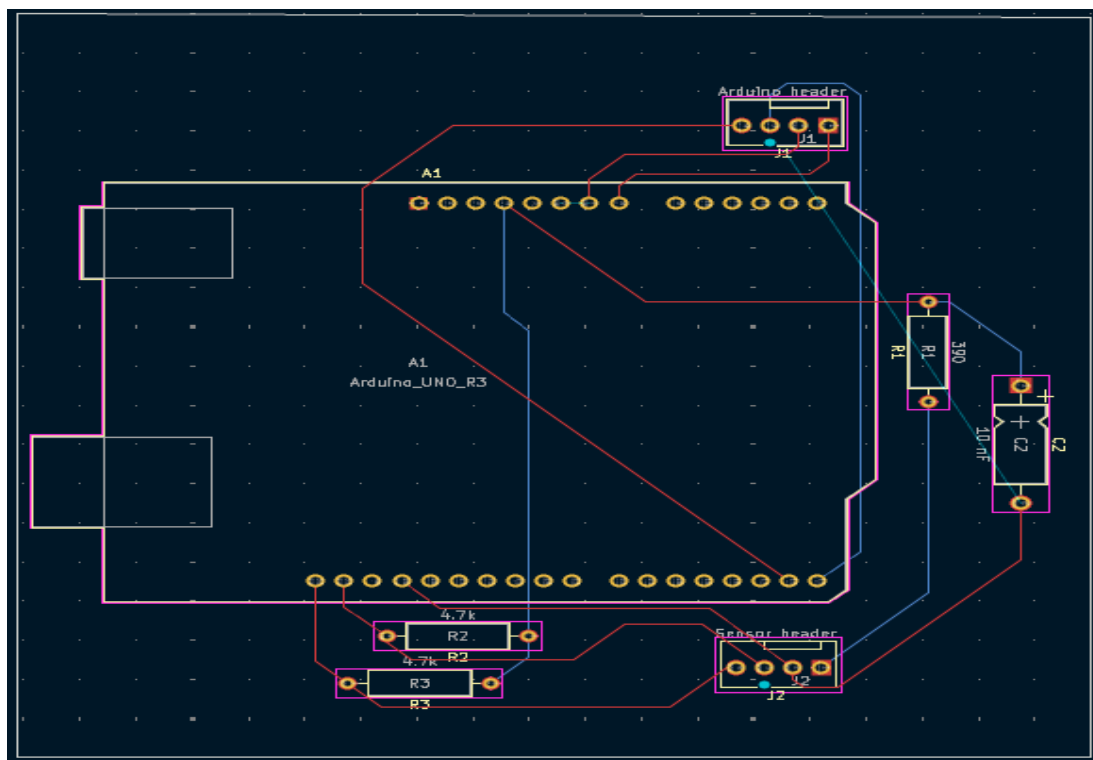


Figure 3.4: Designed PCB layout.

The temperature and humidity sensor is connected to the MCU via an I²C communication bus. Figure 3.3 shows the schematic of the custom PCB used to interface both components.

As shown in Figure 3.3, the MCU is powered through a USB-to-TTL serial cable connected to the SBC. A 4-pin male header receives the cable's VDD, GND, RX, and TX lines. The power lines are routed through the PCB to the MCU's Vin and GND pins, while the UART lines are directed to the MCU's TX and RX pins, enabling both power delivery and serial communication.

The sensor is powered by the 3.3 V pin of the MCU, as shown in Figure 3.3. This choice was made because the output 3.3 V provides a more stable voltage than the 5 V line, while still falling well within the sensor operating range of 2.2 V to 5.5 V [1].

According to the MCU datasheet [2], the 3.3 V output is generated via a secondary LDO regulator powered by the main 5 V rail, providing additional isolation from ripples induced by loads and noise on the 5 V power line (see Figure 3.5). The sensor has a typical power consumption of 3.2 mW [1], which corresponds to less than 1 mA at 3.3 V, and is thus well within the 50 mA current capacity of the 3.3 V pin. This makes the 3.3 V rail a suitable and stable power source for this application.

The recommended sensor application circuit, as provided in the datasheet [1], is

3.3 Power Tree

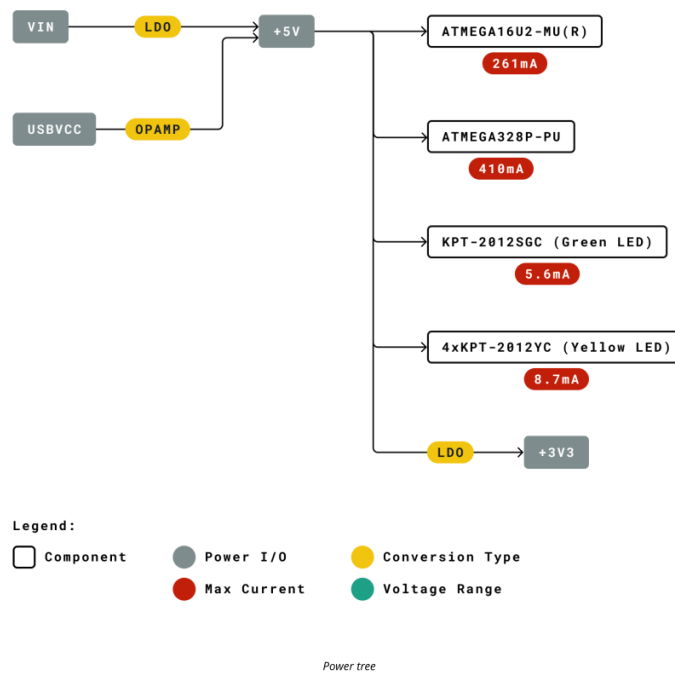


Figure 3.5: Power distribution tree of the Arduino Uno R3. The 3.3 V line is generated by a dedicated LDO regulator powered by the main 5 V rail. This configuration reduces voltage ripple and noise, improving stability for sensors [2].

shown in Figure 3.6. This schematic outlines the essential components required to ensure proper sensor operation, including an RC filter on the power line and pull-up resistors on the I²C communication lines.

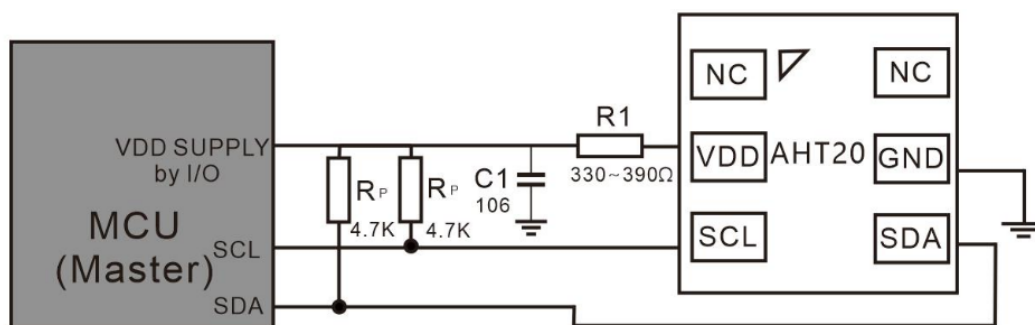


Figure 3.6: Typical application circuit for the AHT20 sensor, including pull-up resistors on the I²C lines (4.7 k Ω), a decoupling capacitor (10 nF) to ground, and a series resistor (390 Ω) between the VDD line and the sensor. These elements collectively form a basic RC low-pass filter to mitigate power line noise and enhance signal stability. Adapted from the AHT20 datasheet [1].

Although not explicitly represented in the application circuit (Figure 3.6), the

datasheet states that a decoupling capacitor of 10 nF must be added between VDD and GND to play a filtering role [1].

Following this recommendation, a resistor of 390 Ω and a capacitor of 10 nF were selected. The resulting cutoff frequency of the RC filter can be calculated using the standard formula:

$$f_c = \frac{1}{2\pi RC}$$

Substituting the chosen values:

$$f_c = \frac{1}{2\pi \cdot 390 \Omega \cdot 10 \text{ nF}} \approx 40.7 \text{ kHz}$$

Frequencies higher than this threshold are filtered, effectively bypassing high-frequency noise to the ground while preserving the steady DC supply to the sensor.

In addition to power line filtering, the typical application circuit includes pull-up resistors on the I²C communication lines (SCL and SDA). These resistors are necessary for proper bus operation, as the I²C protocol relies on open drain outputs that require external pull-ups to define high-level logic. Without them, the lines would float when not actively driven low, leading to undefined or unstable communication.

As shown in Figure 3.6, two 4.7 k Ω pull-up resistors are connected between the 3.3 V rail and the SCL and SDA lines.

3.2.4 CONNECTIVITY

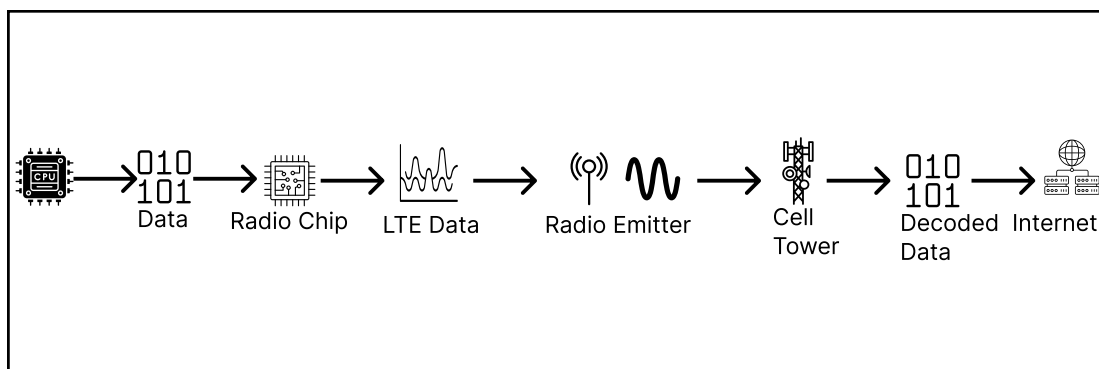


Figure 3.7: Schematic representation of data flow from the SBC to the internet. The CPU generates a digital bitstream, which is modulated by the cellular radio chip into an analog signal. This signal is transmitted via the antenna, received by the cell tower, and routed through the carrier’s infrastructure to reach the internet.

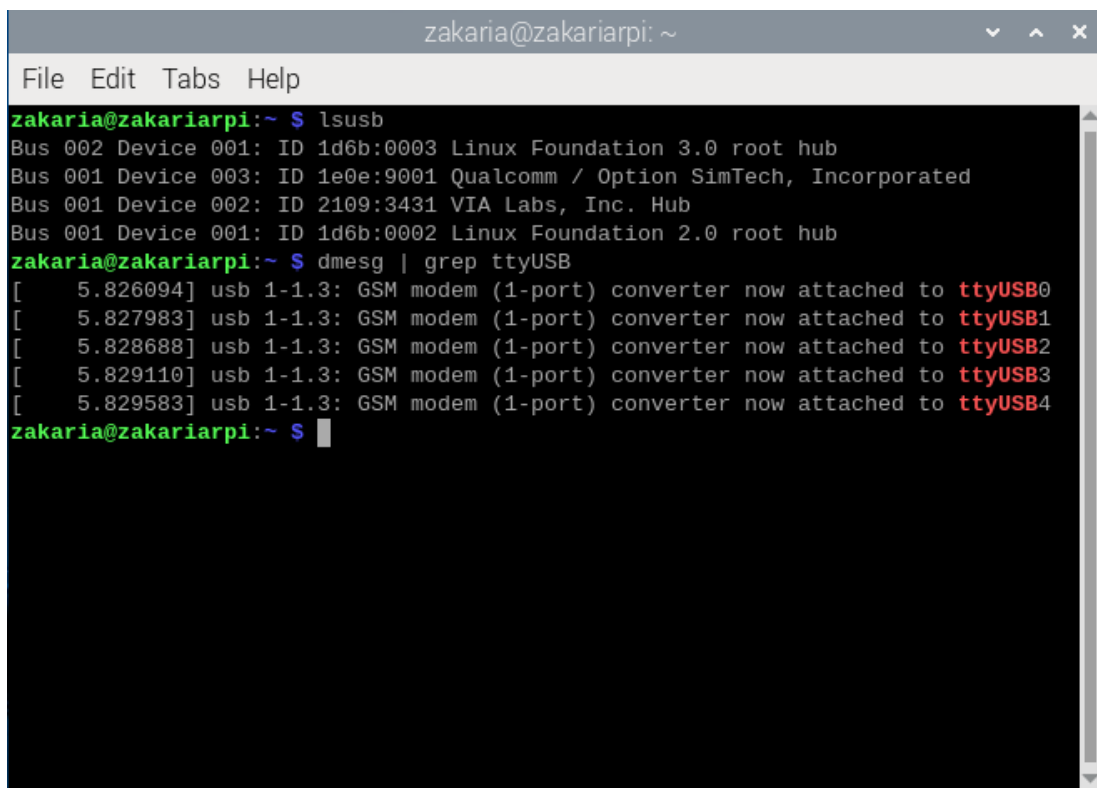
To enable wireless data transmission from remote locations, the monitoring system integrates an LTE modem [37] (see Appendix A for more information). This module provides cellular connectivity through LTE networks (see Figure 3.7) and

is physically mounted on the GPIO header of the SBC. In addition to the GPIO interface, a USB cable links the modem to the SBC, allowing data communication and exposing multiple logical interfaces over a single physical connection.

When connected via USB, the modem is recognized by the operating system as a USB composite device. This means that although there is only one USB connection, the device presents multiple interfaces to the system, each dedicated to a specific function. Running the command:

```
dmesg | grep ttyUSB
```

reveals the serial ports detected by the SBC (Figure 3.8).



```
zakaria@zakariarpi: ~
File Edit Tabs Help
zakaria@zakariarpi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 1e0e:9001 Qualcomm / Option SimTech, Incorporated
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
zakaria@zakariarpi:~ $ dmesg | grep ttyUSB
[ 5.826094] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB0
[ 5.827983] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB1
[ 5.828688] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB2
[ 5.829110] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB3
[ 5.829583] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB4
zakaria@zakariarpi:~ $
```

Figure 3.8: CLI command output showing the virtual serial interfaces forwarded by the LTE modem over USB to the SBC.

These virtual serial interfaces serve different roles in modem operation. Although the exact mapping may vary with firmware or configuration, typical roles include

- **Data channel:** Facilitates the mobile broadband connection for internet access.
- **AT command interface:** Used to control the modem via standard AT commands (e.g., SIM status, signal strength, network registration).
- **GPS interface:** Streams GNSS data in NMEA format.

- **Diagnostic interface:** Used for debugging or internal logging purposes.

3.2.5 SENSING, DATA ACQUISITION AND TRANSMISSION

3.2.5.1 Sensing

To monitor both environmental conditions and nocturnal insect activity at the study sites, the system acquires the following types of data:

- Temperature and humidity
- Geolocation
- Images

Temperature and humidity monitoring

The environmental temperature and humidity are measured using the AHT20 sensor [1], which communicates with the MCU via the I²C protocol. The MCU operates using embedded code without a full operating system, providing stable and responsive real-time data acquisition [12]. It is connected to the SBC through a USB-to-TTL serial cable, which handles both power delivery and data communication.

Geolocation (GPS)

Geolocation data are provided by the onboard 4G modem module [37], which is physically mounted on the SBC. The GPS antenna is connected directly to the modem antenna port for the reception of satellite signals. The modem communicates with the SBC through a USB connection. This approach enables concurrent 4G network data transmission and GPS positioning.

Imaging

Visual data is captured using the Raspberry Pi V3 NoIR Camera Module [32], which connects to the SBC through the dedicated Camera Serial Interface (CSI) port. The camera can operate in low light conditions and is used to observe insect activity. The images are saved locally and can be transferred or processed as part of the data pipeline.

Data Access and Acquisition

Sensor data is collected by the SBC through the corresponding USB serial ports. Environmental measurements from the MCU and GPS data from the 4G modem are accessed using standard Linux serial interfaces. To enable real-time data monitoring and logging, a terminal multiplexer is used, allowing multiple serial connections to be observed or logged in parallel from the command line.

3.2.5.2 Data acquisition

Environmental and observational data are acquired through coordinated communication between the sensors, the MCU, and the SBC.

Environmental data acquisition: The MCU reads sensor temperature and humidity measurements from the sensor via the I²C protocol. The sensor readings are obtained every second and are formatted into human-readable strings, e.g., "Temperature: 23.5 °C" and "Humidity: 60.2% rH". These formatted strings are transmitted over a USB-Serial connection to the SBC.

On the SBC, a Python script (Listing B.5) continuously listens to the USB serial port using the `pyserial` library [16]. Upon receiving a new line of data, the script timestamps the entry and appends it to a structured log file (`sensor_log.txt`) for later processing. The Arduino sketch responsible for acquiring and transmitting environmental measurements is shown in Listing B.2.

A challenge encountered during implementation was the non-deterministic assignment of USB port numbers by the operating system. Initially, the Python script relied on a hardcoded device path (e.g. `/dev/ttyUSB0`) to identify the Arduino interface. However, this approach was unreliable, as port numbering varies between system boots and when multiple USB devices are connected.

To address this, a more robust identification method was implemented using Linux's persistent device labels. Specifically, the following command was used to list stable symbolic links corresponding to connected USB serial devices:

```
ls -l /dev/serial/by-id/
```

This link includes a unique identifier derived from the device's serial number and manufacturer. By referencing this fixed path in the script instead of a volatile port number, the system achieves consistent and automated detection of the correct serial interface across reboots and deployments.

Geolocation data acquisition: The 4G HAT provides GPS data through its USB connection to the SBC. The SBC initiates and controls GPS sessions using AT commands sent through a serial interface. The GPS acquisition process (shown in Listing B.4) proceeds as follows:

1. The SBC powers the GPS functionality of the 4G HAT.
2. It sends an `AT+CGPS=1,1` command to activate GPS acquisition.
3. Once active, the SBC queries the GPS coordinates using the `AT+CGPSINFO` command.

Observational data acquisition: The camera module captures observational data in the form of short videos. Instead of capturing isolated images, the system records 1-minute video sequences at a frame rate of 1 fps and a resolution of 1640×1232 pixels using the libcamera-vid utility. The script used to perform video acquisition and post-processing is provided in Listing B.3.

The captured video is first recorded in H.264 format and subsequently converted to an MP4 container using ffmpeg to facilitate compatibility and playback. The video files are stored inside dedicated timestamped folders created during each acquisition session. Each video recording is time-stamped through its filename and folder structure, allowing easy synchronization with environmental and GPS data.

3.2.5.3 Data transmission

To enable secure and reliable communication between the SBC and the CCU, a VPN was established using Tailscale. Tailscale creates an encrypted peer-to-peer mesh network based on the WireGuard protocol, allowing devices to communicate as if they were on the same local network, even when physically distant. For proper operation, the Tailscale client must be active on both devices. It provides a web-based administration interface (the "workbench") that allows fine-tuning access policies and device visibility.

To further streamline and secure remote access, Tailscale SSH was enabled on both the SBC and the CCU. Tailscale SSH replaces traditional SSH key management by leveraging Tailscale's internal authentication, encryption, and ACLs. In this setup, SSH connections are automatically authenticated based on the devices' Tailscale identities, eliminating the need for manual password entry or traditional SSH key distribution. SSH traffic is routed internally through the Tailscale daemon (process), ensuring that connections are authenticated and encrypted without additional user interaction.

For data transmission, a custom Bash script (Listing B.6.) was developed to automate the packaging and secure transfer of recorded data. The script compresses the specified data folder into a .tar.gz archive and transfers it via rsync (native and lightweight Linux tool) over the Tailscale SSH tunnel to the CCU. An overview of the script's operation is as follows:

1. Verifies the existence of the folder corresponding to the measurement timestamp.
2. Creates a compressed archive of the folder.
3. It uses rsync to transfer the archive to a predefined directory on the CCU, leveraging the secure and authenticated Tailscale SSH connection.

4. The local archive is deleted after the transfer to save space.

3.2.5.4 Automation

Finally, all the above-mentioned processes are automated using a routine script (Listing B.1). The automation is designed to streamline the complete workflow, from data acquisition to secure transmission, minimizing manual intervention and potential errors.

The shell script follows a sequential structure:

- First, it activates the virtual Python environment required to run the sensor logging script.
- A shared timestamp is then generated, serving both as a folder name to organize recorded data and as a reference for the measurement session.
- Data logging is initiated in the background, while video capture is performed simultaneously, ensuring synchronized acquisition of environmental and visual information.
- After video recording is completed, the background sensor logging process is stopped.
- The resulting data, including sensor logs and recorded video, is archived and transmitted to the CCU via a secure Tailscale SSH connection using the previously described transfer script.
- After successful transmission, the local data and the archive are deleted to conserve storage space.

3.2.6 SETUP MANUAL

Step 1: Preparation of the SD card or USB drive

Ensure that the following materials are available:

- An SD card or USB drive (sufficient storage capacity for the operating system, estimated at xx GB)
- A computer with the Raspberry Pi Imager software installed

Step 2: Flashing of the operating system on the storage device

The Raspberry Pi Imager is used to flash the operating system onto an SD card or USB drive. In the application interface, proceed with the following selections (Figure 3.9):

- **Device:** Raspberry Pi 4 (Figure 3.9a)
- **Operating System:** Raspberry Pi OS (64-bit) (Figure 3.9b)
- **Storage:** Select the SD card or USB drive connected to your computer (Figure 3.9c)

Before starting the flashing process, optional configurations can be enabled by accessing the advanced settings menu (Figure 3.11). This includes activating SSH access (Figure 3.11b) and configuring Wi-Fi credentials (Figure 3.11a). The configuration interface can be accessed as shown in Figure 3.10a. Once the flashing is complete, a confirmation dialog is displayed (Figure 3.10b).

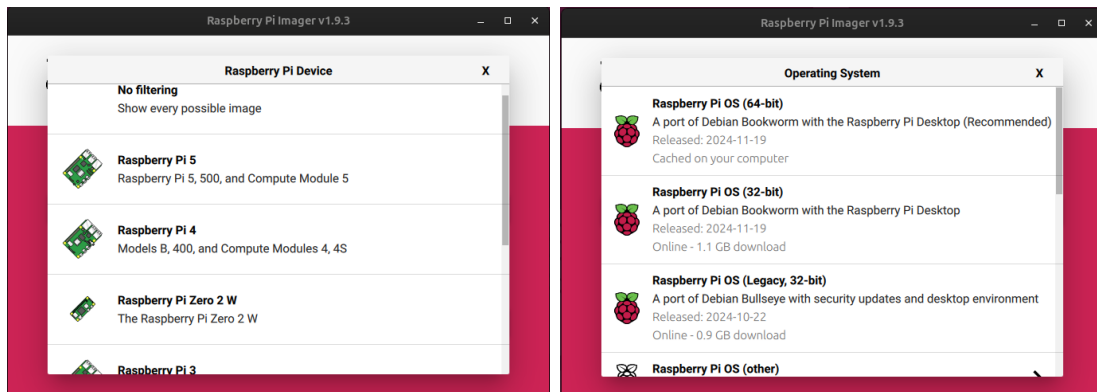
After flashing, insert the SD card or USB drive into the Raspberry Pi. Connect the device to a power supply. Optionally, an HDMI-connected monitor can be used to verify that the device is operating correctly.

Step 3: SSH access and enable the VNC interface

To facilitate user-friendly interaction with the SBC, the VNC interface is enabled. This allows graphical remote access via a VNC viewer application on a remote computer.

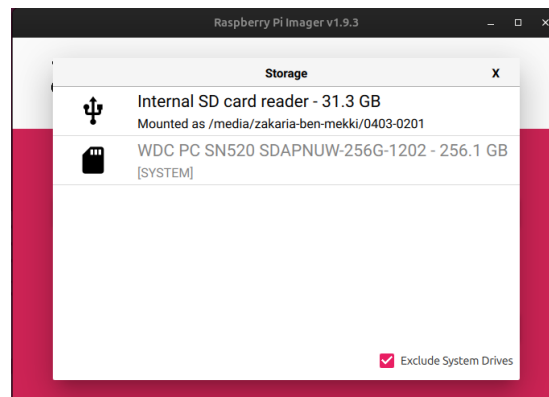
Establishing an SSH connection To connect to the Raspberry Pi remotely, ensure that:

- The Raspberry Pi and your computer are connected to the same local network.

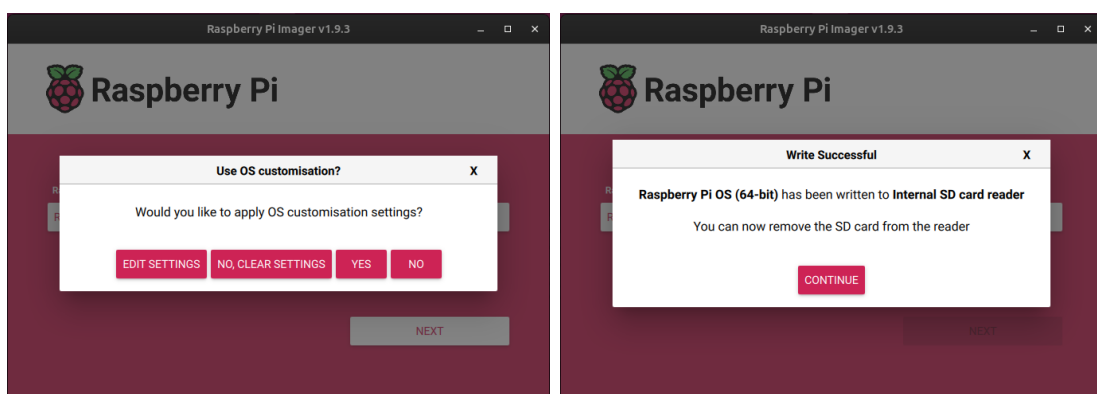


(a) Device selection

(b) OS selection



(c) Storage selection

Figure 3.9: Flashing the operating system using Raspberry Pi Imager.

(a) Editing settings

(b) Confirmation: flashing complete

Figure 3.10: Optional pre-configuration and confirmation steps (part 1).

(a) Wi-Fi configuration

(b) SSH configuration

Figure 3.11: Optional pre-configuration and confirmation steps.

- An SSH client [26] is installed on your computer (commonly included in Linux / MacOS systems or through tools like PuTTY [30] on Windows).

Obtain the IP address of the Raspberry Pi using the following steps:

1. Identify the subnet using the command:

```
ip a
```

This reveals your current machine's IP address and subnet mask, e.g., 192.168.1.105/24.

2. Scan the local network using `nmap` [10]:

```
nmap -sn --system-dns 192.168.1.1/24
```

This command searches for active devices on the subnet (in this example, managed by the router at 192.168.1.1) and returns their IP addresses and host names.

Once the Raspberry Pi's IP address is identified, establish an SSH connection with:

```
ssh <USER_NAME>@<IP_ADDRESS>
```

Optional: Update the system After successfully connecting to the Raspberry Pi, it is recommended to update the system packages by running:

```
sudo apt update
sudo apt upgrade
```

This step ensures that all installed packages and dependencies are brought to their latest stable versions. As mentioned in Section 3.2.6, the operating system image used during installation is a specific snapshot of the Raspberry Pi OS. By the time the image is flashed, some packages may already be outdated or have known issues. Updating the system helps correct these issues, improve compatibility with hardware and libraries, and ensure that subsequent software installations proceed smoothly.

Enabling the VNC interface To activate VNC, which allows for graphical access to the Raspberry Pi over the network, run the following commands:

```
sudo systemctl enable vncserver-x11-serviced.service
sudo systemctl start vncserver-x11-serviced.service
sudo raspi-config nonint do_vnc 0
```

Once enabled, the VNC server will launch on boot and be accessible from any device connected to the same local network.

Accessing the Raspberry Pi via VNC Viewer Install a VNC client application (e.g. RealVNC Viewer [33]) on your computer or mobile device. Enter the Raspberry Pi's IP address in the connection field. Upon connecting, you will be asked to enter the username and password of the Raspberry Pi. Once authenticated, the graphical desktop interface will be accessible remotely.

Step 4: Installation and configuration of Tailscale VPN on SBC

To establish a VPN connection, Tailscale [36] is installed on the Raspberry Pi. This setup enables seamless remote access to the unit across different networks without requiring manual port forwarding or static IP configuration.

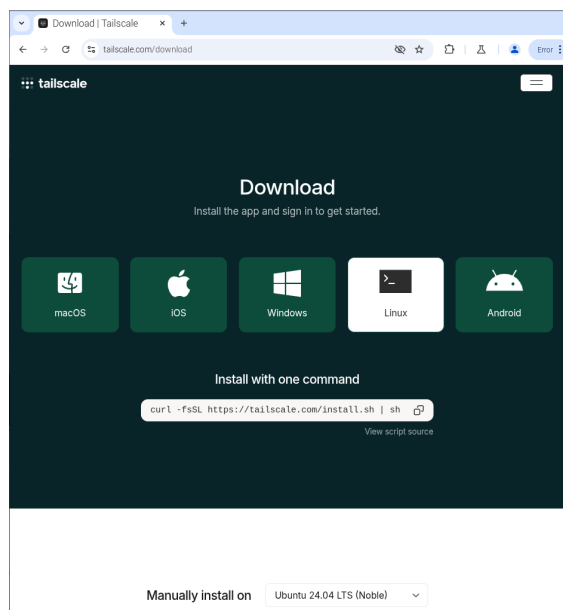
In this system, both the remote monitoring module and the CCU are portable and can be connected from varying locations and networks, including in different countries. Traditional remote access methods would require configuring port forwarding on each intermediate router or firewall, which is impractical. Tailscale,

based on the WireGuard protocol, forms a mesh VPN that bypasses these limitations by allowing direct encrypted connections between authorized devices, regardless of their physical or network location.

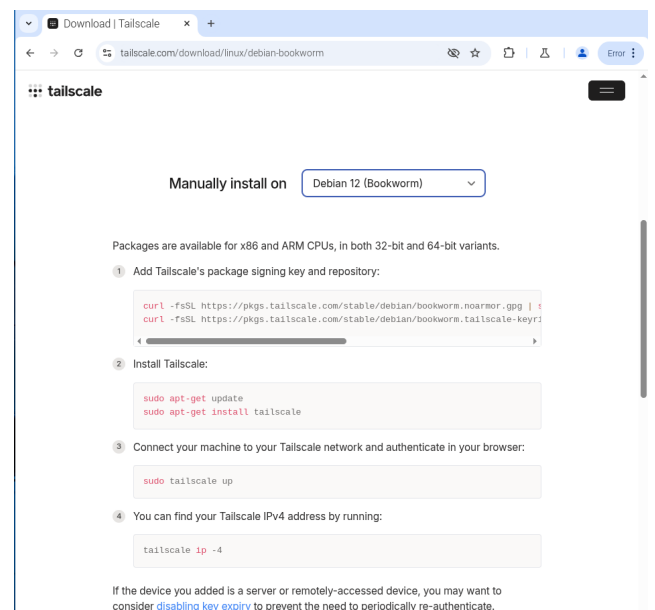
This capability is essential for remotely accessing the Raspberry Pi via SSH to perform file transfers, update software, troubleshoot issues, or monitor system performance. In essence, the VPN provides flexibility, allowing the field unit to remain manageable even when deployed in geographically distributed or network restricted environments.

Begin by opening a web browser on the Raspberry Pi and navigate to <https://tailscale.com/download>. Two installation options are presented (Figure 3.12):

- **One-command installation:** A simple command that automatically installs Tailscale with default settings (Figure 3.12a).
- **Manual installation:** A more customizable method that allows users to select OS-specific packages, offering more control over the installation process (Figure 3.12b).



(a) One-command installation



(b) Manual installation for selected OS

Figure 3.12: Tailscale installation options on the Raspberry Pi.

Once the installation is completed, the system asks the user to authenticate via a web browser to connect the Raspberry Pi to the user's Tailscale network, also known as a tailnet. This process virtually links the device to the private mesh network associated with the Tailscale credentials of the user.

After successful authentication, Tailscale SSH must be enabled to allow secure remote access over the Tailscale network. This can be done by executing the following command on the Raspberry Pi terminal:

```
sudo tailscale up --ssh
```

To confirm that SSH is enabled, look for the 'ssh' label associated with the Raspberry Pi on the Tailscale dashboard.

Step 5: Integrating the Camera Module

In recent versions of the Raspberry Pi operating system, such as the one previously flashed, camera support is enabled by default. The required kernel drivers and the `libcamera` stack are preinstalled, which means that no additional configuration is necessary to activate the hardware interface.

To enable camera functionality for user-level applications, install the corresponding `libcamera` tools [15] by executing the following command in the terminal:

```
sudo apt install libcamera-apps -y
```

This package provides access to a suite of applications such as `libcamera-still`, `libcamera-vid`, and `libcamera-hello`, which allow users to capture images, record video, and preview the camera stream.

Step 6: Integrating the 4G HAT

To enable mobile connectivity on the Raspberry Pi, a 4G HAT is physically installed and configured with the required drivers and utilities.

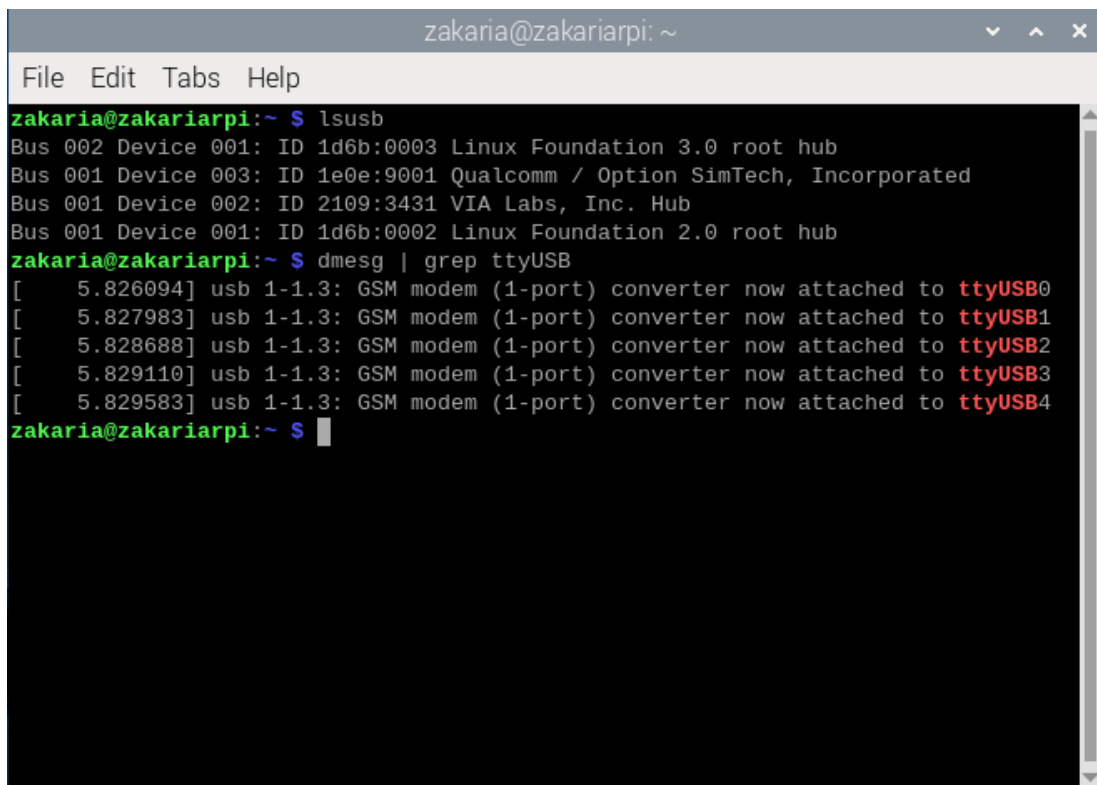
Hardware Setup Before proceeding, ensure that the Raspberry Pi is powered off. Then follow these steps:

1. Insert the SIM card into the designated slot on the 4G HAT.
2. Mount the HAT onto the Raspberry Pi via the GPIO header and the Micro USB interface.
3. Connect the external 4G antenna to the antenna socket.
4. Bridge the power pin to pin D6 to enable automatic startup of the HAT when the Raspberry Pi is powered on.

Initial detection After powering on the Raspberry Pi, check whether the 4G module is recognized by the system:

```
lsusb
dmesg | grep ttyUSB
```

The `lsusb` command lists connected USB devices, while the `dmesg` command shows whether serial ports (e.g., `ttyUSB0`, `ttyUSB1`) are assigned to the modem (Figure 3.13).

A terminal window titled 'zakaria@zakariarpi: ~' showing the execution of two commands. The first command is 'lsusb', which lists several USB devices including root hubs and a Qualcomm modem. The second command is 'dmesg | grep ttyUSB', which shows five log entries indicating that GSM modem converters are attached to ttyUSB0 through ttyUSB4.

```
zakaria@zakariarpi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 1e0e:9001 Qualcomm / Option SimTech, Incorporated
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
zakaria@zakariarpi:~ $ dmesg | grep ttyUSB
[ 5.826094] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB0
[ 5.827983] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB1
[ 5.828688] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB2
[ 5.829110] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB3
[ 5.829583] usb 1-1.3: GSM modem (1-port) converter now attached to ttyUSB4
zakaria@zakariarpi:~ $
```

Figure 3.13: Checking whether the 4G modem is detected by the Raspberry Pi.

Installing required packages To establish communication with the modem and manage the connection, install the following packages:

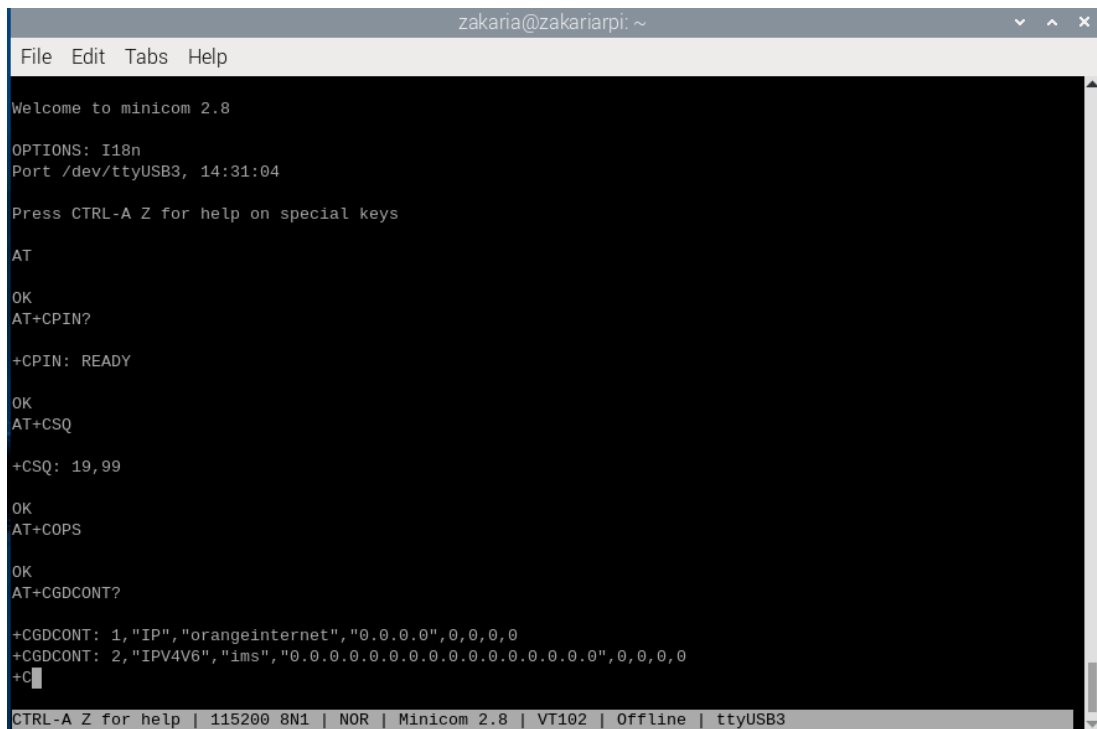
```
sudo apt install modemmanager network-manager usb-modeswitch minicom -y
```

These components serve the following purposes:

- **ModemManager:** Manages communication with the modem [24].
- **NetworkManager:** Simplifies network setup and management [25].

- **usb-modeswitch**: Switches USB mode if needed.
- **minicom**: Terminal emulator for sending AT commands [23].

To verify modem communication, open `minicom` and run basic AT commands. This allows you to confirm that the modem responds to requests and is network-aware (Figure 3.14).



```

zakaria@zakariarpi: ~
File Edit Tabs Help
Welcome to minicom 2.8
OPTIONS: I18n
Port /dev/ttyUSB3, 14:31:04
Press CTRL-A Z for help on special keys
AT
OK
AT+CPIN?
+CPIN: READY
OK
AT+CSQ
+CSQ: 19,99
OK
AT+COPS
OK
AT+CGDCONT?
+CGDCONT: 1,"IP","orangeinternet","0.0.0.0",0,0,0,0
+CGDCONT: 2,"IPV4V6","ims","0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0",0,0,0,0
+C
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.8 | VT102 | Offline | ttyUSB3

```

Figure 3.14: Running basic AT commands inside a `minicom` session.

Creating and activating the mobile connection With the modem detected and operational, a new mobile broadband profile can be created using Network-Manager. Replace `<CONNECTION_NAME>` with a user-defined name and `<ISP_APN>` with the Access Point Name (APN) of the Internet Service Provider (ISP):

```

mmcli -L
sudo nmcli connection add type gsm ifname "*"
con-name <CONNECTION_NAME> apn <ISP_APN>
sudo nmcli connection up <CONNECTION_NAME>
sudo nmcli connection modify <CONNECTION_NAME> connection.autoconnect yes

```

This sequence:

- Registers the GSM modem.

- Establishes a new 4G mobile data connection.
- Configures automatic reconnection at startup.

Connection priority After configuration, the Raspberry Pi will attempt to connect to the internet at boot using the following default priority:

Ethernet → Wi-Fi → 4G

Step 7: Importing project scripts from GitHub

The final step in setting up the remote unit involves importing the required scripts and resources from the project's GitHub repository using Git. This includes code for monitoring, dependencies, and virtual environment setup.

Installing Git First, install Git using the following command:

```
sudo apt install git -y
```

This installs the Git version control system, which is necessary for cloning and managing the project repository.

Creating a local project directory and initializing Git Create a new directory to store the project and initialize it as a Git repository:

```
mkdir <DIR_NAME> && cd <DIR_NAME>  
git init
```

This prepares the directory for connection to the remote repository.

Connecting to the remote repository Add the remote Git repository by replacing <URL> with the actual repository address:

```
git remote add -f origin <URL>.git
```

The repository is available at <https://github.com/ZakariaElGharbaoui/thesis-project>.

Using sparse checkout (Optional but efficient) If only a specific folder (e.g., `monitoring_unit`) is needed from the repository, Git's sparse checkout feature can be used to pull only the relevant content:

```
git sparse-checkout init --cone
git sparse-checkout set monitoring_unit
git pull origin main
```

This approach minimizes data transfer and avoids cloning unnecessary files (Figures 3.15).

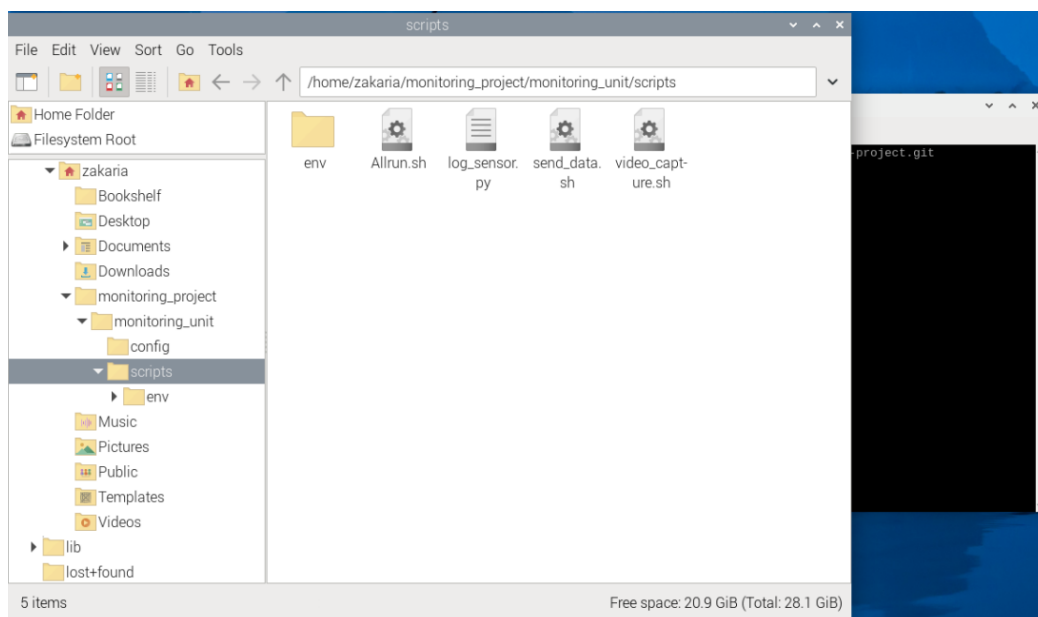


Figure 3.15: Accessing the repository content on the Raspberry Pi after cloning.

With the scripts successfully imported, the remote unit configuration is now complete. The system is ready to execute its assigned tasks according to the requirements of the project.

3.3 CENTRAL LAYER: CENTRAL COMPUTING UNIT

3.3.1 FUNCTIONAL OBJECTIVES AND REQUIREMENTS

The CCU serves as the central hub for data management and coordination within the system architecture. Its primary functional objectives and requirements are as follows:

- **Data reception:** The CCU must reliably receive incoming data streams from remote monitoring units.
- **Data storage:** Upon reception, the CCU must securely store the incoming data.
- **Data preprocessing:** Before further analysis or distribution, the CCU must perform the necessary preprocessing steps.
- **Connection management:** The CCU must manage and maintain stable and secure connections with the remote unit.

3.3.2 PHYSICAL AND SOFTWARE SETUP

For the purposes of this project, the CCU was implemented using a personal computer, which proved sufficient for handling a single communication channel with one remote unit. Given the nature of the project, where real-time data access and immediate processing were not critical, this set-up provided a practical and efficient testing environment.

Hardware Specifications

- Type: Laptop
- Processor: Intel i7-10510U (8) @ 4.900GHz
- RAM: 8 GB RAM
- Storage: 250 GB
- Network Interface: Wi-Fi

The hardware capacity proved sufficient for handling incoming data, executing storage operations, and running preprocessing scripts. **Software Environment**

- Operating System: Ubuntu 24.04.2 LTS
- Key Software Components:

- SSH server and client: Used for secure data reception from the remote unit, as well as remote system management, including updating scripts or modifying configurations.
- File storage: Raw data, primarily in the form of compressed files received from the remote unit, are stored directly in the filesystem for easy access and organization.
- Relational database (MySQL Server [27]): An SQL database is used to manage the processed data associated with the raw data. This includes:
 - * Image IDs
 - * Sensor measurements
 - * Geolocation
 - * API responses
- Preprocessing scripts: scripts that are used to:
 - * Factor incoming video data into individual frames,
 - * Perform background removal to isolate relevant information,
 - * Store processed frames into dedicated database for structured access and future analysis.

3.3.3 DATABASE

A database is an organized collection of data that allows for the storage, retrieval, and management of information. At its core, a database enables users and systems to structure data in a way that supports queries, filtering, linking, and updating, all while maintaining consistency and integrity.

In terms of computer software, these functionalities are offered in varying forms by commonly used tools such as Microsoft Excel, Google Spreadsheet, and MATLAB. Although these applications provide accessible interfaces for handling structured data, they present several limitations when applied to large-scale, automated, or long-term scientific data collection systems.

In the scope of this project, the following data types must be stored in a structured way:

- Captured image files,
- Environmental sensor data (temperature, humidity),
- Time and location metadata,
- Classification responses from external APIs.

3.3.3.1 DBMS: Database Management System

This project adopts a relational database model, implemented through an open-source DBMS. Conceptually, a DBMS resembles the structure of a conventional workbook or spreadsheet, organizing data into:

- Rows and columns,
- Header fields,
- Typed cell values.

However, the key advantage of a DBMS lies in its performance and flexibility. These systems are optimized to handle large datasets efficiently and provide reliable tools for data integrity, concurrent access, and automated management. This makes them more suitable for scientific and engineering applications where the volume of data exceeds the capabilities of traditional spreadsheet software.

All major DBMS platforms, including MySQL, PostgreSQL, and SQLite, are operated using **Structured Query Language (SQL)**, a standardized language used to insert, update, retrieve, and manipulate data across multiple tables. SQL provides capabilities for organizing interconnected records, such as images, sensor data, timestamps, and API responses.

In this project, the selected DBMS is MySQL [27], a widely supported open source solution.

The installation is performed directly from the CCU terminal using the following command:

```
sudo apt install mysql-server -y
```

The package `mysql-server` installs the MySQL database hosting engine, enabling the system to create and manage local databases.

DBMS platforms such as MySQL are composed of two core components:

1. **Client:** The interface used to connect to and interact with the DBMS.
2. **Server:** The background service that stores, processes, and manages access to the databases.

After installation, one can verify that the MySQL server is running with the following command:

```
sudo systemctl status mysql
```

The successful status indicates that the server is active and ready to host the project database locally. Otherwise, it can be started with:

```
sudo systemctl start mysql
```

3.3.3.2 Layout, structure and relations of the database

The database scheme consists of three core tables:

1. ImageData,
2. RemoteMonitoringData, and
3. TaxonomicData.

Each table captures a specific aspect of the dataset, while maintaining a shared primary key reference, `image_id`, which acts as the linking field between them. Figure 3.16 provides a visual representation of the relational structure between the tables.

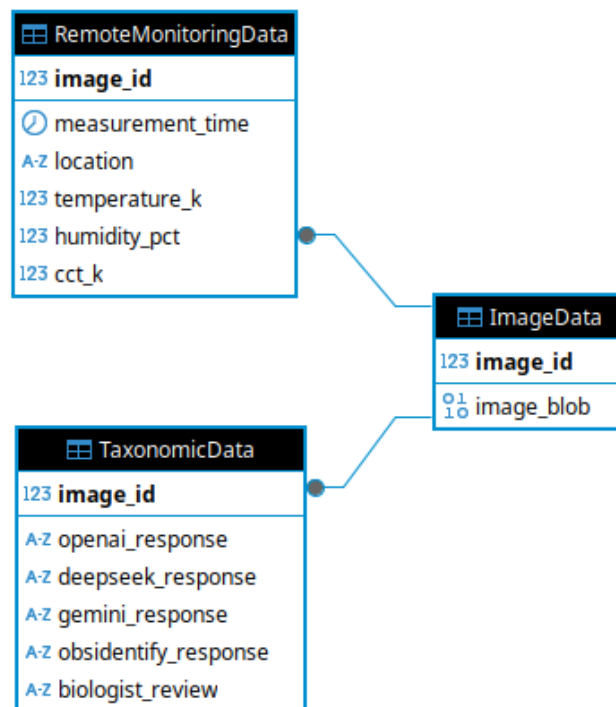


Figure 3.16: Relational structure of the local MySQL database.

- **ImageData:** Holds the binary image files. Each image is stored as a **BLOB (binary large object)**, with a unique `image_id` serving as the primary key. This table serves as the anchor for all linked metadata.

- **RemoteMonitoringData:** Contains the environmental measurements and metadata captured during acquisition. It includes:
 - `measurement_time` (timestamp of recording),
 - `location` (latitude and longitude as a string),
 - `temperature_k` and `humidity_pct` (sensor readings),
 - `cct_k` (correlated color temperature).

Each entry references its corresponding image via the `image_id` foreign key.

- **TaxonomicData:** Stores the classification responses of various image recognition services (OpenAI, DeepSeek, Gemini, ObsIdentify) and manual annotations. Each field contains the response in text form, and the table uses the same `image_id` as the primary key to link the classification results with the original image.

This schema design preserves one-to-one relationships between images and their associated data. The use of a consistent primary key across tables ensures relational consistency and simplifies data handling.

To validate the table structure and inspect the schema, the MySQL CLI was used. Figure A.1 shows the terminal session displaying the available databases, selected schema, and column descriptions for each table.

3.3.4 DATA FLOW

3.3.4.1 Raw data reception and storage

Incoming data from the remote unit are transmitted periodically as compressed archives via SSH using `rsync`. Each archive contains:

- Video sequences captured by the monitoring unit,
- Sensor measurements (temperature, humidity),
- Associated metadata (timestamps, geolocation).

Upon reception, the archives are automatically extracted. This process is automated through a Python script (Listing B.8) that continuously monitors the designated raw data directory using the `watchdog` library. When a new `.tar.gz` archive is detected, it is first extracted into a corresponding folder within the raw directory. The archive contains a short video sequence, along with a text file containing synchronized sensor readings, time, and geolocation.

Once extraction is complete, the script initiates a frame factorization routine using the `OpenCV` library. This routine reads the video file and sequentially converts

it into individual JPEG frames, which are then saved into a corresponding folder within the `processed` directory. The text file present is also copied into that same folder, preserving the association between visual data and sensor data.

By using this pipeline, the system ensures that incoming data are decomposed and structured immediately upon arrival, enabling subsequent indexing and database ingestion steps to operate on already-processed data.

Once frame extraction is complete and the data are moved to the `processed` directory, a second Python script (Listing B.7) is triggered, again using the `watchdog` library to handle database ingestion. This script continuously monitors the processed directory for newly added folders and, upon detection, performs a two-stage operation.

First, it reads the accompanying sensor log file (in `.txt` format), parsing weather-related values such as temperature, humidity, timestamp, and geolocation into structured rows. Then, it iterates over the extracted image files and associates each image with a corresponding sensor reading.

Each image is inserted as a BLOB into the `ImageData` table, and the corresponding environmental measurements are inserted into the `RemoteMonitoringData` table. These entries are linked via the `image_id`, which is generated automatically by MySQL upon image insertion and retrieved using the `cursor.lastrowid` mechanism. This ensures globally unique identifiers even across system reboots, database resets, or partial data deletions.

The use of a consistent, auto-incremented primary key (`image_id`) as the relational link across tables guarantees data integrity and prevents ID reuse unless explicitly overridden. By automating the pairing and insertion of visual and sensor data in this way, the system achieves real-time alignment of heterogeneous data streams within a unified relational model.

3.4 CLOUD-LAYER: RECOGNITION API

The cloud-layer of the monitoring system acts as the final stage in the automated data processing pipeline. Its primary purpose is to perform species recognition on images captured by the remote monitoring unit, using cloud-based (artificial intelligence) services. This layer is hosted in the CCU, which is responsible for fetching image data from the database, executing recognition tasks through external APIs, and logging the classification results.

3.4.1 ARCHITECTURE AND FUNCTION

The cloud-layer does not run on the remote monitoring unit, but on the physically separate CCU with higher computational and storage capabilities. Once an image is uploaded by the monitoring unit in the CCU, and logged in the database with the appropriate metadata, a recognition script [B.9](#) on the CCU retrieves the image and submits it to the recognition APIs. Each image is processed sequentially, and the results are stored directly in the dedicated columns of the database. This approach centralizes recognition efforts and avoids overloading the monitoring unit with heavy network or compute tasks.

3.4.2 RECOGNITION APIS USED

Three APIs were considered in this project:

- **ObsIdentify API:** A specialized application programming interface developed for the automatic recognition of flora and fauna. It is the most domain-specific tool among those evaluated. The API supports a limited number of calls per day (10 calls/day on the public endpoint) and returns a JSON object containing the most probable species along with confidence scores. The images are submitted using the `requests.post()` method, with specific metadata and application parameters.
- **OpenAI GPT API:** The GPT API is used as a general-purpose vision-language model. Images are encoded in base64 and submitted through prompt-based queries. Although not specialized in species identification, it can describe visual features and offer broad categorizations. Responses are parsed and stored as raw textual outputs.
- **Gemini API (Google):** Similar to GPT, Gemini supports multimodal input and is tested here for its ability to infer taxonomic categories from JPEG images. Input formatting and result parsing follow the standard Gemini interface.

3.4.3 DATA FLOW AND LOGGING STRATEGY

The image recognition script iterates over new database entries that do not yet contain recognition results. Each image is submitted to the chosen APIs, and the responses are parsed and written to the appropriate row in the database. This logging mechanism supports reproducibility, further filtering, and future re-processing if improved models are integrated.

For ObsIdentify, key outputs such as scientific name, common name, and confidence score are extracted. For OpenAI and Gemini, the outputs are less structured and are stored as free-form text.

3.4.4 LIMITATIONS AND MOTIVATION FOR AI INTEGRATION

Although ObsIdentify offers high domain-specific accuracy, its license (~10,000 EUR/year) presents a significant barrier for low-cost deployment. The daily call limitation of its free endpoint further restricts feasibility for continuous monitoring.

In the scope of Section 1.3.1, this thesis investigates whether general-purpose AI systems such as OpenAI's GPT and Google's Gemini can serve as low-cost alternatives for basic recognition tasks. Although these systems lack field specialization, they offer potential, or suggest broad classifications, a trade-off that may suffice for certain ecological indicators or exploratory citizen science applications.

3.4.5 CASE STUDY: EVALUATING GENERAL-PURPOSE AI FOR INSECT RECOGNITION

Given the high licensing cost of ObsIdentify and its strict limitation of 10 recognition calls per day via the free endpoint, it is not feasible to conduct a full-scale measurement campaign within the time constraints of this thesis. Such a campaign would typically require the deployment of the remote monitoring unit in the field, the selection of a representative site, the justifying of its ecological relevance, and the collection and validation of a substantial number of insect images. Therefore, to evaluate the potential of general-purpose AI tools for insect identification, a controlled case study was conducted using pre-existing images.

The goal of this case study is to assess the feasibility and performance of general-purpose AI models, specifically:

- OpenAI `gpt-4o-mini`,
- Google `gemini-2.0-flash-exp`,

in classifying insect species from images. Both models support image input via vision prompts and provide text-based outputs.

Dataset preparation. To create a standardized and taxonomically diverse test set, images were selected from the ObsIdentify database. Three insect orders were chosen: *Lepidoptera*, *Diptera*, and *Coleoptera*. For each order, three representative images were selected, each with a resolution of 1000x1000 pixels. The resulting dataset was organized into a structured folder hierarchy as follows:

```
case_study_images/  
|-- lepidoptera/  
|   |-- image1.jpg  
|   |-- image2.jpg  
|   |-- image3.jpg  
|-- diptera/  
|   |-- image1.jpg  
|   |-- image2.jpg  
|   |-- image3.jpg  
|-- coleoptera/  
    |-- image1.jpg  
    |-- image2.jpg  
    |-- image3.jpg
```

Resolution testing. To evaluate how image resolution affects recognition performance, each image was progressively downscaled using a Python script. This allowed testing of model robustness under varying quality conditions that may reflect real-world limitations of low-cost hardware.

Prompt design. For consistency, the same structured prompt was used for both AI systems:

"Given the output of an insect recognition model, list each identified species. Extract the scientific name and, beside it, the full taxonomic hierarchy from order down to species (Order-Family-Genus-Species)."

Although these models are not designed for strict biological classification, the aim was to measure their ability to extract and return structured taxonomic information in response to a biologically focused prompt.

Execution and evaluation. Each model was queried individually using Python scripts that interface with the respective APIs. The output was manually parsed and logged in a spreadsheet for each resolution level. The responses were compared against the known taxonomy of each insect to determine the maximum correct taxonomic level (Order, Family, Genus, Species) reached by each model.

For OpenAI, the newly released `/v1/responses` endpoint was used. This endpoint, introduced in March 2025 and still in beta at the time of writing, offers a more flexible and customizable framework compared to the traditional `chat/completions` endpoint. It enables building domain-specific AI agents or tools that can integrate external knowledge, making it a promising development for future applied ecological AI systems. However, its evolving documentation, experimental status, and recent SDK changes introduced some additional complexity in implementation.

Challenges and Opportunities. One of the notable challenges in working with general-purpose AI models is the rapid pace of development. Documentation, API endpoints, and model behaviors are updated frequently, requiring ongoing adaptation of the scripts and infrastructure used. However, this dynamic landscape also opens opportunities to build tailored tools with more domain awareness, especially if integration with structured external datasets becomes possible.

In the next chapter, results from the full-resolution case study are presented, followed by a comparison across lower resolutions to analyze the effect of image resolution on AI classification performance.

4.1 COMPARATIVE PERFORMANCE OF GENERAL-PURPOSE VISION MODELS FOR INSECT RECOGNITION

4.1.1 OBJECTIVE

I compared two general purpose vision language models, OpenAI gpt4omini and Google gemini 2.0flashexp, on a simple but practical task:

"given a single field photograph of an insect, can the model report the Order, Family, Genus, and Species?"

The goal is not to crown a winner but to understand what level of taxonomy these models can reliably reach and how image resolution affects that ability.

4.1.2 DATASET

The dataset consisted of nine insect images, evenly balanced across three major Orders: Lepidoptera, Diptera, and Coleoptera (three images per Order). For each image, authoritative taxonomic labels from ObsIdentify were available down to the Species level. These served solely as reference points for evaluation; the AI models did not have access to these labels during inference.

To assess the effect of image resolution on recognition, each photograph was pre-processed into five square resolutions:

- 100% (1000 × 1000 px; “full-scale”)
- 75% (750 × 750 px)
- 50% (500 × 500 px)
- 25% (250 × 250 px)
- 10% (100 × 100 px)

Both models were tested using the same concise prompt that instructed them to return the Order, Family, Genus, and Species in plain text.

4.1.3 SCORING AND METRICS

The predictions were scored against the reference images at each rank.

- Per-rank accuracy (Order/Family/Genus/Species): share of images where the prediction exactly matches the reference at that rank.
- Rank-depth score (0–4): for each image, we count how many consecutive ranks are correct starting from Order. Example: if Order and Family are correct but Genus is wrong, the image gets a score 2. We report the average of this score per model and resolution. (This is the “Average rank accuracy” in the workbook.)
- “Reached at least Family/Genus”: share of images where the model is correct to that depth or deeper (useful for practical thresholds).

These metrics are easy to read, correlate well with how the results feel in practice, and avoid overinterpreting a small sample.

4.1.4 RESULTS

Table 4.1: Summary matrix of AI recognition performance across five image resolutions. The table reports the average taxonomic rank accuracy (mean correct rank position) and the average percentage of correct identifications at the Order, Family, and Genus levels for OpenAI and Gemini models.

metric	100%	75%	50%	25%	10%
<i>rank accuracy</i>					
OpenAI	2.56	1.67	0.89	0.67	0.33
Gemini	2.11	1.33	1.22	1.33	1.00
<i>order correctness (%)</i>					
OpenAI	100.0	88.9	66.7	44.4	22.2
Gemini	100.0	100.0	100.0	100.0	66.7
<i>family correctness (%)</i>					
OpenAI	88.9	55.6	33.3	33.3	11.1
Gemini	77.8	33.3	22.2	33.3	33.3
<i>genus correctness (%)</i>					
OpenAI	55.6	22.2	0.0	0.0	0.0
Gemini	22.2	0.0	0.0	0.0	0.0

4.1.4.1 Full-resolution: 1000×1000 px

- Both models reliably detect the Order (100% in this sample).

- Family: OpenAI is higher ($\approx 89\%$) than Gemini ($\approx 78\%$).
- Genus: OpenAI again leads ($\approx 56\%$) over Gemini ($\approx 22\%$).
- Species: 1 case for both matches the exact Species in these unconstrained field photos.
- Average rank-depth: OpenAI 2.56 vs Gemini 2.11 (that is, on average, OpenAI reaches between Family and Genus; Gemini, around Family).

At high resolution, both are good at Order; OpenAI is notably better at Family and Genus, but Species level is, generally, out of reach for this setup.

4.1.4.2 Effect of downscaling

- As images shrink, information disappears quickly. The Family and Genus accuracy drop for both models.
- Order remains relatively robust, especially for Gemini (which holds Order well to 250 px in this sample), while OpenAI's Order accuracy declines earlier.
- The rank-depth curves (Figure 4.1) mirror this: OpenAI leads at high resolution but Gemini overtakes at lower resolution, by maintaining Order when OpenAI starts to miss it.

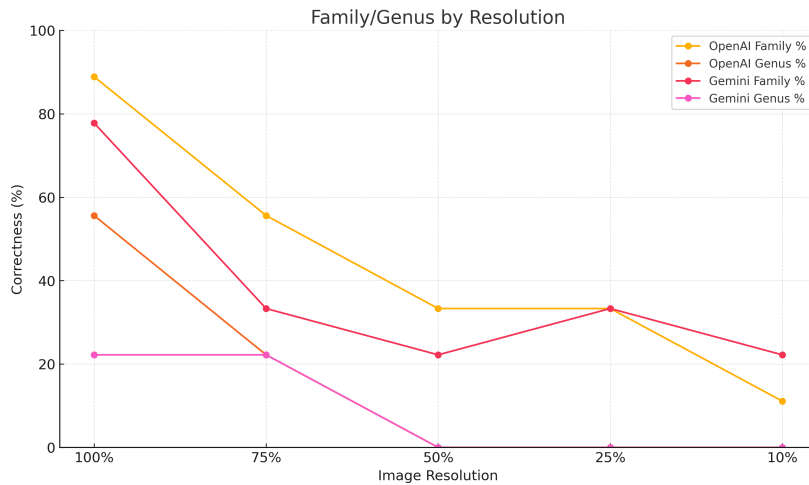
OpenAI showed better results at Family/Genus when resolution is high, the rank-depth is best at full scale. Gemini, on the other hand, is more robust at Order as resolution drops, and it works better at lower resolutions mainly by preserving Order.

4.1.4.3 Limitations

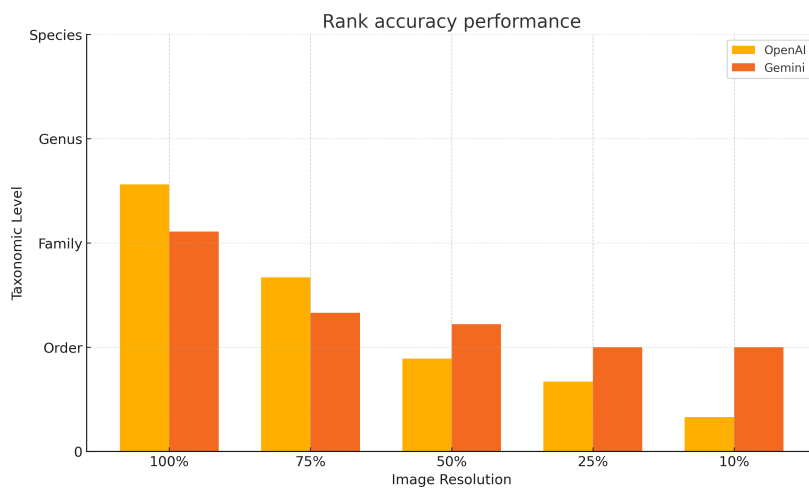
This case study was based on a small sample of just nine insect images, representing a limited range of taxa. As such, the findings should be considered indicative rather than definitive.

Model performance was assessed using a strict exact-match approach, any deviation from the authoritative label, including valid synonyms or near identifications at the Genus or Species level was counted as incorrect.

Each image was processed using a single prompt and one attempt per model. Alternative prompt structures or adjustments to model parameters could potentially influence outcomes.



(a) Family/Genus correctness (%).



(b) Average rank accuracy (Order → Species).

Figure 4.1: Effect of image downscaling on AI recognition (nine test images). (a) Exact matches at Family and Genus levels for OpenAI and Gemini across five resolutions: both decline with lower resolution; OpenAI leads at Family for higher resolutions, but Genus accuracy drops to 0% below 50%. (b) Mean highest correctly identified rank (Order < Family < Genus < Species): accuracy peaks at full resolution and degrades with downscaling; OpenAI is stronger at high resolutions, Gemini more stable at lower ones.

Table 4.2: Shapiro-Wilk Test Results: Tests indicated that the normality assumption was violated in more than half of the tested conditions ($p < 0.05$). Therefore, it was preferable to also consider the non-parametric Friedman tests as a more robust alternative.

resolution	tool	Shapiro statistic	p-value
100	Gemini	0.82	0.04
100	OpenAI	0.56	3.6×10^{-5}
250	Gemini	0.62	1.5×10^{-4}
250	OpenAI	0.71	0.002
500	Gemini	0.54	1.7×10^{-5}
500	OpenAI	0.82	0.04
750	Gemini	0.62	1.5×10^{-4}
750	OpenAI	0.90	0.27
1000	Gemini	0.85	0.07
1000	OpenAI	0.91	0.34

4.1.4.4 Statistical significance: RM-ANOVA test

A key limitation of this case study lies in the small sample size. In other words, while the results suggest that image resolution strongly affects recognition and that the two models behave differently at certain levels, it is not clear whether these patterns would generalize beyond this dataset.

RM-ANOVA is a statistical test that provides more insight into whether the observed trends are likely to hold in a larger population, rather than being exclusive to the small dataset. This method is specifically suited for cases where the same subjects are measured repeatedly under different conditions, which corresponds to the present design in which each insect image was tested across multiple resolutions and with two recognition tools.

The statistical analysis confirms what the descriptive results already suggested, image resolution is the dominant factor shaping recognition performance. Both the parametric RM-ANOVA and the non-parametric Friedman tests agreed that recognition depth drops significantly as resolution decreases. Refer to Appendix A for complete results.

This chapter steps back from the single case study to synthesize what the thesis reveals about building a low-cost field-deployable monitoring unit to study ALAN-insect interactions.

5.1 WHAT THE RESULTS MEAN FOR THE PROJECT

5.1.1 CAPABILITY ENVELOPE

General purpose vision-language models proved reliable in identification of the *Order* level, often usable at *Family* when the images were sufficiently detailed and only rarely reliable beyond *Genus* (see Table 4.1). In this small sample, two exact species matches occurred at full resolution (one per model on the same image), underscoring that species-level hits are possible but not yet reliable without specialization.

5.1.2 RESOLUTION MATTERS

Based on the findings of the case study, the taxonomic depth of AI-based classifications decreases noticeably when image resolution is reduced. Although the overall *Order*-level remains relatively robust under such downscaling, finer taxonomic levels are affected more strongly (see Figure 4.1). It should be noted that this resolution threshold refers to the image as a whole and not specifically to the pixel coverage of the insect within the frame.

5.1.3 BALANCING AI ASSISTANCE AND EXPERT VALIDATION

In the context of this project, VLMs proved to be more valuable when used as broad preliminary taggers. They can provide initial categorizations, but their output should be treated as guidance rather than as definitive identifications. When precise and authoritative species labels are required, the role of a specialized species-recognition service, or a human expert, remains irreplaceable. In this way, AI should serve to streamline the process, while the final taxonomic authority is retained by domain specialists.

5.1.4 AUTO-CRITICAL REFLECTIONS

5.1.4.1 Case-study scope

The case study is based on a limited set of three images for a limited set of three orders of insects (lepidoptera, diptera, and Coleoptera), and the images themselves are relatively ideal, with minimal blur and occlusion. Consequently, the results should be interpreted as indicative trends rather than definitive conclusions, offering preliminary insights that would require validation on larger and more diverse datasets.

5.1.4.2 Prompting assumptions

Each image was queried once using a single, non-optimized prompt. Prompt engineering, the practice of structuring and refining queries to guide AI models toward more accurate or relevant output, can significantly influence classification performance. In this study, no particular attention was paid to optimize the prompt or adjust the request parameters.

5.1.4.3 API instability

AI-based vision APIs are evolving at a pace that matches the rapid emergence of new AI tools, making long-term integration inherently challenging. Endpoints, request formats, and available parameters can change significantly between versions. For example, the OpenAI endpoint used in this project was modified twice during the development period, with the latest iteration adopting a completely different structure and introducing tailoring parameters that were previously unavailable. Such shifts require continuous adaptation and reinforce the importance of designing systems with thin adapters, version pinning, and explicit fallbacks to maintain operational stability, hence the relevance of the GitHub project page for this work.

In this work, the selected APIs were primarily those positioned for the broader public rather than niche professional markets. It is highly likely that low-cost, recognition-oriented AI APIs already exist or will soon emerge in more specialized domains. Identifying and evaluating such services could yield substantial benefits in terms of cost efficiency and accuracy, making this a key consideration for future work.

5.1.4.4 Strategic pivot

Following an unaffordable ObsIdentify licensing proposal, the project expanded to evaluate general-purpose vision APIs under realistic cost constraints. This changed the contribution from "device-only" to also include a comparative methodology for labeling strategies.

5.1.4.5 Unknown necessary taxonomic depth

In the scope of this thesis, the minimally sufficient rank for detecting ALAN effects remains unclear. If behavior converges by *Order*, species-level recognition may be an unnecessary overhead. If the effects diverge by *Family* or *Genus*, a higher taxonomic resolution becomes essential. This uncertainty makes it difficult to provide a definitive answer to the research sub-question.

If the effects of ALAN on nocturnal insects converge by *Order*, then the system developed here already enables the study of the insect-ALAN dynamics while addressing key methodological gaps. If finer taxonomic resolution is required, this becomes a priority to clarify in order to make correct engineering decisions. Given that low-cost accessibility is a core motivation of this project, over-engineering beyond what is necessary would undermine that goal.

5.1.5 LOW-COST DESIGN: COMPONENTS AND COSTS

The prototype emphasizes affordability while remaining field-capable. The costs are listed in Table 5.1, and the prices vary by vendor and availability.

Table 5.1: Indicative bill of materials for the on-site unit.

subsystem	part	cost (€)
computer	Raspberry Pi 4B (4GB)	72.16
camera	Pi Camera Module v3 (NOIR)	35.92
connectivity	WaveShare SIM7600G-H	102.32
microcontroller	Arduino Uno	30.00
sensors	AHT20	12,09
power	240 VAC → 5 VDC supply	17.99
enclosure	IP65	64.38
misc.	IR-LED	11.99
total		346.85

This chapter outlines potential next steps so that the work remains extensible as hardware, APIs, and scientific questions evolve. Advancing this system from a functional prototype to a reliable and scalable monitoring platform requires coordinated progress across the scientific, algorithmic, and engineering domains. The following priorities are proposed for the future.

6.1 FUTURE WORK

6.1.1 CLARIFYING THE REQUIRED TAXONOMIC DEPTH

A key scientific question that remains unresolved is the minimally sufficient taxonomic rank needed to detect ALAN effects with ecological relevance. If impacts converge at the *Order* level, current capabilities may already meet the requirement. If effects diverge at *Family* or *Genus*, higher-fidelity imaging and more specialized classification tools will be necessary. Determining this threshold should be the first priority, as it will guide all subsequent design decisions.

6.1.1.1 Targeting appropriate recognition tools

Once the required taxonomic depth is established, the recognition pipeline should be adapted accordingly. This includes selecting or developing AI models and algorithms. Whether wide-public VLMs, fine-tuned species classifiers, or hybrid approaches, that can reliably achieve the necessary depth under field conditions.

6.1.1.2 Extending recognition to counting

After achieving consistent classification performance at the required taxonomic resolution, the next step is to extend the functionality from recognition to counting. This requires detecting and classifying multiple insects in the same frame. Such multi-object recognition will be essential for ecological metrics beyond simple presence/absence.

6.1.1.3 Developing an abundance metric

In traditional light traps, insects are killed and counted, ensuring that each specimen is counted only once. In a non-lethal, image-based system, the same insect

may appear in multiple frames, risking overestimation. Future work should explore algorithms or indices, possibly based on screen time or recurrence analysis, that can quantify abundance in a way that is biologically meaningful and methodologically consistent.

6.1.1.4 Production-ready deployment architecture

From a production standpoint, containerization (e.g., Docker) should be employed to ensure a standardized and reproducible monitoring unit. This requires careful consideration of what functions remain on the SBC versus inside containers, including interpreters, drivers, hardware access, and permission management. Although a preliminary trial with containerization has been made, it served primarily to highlight the complexity of this task rather than to provide a ready-to-use strategy. Creating a robust container architecture is not as simple as 'placing the system in Docker'. It requires a good understanding of the operation of the monitoring unit and a careful allocation of responsibilities between the SBC and containerized environments. Practical experience will be essential for anticipating which components are prone to version conflicts and for designing an architecture that can evolve without breaking core functionality.

6.1.1.5 Secure and accessible data infrastructure

The current local PC database should be migrated to a secure and accessible storage solution, such as a managed cloud service or a Network Accessible Storage (NAS). This migration must include proper access control, encryption in transit and at rest, automated backups, and monitoring. Given that remote units will be deployed outdoors and connected to networks, cybersecurity measures to protect access and prevent unauthorized intrusion are essential.

In summary, the prototype has shown that a sub-€ 500 on-site unit can already support *Order*-level analysis at scale. The priorities above outline the scientific and engineering steps needed to refine this into a reliable, secure, and scalable platform, capable of delivering ecologically meaningful insights into ALAN–insect dynamics.

Artificial Light at Night is increasingly recognized as an ecological stressor, but long-term, standardized, and scalable monitoring of nocturnal insect responses to ALAN remains limited. This thesis set out to narrow that gap by designing a low-cost, field-deployable monitoring unit and accompanying system that can collect the measurements necessary to study ALAN–insect dynamics under real-world conditions. Guided by the PICO framework, the research question asked:

How do different LED lighting parameters affect the abundance and community composition of positive phototactic nocturnal insects in outdoor, suburban, rural, and semi-natural habitats under real-world field conditions?

Recognizing that fully answering this ecological question requires season-scale datasets across multiple sites and lighting regimes, the present work focused on the crucial methodological sub-question:

Is it feasible to provide a low-cost, camera-based monitoring solution that can quantify the abundance and community composition of positive phototactic nocturnal insects across habitats with contrasting LED lighting parameters?

7.1 SUMMARY OF CONTRIBUTIONS

7.1.1 A LOW-COST, FIELD-READY PLATFORM

This thesis presents a functional on-site monitoring unit built around a SBC, a camera, a 4G modem, and a MCU for sensor integration. The system combines weather measurements, geolocation data, lighting parameters, and image capture of insects attracted to the light source. The indicative material bill for the core unit remains under the € 500 threshold (for detailed pricing information, see Table 5.1.), making it affordable for large-scale deployment and straightforward replication.

7.1.2 OPERATIONAL WORKFLOW AND GOVERNANCE

The system implements an *AI-assisted, human-confirmed* labeling workflow. Every record stores the image and metadata, while a dedicated field allows biologists to enter the authoritative taxonomic classification. VLMs are retained as advisory tools rather than sources of record.

7.1.3 IMAGING RESOLUTION AND VLM PERFORMANCE IN CONTEXT

A small controlled case study evaluated two general-purpose vision–language models to explore how image resolution influences taxonomic accuracy. Both models consistently achieved reliable identifications at the *Order* level, even when the image resolution was decreased. However, moving to *Family* or *Genus* required higher image detail, and performance dropped sharply in unconstrained field images.

In particular, both models achieved an exact species match on the same image when provided with full-resolution data, showing that species-level identification is possible under ideal conditions. However, such outcomes remain inconsistent without model specialization, particularly in real-world field settings.

These results suggest that VLMs can serve as cost-effective support tools within a broader workflow, delivering *Order*-level classifications and occasional deeper matches, while keeping taxonomic authority with human experts. This approach also leaves open the possibility of integrating specialized species-recognition services in the future as needs and resources evolve.

7.2 ANSWER TO THE SUB-QUESTION

Feasibility is demonstrated. This work has produced both a complete and reproducible system design and a fully functioning prototype, showing that low-cost field-ready monitoring of nocturnal insect activity under varying artificial light conditions is not only possible, but also practical. The prototype brings together key capabilities:

1. A system that receives and stores incoming data from the field unit, and organizes it so that images and metadata are stored in a database accessible both for AI recognition scripts and for review by biologists or other experts (see Figure 7.1),
2. The integration of AI as a supportive tool without undermining the authority of human validation.

The screenshot shows the DBeaver SQL GUI tool interface. At the top, there are tabs for 'on_site_monitoring_system_data', 'TaxonomicData', 'ImageData', and 'RemoteMonitoringData'. The 'TaxonomicData' table is selected, showing a single row with columns: 'image_id' (320), 'openai_response' (Lepidoptera Noctuidae Helicoverpa Helicoverp), 'geminai_response' (Lepidoptera Noctuidae Amf), 'obsidentify_response' ('api_implementation': 'tag', 'api-v2-2': [NULL]), and 'biologist_review'.

The 'ImageData' table is shown in the bottom right, with columns: 'image_id', 'image_blob', and 'image_c'. It contains rows with image IDs 444 through 450 and corresponding image blobs.

The 'RemoteMonitoringData' table is shown in the bottom left, with columns: 'measurement_time', 'location', 'temperature_k', 'humidity_pct', 'cct_k', and 'image_id'. It contains rows with measurement times from 2025-06-22 13:08:46 to 2025-06-22 13:08:52 and various environmental data points.

Figure 7.1: Database structure and content in DBeaver SQL GUI tool: (Top) `TaxonomicData` table showing API responses for a sample image as well as an empty column for authority label. (Bottom left) `ImageData` table containing stored image BLOBs. (Bottom right) `RemoteMonitoringData` table containing environmental meta-data linked to image IDs.

These elements combine to create a remote monitoring unit that, as a hardware and data acquisition subsystem, is ready for deployment in season-long studies across diverse sites and lighting environments. What remains a task for future work is refining the AI-based recognition pipeline so that large-scale, multi-site deployments can deliver consistently reliable taxonomic outputs. In its current form, the system can be deployed immediately for data collection, while the classification component shall continue to improve.

7.3 PROGRESS TOWARD THE MAIN QUESTION

The main ecological question, how LED lighting parameters influence nocturnal insect abundance and community composition, cannot be answered on the basis of a single pilot study and a small-scale VLM evaluation. However, this work has laid the foundation needed to address it in the future.

A key outcome is the development of a low-cost monitoring unit that can be replicated. The next step is to build on this foundation by pursuing the future work outlined in Chapter 6. Iterating on the prototype until it delivers reliable performance in both recognition and counting, and ultimately conducting long-term, multi-site studies under controlled lighting conditions.

7.4 OBJECTIVE-BY-OBJECTIVE REFLECTION

Objective 1: Critically analyze and synthesize the existing scientific literature on ALAN and its effects on nocturnal insects.

The literature review synthesized, according to the PRISMA procedure, current knowledge about ALAN impacts, highlighted taxonomic and methodologi-

cal blind spots, and motivated the need for standardized and scalable monitoring under field conditions.

Objective 2: Develop a low-cost system to collect reliable data on the impact of ALAN on nocturnal insect communities.

A functional first prototype was developed and implemented that allowed the collection of climate (temperature and relative humidity), geographic, lighting (CCT), and image data. AI-based taxonomic suggestions were integrated as auxiliary outputs. Although the system met functional requirements, there is still a significant scope for improvement in hardware robustness, data management, and classification accuracy (see Chapter 6).

Objective 3: Design and implement a monitoring system on site for the observation of nocturnal insects.

The system incorporated defined resolution thresholds to ensure that the collected images met minimum quality requirements for ecological relevance. However, human validation remains necessary for species identification, with current AI tools serving as an aid rather than a full replacement for expert classification.

Objective 4: Describe potential future improvements and directions for continued research (see Chapter 6).

Future work priorities have been clearly defined to guide scientific and engineering development. On the scientific side, the first step is to determine the minimally sufficient taxonomic depth for detecting ALAN effects, which will inform the choice of recognition tools and the level of imaging fidelity required. Once reliable classification at this depth is achieved, the system should evolve to count multiple insects per frame and develop a robust abundance metric suited to non-lethal monitoring.

On the engineering side, the next steps include designing a production-ready containerization strategy that carefully allocates functions between the bare SBC and containerized environments, migrating the database to a secure and accessible cloud or NAS infrastructure, and implementing strong cybersecurity measures for remote deployments. These directions aim to improve the scalability, maintainability, and reliability of the platform while keeping costs low and ensuring that the system remains adaptable to evolving AI and recognition capabilities.

RM-ANOVA test results

Repeated-measures ANOVA (RM-ANOVA) is a statistical method that assesses whether observed trends are likely to generalize to a larger population rather than being artifacts of a small dataset. This method is particularly suitable for designs in which the same subjects are measured repeatedly under different conditions, as in the present case where each insect image was tested across multiple resolutions and with two recognition tools.

The procedure begins with a normality check. If the assumption of normality is met, RM-ANOVA can be applied; if not, a non-parametric alternative such as the Friedman test is more appropriate. When significant effects are detected, post-hoc Wilcoxon rank tests are used to identify which specific pairs of conditions differ.

Table A.1: Shapiro-Wilk test results

resolution (px)	tool	Shapiro statistic	p-value
100	Gemini	0.82	0.0373
100	OpenAI	0.56	0.000036
250	Gemini	0.62	0.000153
250	OpenAI	0.71	0.001894
500	Gemini	0.54	0.000017
500	OpenAI	0.82	0.03728
750	Gemini	0.62	0.00015
750	OpenAI	0.90	0.2728
1000	Gemini	0.85	0.06785
1000	OpenAI	0.91	0.33793

Normality check. The Shapiro-Wilk tests (Table A.1) indicated that the normality assumption was violated in more than half of the tested conditions ($p < 0.05$). Consequently, the Friedman test was also considered as a more robust alternative.

Non-parametric check (Friedman). Both tools showed significant performance drops when moving from high to low resolutions, confirming that the effect of resolution is robust even without normality.

Table A.2: Chi-square test results

tool	χ^2	p-value	interpretation
Gemini	10.41	0.034	resolution effect confirmed
OpenAI	15.48	0.004	resolution effect confirmed

Table A.3: Pairwise Comparison Results (Gemini vs OpenAI)

resolution (px)	comparison	statistic	p_{raw}	p_{adj}	$p < 0.05$
100	Gemini vs OpenAI	2.5	0.156	0.781	False
250	Gemini vs OpenAI	3.0	0.188	0.781	False
500	Gemini vs OpenAI	5.0	0.750	0.781	False
750	Gemini vs OpenAI	1.5	0.375	0.781	False
1000	Gemini vs OpenAI	0.0	0.250	0.781	False

Tool comparisons (Wilcoxon). No significant differences were found between Gemini and OpenAI at any resolution.

Table A.4: Descriptive Statistics by Tool and Resolution

tool	resolution (px)	mean	std. dev.	count	SE
Gemini	100	1.000	0.866	9	0.289
Gemini	250	1.333	0.500	9	0.167
Gemini	500	1.222	0.441	9	0.147
Gemini	750	1.333	0.500	9	0.167
Gemini	1000	2.111	0.928	9	0.309
OpenAI	100	0.333	0.707	9	0.236
OpenAI	250	0.778	0.972	9	0.324
OpenAI	500	1.000	0.866	9	0.289
OpenAI	750	1.778	0.972	9	0.324
OpenAI	1000	2.556	0.882	9	0.294

LTE modem

An LTE modem is a hardware device that enables embedded systems (e.g. Raspberry Pi, Arduino) to connect to the Internet through cellular networks such as LTE, a fourth-generation (4G) wireless communication standard.

Connecting to the Internet means obtaining access to the global IP network, which enables the system to:

- Send data to remote servers or services.
- Receive data from other systems.

The LTE modem establishes this link by communicating via radio waves with a nearby cell tower. The cell tower forwards the signal to the carrier’s core infrastructure, from where it is routed to the broader internet.

A typical LTE modem contains the following core components:

- **Cellular radio chip:** Responsible for encoding and decoding digital data into analog RF signals that comply with LTE protocols.
- **Antenna:** Radiates and captures electromagnetic waves during communication with cell towers.
- **SIM card:** Stores subscriber identity information required for authentication and registration with the mobile network.

The overall data flow is illustrated in Figure [A.1](#):

A.1 MODEM BOOT AND NETWORK ACCESS WORKFLOW

Upon boot, the modem’s internal firmware performs the following sequence:

- Powers up the SIM card interface.
- Reads SIM data (IMSI, operator information, PIN status).
- Enters a “ready” state if the SIM is valid and unlocked.

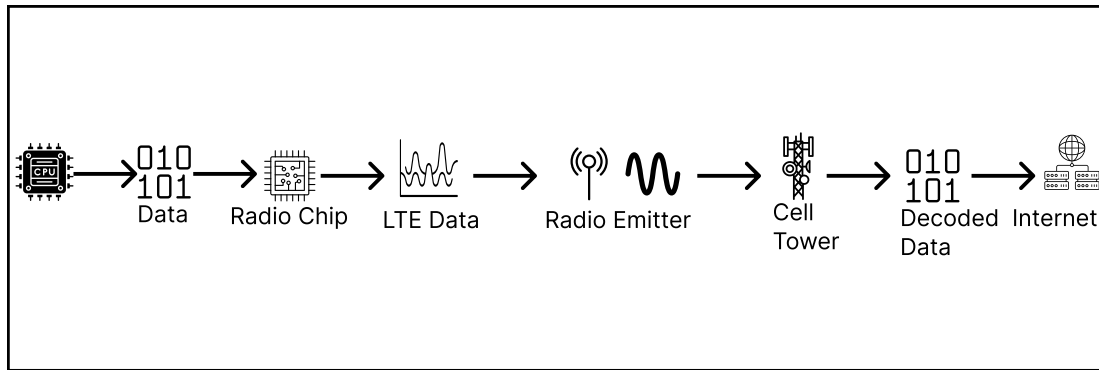


Figure A.1: Schematic representation of data flow from the SBC to the internet. The CPU generates a digital bitstream, which is modulated by the cellular radio chip into an analog signal. This signal is transmitted via the antenna, received by the cell tower, and routed through the carrier’s infrastructure to reach the internet.

The modem then initiates a network scan by:

- Determining home network identity:
 - **MCC:** Mobile Country Code
 - **MNC:** Mobile Network Code
- Selecting allowed **Radio Access Technologies (RATs)**, e.g., LTE, 3G.
- Listening for nearby tower broadcast signals (synchronization, beaconing).
- Measuring **received signal strength (RSSI)**.

When a suitable tower is found, the modem initiates a network registration request. If successful, the modem becomes “attached” to the cellular network, meaning:

- The SIM is authenticated.
- The modem is registered with the mobile network.
- It is authorized to begin a data session.

At this stage, the modem is connected to the mobile network. However, the system (SBC) still lacks the configuration needed to use the modem for Internet access. Because even though the modem is registered and authorized to communicate with the mobile network:

- No IP address has been assigned to the SBC.
- No data bearer (PDP context or default bearer) is set up yet.

- The SBC network system does not yet see the modem as a valid interface for routing Internet traffic.

What is still missing? For complete Internet connectivity, three additional steps must occur:

- **IP address assignment:** Needed for packet-level communication over the Internet.
- **Data session activation:** Also known as default bearer setup (in LTE), this creates a virtual tunnel (PDP context) between the device and the carrier’s core network.
- **Routing through the carrier gateway:** Data must be routed through the Packet Gateway (PGW) to reach external IP networks.

Enabling Internet access The modem module [37] supports two methods to initiate a data session and obtain an IP address for the SBC:

- **PPP (Point-to-Point Protocol):** A legacy method that treats the modem like a dial-up device. The SBC initiates a serial-based dial session over a dedicated port using AT commands (`ATD*99#`), after which a PPP daemon negotiates IP settings.
- **QMI (Qualcomm MSM Interface):** A modern binary protocol developed by Qualcomm. It uses a control channel (e.g., `/dev/cdc-wdm0`) to manage the modem via QMI commands and creates a virtual network interface (e.g., `wan0`) for efficient LTE data transmission.

In this implementation, QMI is preferred and used, as it supports faster session setup, lower overhead, and better integration with LTE features such as **Quality of Service (QoS)**. Once the QMI session is established, the mobile network assigns an IP address to the SBC as part of the data session (default bearer) setup. The modem acts as a bridge or pass-through, forwarding IP traffic between the SBC and the network.

Database structure

MySQL terminal session

Relational structure of the local MySQL database

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| on_site_monitoring_system_data |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> use on_site_monitoring_system_data;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_on_site_monitoring_system_data |
+-----+
| ImageData |
| RemoteMonitoringData |
| TaxonomicData |
+-----+
3 rows in set (0.00 sec)
```

(a) Database selection and table overview.

```
mysql> describe ImageData;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| image_id | int | NO | PRI | NULL | auto_increment |
| image_blob | longblob | NO | | NULL | |
+-----+
2 rows in set (0.00 sec)

mysql> describe RemoteMonitoringData;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| measurement_time | datetime | NO | | NULL | |
| location | varchar(255) | NO | | NULL | |
| temperature_k | float | NO | | NULL | |
| humidity_pct | float | NO | | NULL | |
| cct_k | float | NO | | NULL | |
| image_id | int | NO | PRI | NULL | |
+-----+
6 rows in set (0.01 sec)

mysql> describe TaxonomicData;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| image_id | int | NO | PRI | NULL | |
| openai_response | text | YES | | NULL | |
| deepseek_response | text | YES | | NULL | |
| gemini_response | text | YES | | NULL | |
| obsidentify_response | text | YES | | NULL | |
| biologist_review | text | YES | | NULL | |
+-----+
```

(b) Schema description for each table.

Figure A.1: MySQL terminal session showing schema structure of the project database.

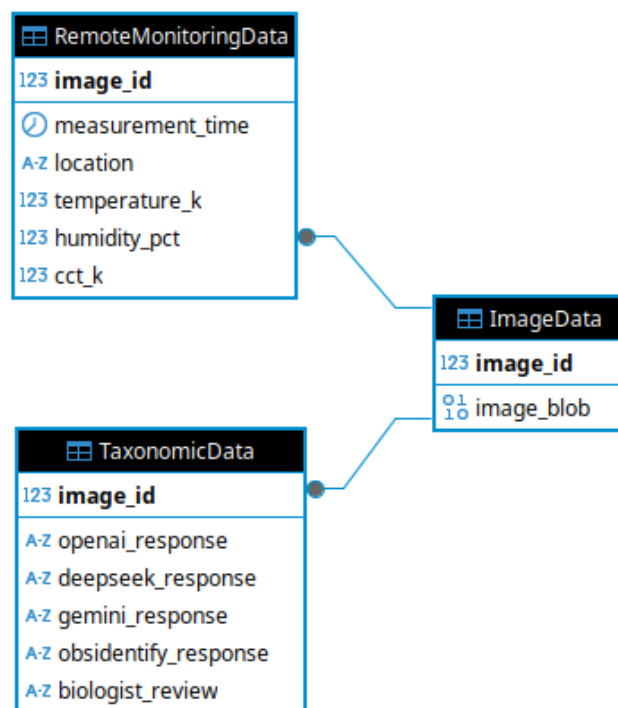


Figure A.2: Relational structure of the local MySQL database.

Listing B.1: Allrun shell script to automate processes

```

1 #!/bin/bash
2 set -e # Exit on any error
3
4 #####
5 # 1) Activate Python venv
6 #####
7 VENV_PATH="/hpmr/zakaria/Desktop/main/env_sensor_data"
8 if [ -d "$VENV_PATH" ]; then
9     echo "Activating Python virtual environment at $VENV_PATH"
10    source "$VENV_PATH/bin/activate"
11 fi
12
13 #####
14 # 2) Create a shared timestamp & folder
15 #####
16 TIMESTAMP=$(date +%Y-%m-%d_%H-%M)
17 mkdir "$TIMESTAMP"
18
19 #####
20 # 3) Start Data Logging in Background
21 #####
22 echo "Starting data logging in the background..."
23 python3 sensor_log.py "$TIMESTAMP" &
24 DATA_PID=$!
25
26 # Optional: small delay to ensure logger is ready before starting
    video
27 sleep 1
28
29 #####
30 # 4) Capture the Video
31 #####
32 echo "Capturing video..."
33 ./video_capture.sh "$TIMESTAMP"
34
35 #####
36 # 5) Stop Data Logging (once video capture is done)
37 #####
38 echo "Stopping data logger (PID: $DATA_PID)..."

```

```
39 kill "$DATA_PID" 2>/dev/null || true
40
41 # (Optional) wait a moment for the logger to flush data
42 sleep 1
43
44 #####
45 # 6) Transfer the Data
46 #####
47 echo "Transferring files..."
48 if ./send_data.sh "$TIMESTAMP"; then
49     echo "Transfer succeeded. Removing local copies..."
50     rm -r "$TIMESTAMP"           # Remove the folder with
        sensor_log.txt and video
51     rm "${TIMESTAMP}.tar.gz"    # Remove the tar archive
52 else
53     echo "Transfer failed! Keeping local files for debugging."
54 fi
55
56 echo "All steps completed successfully!"
```

Listing B.2: Arduino code for AHT20 sensor data acquisition

```
1 #include <Adafruit_AHTX0.h>
2
3 // Create AHT20 object
4 Adafruit_AHTX0 aht;
5
6 // Constants and variables initialization
7 // 1-second interval for printing
8 unsigned long previousMillis = 0;
9 const unsigned int intervalMillis = 1000;
10
11 void setup()
12 {
13   Serial.begin(115200);
14
15   if (!aht.begin())
16   {
17     Serial.println("Could not find AHT sensor! Check wiring.");
18     while (1) delay(10);
19   }
20   Serial.println("AHT10 or AHT20 sensor found.");
21 }
22
23 void loop()
24 {
25   unsigned long currentTime = millis(); // Declare and assign
      currentTime correctly
26   if (currentTime - previousMillis >= intervalMillis)
27   {
28     previousMillis = currentTime; // Update the previousMillis to
      avoid continuous triggering
29
30     sensors_event_t humidity, temp;
31     aht.getEvent(&humidity, &temp); // Populate temp and humidity
      objects with fresh data
32
33     Serial.print("Temperature: ");
34     Serial.print(temp.temperature);
35     Serial.println(" degC");
36
37     Serial.print("Humidity: ");
38     Serial.print(humidity.relative_humidity);
39     Serial.println("% rH");
40   }
41 }
```

Listing B.3: Shell script for image data acquisition

```

1 #!/bin/bash
2 set -e # Exit on error
3
4 if [ -z "$1" ]; then
5     echo "Error: No timestamp folder provided!"
6     echo "Usage: ./video_capture.sh <timestamp_folder>"
7     exit 1
8 fi
9
10 TIMESTAMP="$1"
11
12 # We assume $TIMESTAMP folder already exists (created by allrun.
    sh)
13 # If needed, check or create again:
14 # mkdir -p "$TIMESTAMP"
15
16 # Capture a 1-minute video at 1 fps and resolution 1640x1232 (4:3
    binned mode)
17 DURATION=60000 # 60 seconds in milliseconds
18 RES_WIDTH=1640
19 RES_HEIGHT=1232
20 FPS=1
21
22 RAW_FILE="${TIMESTAMP}/raw_video.h264"
23 FINAL_FILE="${TIMESTAMP}/${TIMESTAMP}.mp4"
24
25 echo "Capturing ${DURATION}ms video at ${FPS} fps, resolution ${
    RES_WIDTH}x${RES_HEIGHT}..."
26 libcamera-vid \
27 -t "${DURATION}" \
28 --width "${RES_WIDTH}" \
29 --height "${RES_HEIGHT}" \
30 --framerate "${FPS}" \
31 --codec h264 \
32 --inline yes \
33 --flush \
34 -o "${RAW_FILE}"
35
36 echo "Converting ${RAW_FILE} to ${FINAL_FILE}..."
37 ffmpeg \
38 -framerate "${FPS}" \
39 -fflags +genpts \
40 -i "${RAW_FILE}" \
41 -c:v copy \
42 -movflags +faststart \
43 -vsync cfr \
44 -hide_banner \
45 -loglevel error \
46 "${FINAL_FILE}"
47

```

```
48 if [ -f "${FINAL_FILE}" ]; then
49   rm "${RAW_FILE}"
50   echo "Removed temporary raw file: ${RAW_FILE}"
51 fi
52
53 echo "Video successfully created: ${FINAL_FILE}"
```

Listing B.4: Python script for geolocation

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """
5 get_gps_location.py
6 - Fetches current GPS coordinates from SIM7600
7 and saves them to geolocation.txt as a single line: <latitude>,<
      longitude>,<region>
8
9 This script must be run with sudo privileges due to ModemManager
      interaction.
10 """
11
12 import serial, subprocess, time, os
13
14 GPS_PORT = "/dev/ttyUSB3" # Update if needed
15 BAUDRATE = 115200
16 OUTPUT_FILE = "geolocation.txt"
17
18 def convert_gps_coord(coord_str, direction):
19     """Convert GPS NMEA ddm.mmmm format to decimal degrees with
      direction"""
20     if not coord_str:
21         return ""
22
23     deg_len = 2 if direction in ("N", "S") else 3
24     deg = float(coord_str[:deg_len])
25     minutes = float(coord_str[deg_len:])
26     decimal = deg + minutes / 60
27     return f"{decimal:.6f}{direction}"
28
29
30 def main():
31     try:
32         subprocess.run(["sudo", "systemctl", "stop", "
      ModemManager"], check=True)
33         print("[gps] ModemManager stopped.")
34
35         with serial.Serial(GPS_PORT, BAUDRATE, timeout=2) as ser:
36             def send_at(cmd, delay=1):
37                 ser.write((cmd + "\r\n").encode())
38                 time.sleep(delay)
39                 return ser.read_all().decode(errors="ignore")
40
41             send_at("AT+CGPS=1,1", delay=2)
42             print("[gps] GPS activated, waiting for fix...")
43
44             for _ in range(15): # Wait up to ~60s
45                 res = send_at("AT+CGPSINFO", delay=3)
46                 if "+CGPSINFO:" in res:

```

```
47         line = res.split("+CGPSINFO:")[1].splitlines
         () [0].strip()
48         parts = line.split(",")
49         if len(parts) >= 4 and parts[0] and parts[2]:
50             lat = convert_gps_coord(parts[0], parts
         [1])
51             lon = convert_gps_coord(parts[2], parts
         [3])
52             with open(OUTPUT_FILE, "w") as f:
53                 f.write(f"{lat},{lon}\n")
54                 print(f"[gps] Written to {OUTPUT_FILE}: {
         lat}, {lon}")
55             return
56             print("[gps] Timeout: no fix.")
57     except Exception as e:
58         print(f"[gps] Error: {e}")
59     finally:
60         subprocess.run(["sudo", "systemctl", "start", "
         ModemManager"], check=False)
61         print("[gps] ModemManager restarted.")
62
63 if __name__ == "__main__":
64     main()
```

Listing B.5: Python script for serial data logging

```

1 import os, sys, datetime, re, serial, requests
2
3 OPENCAGE_KEY = "KEY"
4
5 # --- ZoneInfo import that works on any Python >3.6 ---
6 try:
7     from zoneinfo import ZoneInfo          # Py 3.9+
8 except ImportError:
9     from backports.zoneinfo import ZoneInfo # pip install
        backports.zoneinfo
10
11 SERIAL_PORT = "/dev/serial/by-id/usb-Prolific_Technology_Inc._USB
        -Serial_Controller-if00-port0"
12 BAUD_RATE   = 115200
13
14 TEMP_RE = re.compile(r"Temperature:\s*([-]?\d+\.\d+)")
15 HUM_RE  = re.compile(r"Humidity:\s*([-]?\d+\.\d+)")
16
17 def geo_lookup():
18     """Reads lat/lon from geolocation.txt and gets the timezone
        from OpenCage"""
19     try:
20         with open("geolocation.txt", "r") as f:
21             lat, lon = f.readline().strip().split(",")
22
23         # Remove direction letters for API query
24         lat_clean = lat.rstrip("NS")
25         lon_clean = lon.rstrip("EW")
26
27         url = "https://api.opencagedata.com/geocode/v1/json"
28         params = {
29             "key": OPENCAGE_KEY,
30             "q": f"{lat_clean},{lon_clean}",
31             "no_annotations": 0
32         }
33         res = requests.get(url, params=params, timeout=5)
34         data = res.json()
35
36         timezone = data["results"][0]["annotations"]["timezone"][
        "name"] # e.g., Europe/Brussels
37         region = data["results"][0]["components"].get("state", "
        unknown")
38
39         return lat, lon, region, timezone
40     except Exception as e:
41         print(f"[geo] lookup failed: {e}")
42         return "", "", "", None
43
44 def main():
45     # ---- destination folder ----

```

```

46     folder = sys.argv[1] if len(sys.argv) > 1 else "."
47     os.makedirs(folder, exist_ok=True)
48
49     # ---- one time location lookup ----
50     lat, lon, _, tzname = geo_lookup() # underscore discards the
    'region'
51     tz = ZoneInfo(tzname) if tzname else datetime.timezone.utc
52     print(f"[geo] lat={lat} lon={lon} tz={tzname}")
53
54     # ---- open serial & log file ----
55     ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
56     log_path = os.path.join(folder, "sensor_log.txt")
57     print(f"[log] writing to {log_path}")
58
59     last_temp = last_hum = None
60
61     with open(log_path, "w") as f: #
    overwrite each run
62         f.write("time_iso,lat,lon,temperature_C,humidity_rH\n")
63
64         try:
65             while True:
66                 raw = ser.readline().decode("utf-8", "replace").
strip()
67
68                 if not raw:
69                     continue
70
71                 # read whichever sensor value arrived
72                 m = TEMP_RE.search(raw)
73                 if m:
74                     last_temp = m.group(1)
75
76                 m = HUM_RE.search(raw)
77                 if m:
78                     last_hum = m.group(1)
79
80                 # write a row only when we have both numbers
81                 if last_temp is not None and last_hum is not None
:
82                     now_iso = datetime.datetime.now(tz).isoformat
(timespec="seconds")
83                     line = f"{now_iso},{lat},{lon},{last_temp},{
last_hum}\n"
84                     f.write(line)
85                     f.flush()
86                     print(line, end="")
87
88                 last_temp = last_hum = None
89             except KeyboardInterrupt:
90                 print("\n[log] stopped by user.")
finally:

```

```
91         ser.close()
92
93 if __name__ == "__main__":
94     main()
```

Listing B.6: Shell script for data transmission

```
1 #!/bin/bash
2 set -e # Exit on error
3
4 # --- Configuration ---
5 USER="zakaria-ben-mekki"
6 HOST="zakariabenmekki"
7 REMOTE_PATH= <PLACEHOLDER>
8
9 # We expect the timestamp folder name as $1
10 if [ -z "$1" ]; then
11     echo "Error: No timestamp folder specified!"
12     echo "Usage: ./send_data.sh <timestamp_folder>"
13     exit 1
14 fi
15
16 FOLDER_NAME="$1"
17 ARCHIVE_NAME="${FOLDER_NAME}.tar.gz"
18
19 # Make sure the folder exists
20 if [ ! -d "$FOLDER_NAME" ]; then
21     echo "Error: Folder $FOLDER_NAME does not exist!"
22     exit 1
23 fi
24
25 # --- Create a tar.gz of the entire folder ---
26 echo "Creating archive $ARCHIVE_NAME from folder $FOLDER_NAME ..."
27 tar -czvf "$ARCHIVE_NAME" "$FOLDER_NAME" || {
28     echo "Error: Failed to create archive."
29     exit 1
30 }
31
32 # --- Transfer to remote server (using rsync or scp) ---
33 echo "Sending $ARCHIVE_NAME to $USER@$HOST:$REMOTE_PATH ..."
34 rsync -avz "$ARCHIVE_NAME" "$USER@$HOST:$REMOTE_PATH" || {
35     echo "Error: Failed to transfer files."
36     exit 1
37 }
38
39 # (Optional) remove local archive if desired
40 # rm "$ARCHIVE_NAME"
41
42 echo "Transfer complete!"
```

Listing B.7: Python script for database logging

```

1 import os, time, mysql.connector, csv
2 from watchdog.observers import Observer
3 from watchdog.events import FileSystemEventHandler
4 import csv, datetime
5
6 #
7
8 -----
9
10 config ----
11 PROCESSED_DIR = "/home/zakaria-ben-mekki/Desktop/master_thesis/
12   central_unit/data/processed/"
13 SUPPORTED_EXT = ('.jpg', '.jpeg', '.png', '.bmp', '.tif', '.tiff'
14   )
15
16 DB_CONFIG = dict(
17     host      = "localhost",
18     user      = confidential,
19     password  = confidential,
20     database  = "on_site_monitoring_system_data"
21 )
22 #
23
24 -----
25
26 helpers ---
27
28 def wait_for_folder_stabilization(folder, dt=2, stable=5):
29     last = set(os.listdir(folder)); since = time.time()
30     while True:
31         time.sleep(dt)
32         cur = set(os.listdir(folder))
33         if cur != last:
34             last, since = cur, time.time()
35         elif time.time() - since >= stable:
36             break
37     print(f"Folder {folder} is stable ({len(cur)} files)")
38
39 def parse_sensor_log(txt_path):
40     """Return list[dict] with keys time, location, temp, hum."""
41     rows = []
42     with open(txt_path, newline="") as f:
43         reader = csv.reader(f)
44         for lineno, line in enumerate(reader, start=1):
45             if not line or len(line) < 6:
46                 continue # blank /
47             incomplete
48             # unpack & strip
49             time_iso, lat, lon, temp, hum, *rest = [c.strip() for
50 c in line + ["", ""]]
51
52             # skip header or non-numeric cells
53             if lineno == 1 and temp.lower().startswith("
54 temperature"):

```

```

42         continue
43     try:
44         t_val = float(temp) if temp else None
45         h_val = float(hum) if hum else None
46     except ValueError:
47         print(f"[parse] skip line {lineno}: cannot
convert '{temp}', '{hum}'")
48         continue
49
50         rows.append(dict(
51             time=time_iso,                                # ISO 8601
string
52             location=f"{lat},{lon}", # single text field
53             temp=t_val,
54             hum=h_val
55         ))
56     return rows
57
58 def insert_image_into_db(img_path):
59     conn = mysql.connector.connect(**DB_CONFIG)
60     cur = conn.cursor()
61     with open(img_path, 'rb') as f:
62         cur.execute("INSERT INTO ImageData (image_blob) VALUES (%
s)", (f.read(),))
63     conn.commit()
64     img_id = cur.lastrowid
65     cur.close(); conn.close()
66     return img_id
67
68 def insert_sensor_row(image_id, row):
69     conn = mysql.connector.connect(**DB_CONFIG)
70     cur = conn.cursor()
71     sql = """
72         INSERT INTO RemoteMonitoringData
73         (measurement_time, location, temperature_k, humidity_pct,
cct_k, image_id)
74         VALUES (%s, %s, %s, %s, 0.0, %s)
75     """
76     temp_k = row['temp'] + 273.15 if row['temp'] is not None else
None
77     cur.execute(sql, (row['time'], row['location'], temp_k, row['
hum'], image_id))
78     conn.commit()
79     cur.close(); conn.close()
80
81 # -----
watchdog -----
82
83 class NewFolderHandler(FileSystemEventHandler):
84     def __init__(self):
85         self.known = set(os.listdir(PROCESSED_DIR))

```

```

86
87     def _process(self, folder):
88         print(f"\n New folder: {folder}")
89         wait_for_folder_stabilization(folder)
90
91         # 1) sensor log
92         txt_files = [f for f in os.listdir(folder) if f.lower().
endswith('.txt')]
93         if not txt_files:
94             print("No sensor log found")
95             return
96         sensor_rows = parse_sensor_log(os.path.join(folder,
txt_files[0]))
97         if not sensor_rows:
98             print("Sensor log empty or malformed")
99             return
100
101         # 2) images
102         images = sorted(f for f in os.listdir(folder) if f.lower
().endswith(SUPPORTED_EXT))
103         pairs = min(len(images), len(sensor_rows))
104         print(f"Pairing {pairs} images with sensor rows")
105
106         for i in range(pairs):
107             img_path = os.path.join(folder, images[i])
108             row      = sensor_rows[i]
109
110             img_id = insert_image_into_db(img_path)
111             insert_sensor_row(img_id, row)
112
113             print(f"image_id {img_id} {row['time']} T={row['
temp']}C H={row['hum']}%")
114
115         # watchdog callbacks
116         def on_created(self, event):
117             if event.is_directory:
118                 name = os.path.basename(event.src_path)
119                 if name not in self.known:
120                     self.known.add(name)
121                     self._process(event.src_path)
122
123         def on_moved(self, event):
124             if event.is_directory:
125                 name = os.path.basename(event.dest_path)
126                 if name not in self.known:
127                     self.known.add(name)
128                     self._process(event.dest_path)
129
130 # -----
main -----
131

```

```
132 if __name__ == "__main__":
133     obs = Observer()
134     obs.schedule(NewFolderHandler(), PROCESSED_DIR, recursive=
False)
135     obs.start()
136     print(f"Watching {PROCESSED_DIR} ... Ctrl-C to stop.")
137     try:
138         while True:
139             time.sleep(1)
140     except KeyboardInterrupt:
141         obs.stop()
142     obs.join()
```

Listing B.8: Python script for extraction and preprocessing

```

1 import os
2 import cv2
3 import shutil
4 import time
5 import tarfile
6 from watchdog.observers import Observer
7 from watchdog.events import FileSystemEventHandler
8
9 def factorize_video(source_folder, destination_base):
10     # 1. Extract the folder name from the provided source path
11     folder_name = os.path.basename(os.path.normpath(source_folder
12     ))
13
14     # 2. Create the destination folder inside the given
15     # destination base path
16     destination_folder = os.path.join(destination_base,
17     folder_name)
18     os.makedirs(destination_folder, exist_ok=True)
19     print(f"Destination folder created: {destination_folder}")
20
21     # 3. Find the first .mp4 file inside the source folder
22     video_files = [f for f in os.listdir(source_folder) if f.
23     lower().endswith('.mp4')]
24     if not video_files:
25         # If no .mp4 file is found, exit the function
26         print("No MP4 video found in the source folder.")
27         return
28
29     # Full path to the video file
30     video_path = os.path.join(source_folder, video_files[0])
31     print(f"Reading video: {video_path}")
32
33     # 4. Read the video and extract frames one by one
34     cap = cv2.VideoCapture(video_path) # Open the video file
35     frame_id = 0 # Initialize frame counter
36
37     while True:
38         ret, frame = cap.read() # Read the next frame
39         if not ret:
40             # No more frames to read; exit the loop
41             break
42
43         # Define the filename for the current frame
44         frame_filename = os.path.join(destination_folder, f"{
45         frame_id}.jpg")
46         # Save the current frame as a .jpg image
47         cv2.imwrite(frame_filename, frame)
48         frame_id += 1 # Increment the frame counter
49
50     cap.release() # Release the video capture object

```

```

46     print(f"Extraction completed: {frame_id} frames saved.")
47
48     # 5. Find and copy the first .txt file inside the source
folder
49     txt_files = [f for f in os.listdir(source_folder) if f.lower
().endswith('.txt')]
50     if txt_files:
51         txt_source_path = os.path.join(source_folder, txt_files
[0])
52         txt_destination_path = os.path.join(destination_folder,
txt_files[0])
53         shutil.copyfile(txt_source_path, txt_destination_path)
54         print(f"Text file copied: {txt_files[0]}")
55     else:
56         print("No TXT file found in the source folder.")
57
58 def extract_tar_gz(tar_path, extract_to):
59     # Extract a .tar.gz file to the specified directory
60     if tarfile.is_tarfile(tar_path):
61         with tarfile.open(tar_path, "r:gz") as tar:
62             tar.extractall(path=extract_to)
63             print(f"Extracted {tar_path} to {extract_to}")
64     else:
65         print(f"File {tar_path} is not a valid tar.gz archive.")
66
67 class NewFolderHandler(FileSystemEventHandler):
68     def __init__(self, destination_base, raw_base_folder):
69         self.destination_base = destination_base
70         self.raw_base_folder = raw_base_folder
71
72     def handle_event(self, event_path, is_directory):
73         if is_directory:
74             print(f"New or moved folder detected: {event_path}")
75             time.sleep(2) # Optional: wait a little in case the
folder is still being populated
76             factorize_video(event_path, self.destination_base)
77         elif event_path.endswith(".tar.gz"):
78             print(f"New tar.gz file detected: {event_path}")
79             time.sleep(2)
80             extract_tar_gz(event_path, self.raw_base_folder)
81             os.remove(event_path)
82
83     def on_created(self, event):
84         self.handle_event(event.src_path, event.is_directory)
85
86     def on_moved(self, event):
87         self.handle_event(event.dest_path, event.is_directory)
88
89
90 if __name__ == "__main__":
91     # Define your source parent folder (where new folders or tar.

```

```
gz files will appear)
92 raw_base_folder = "/home/zakaria-ben-mekki/Desktop/
master_thesis/central_unit/data/raw"
93 destination_base = "/home/zakaria-ben-mekki/Desktop/
master_thesis/central_unit/data/processed"
94
95 # Set up the watchdog observer
96 event_handler = NewFolderHandler(destination_base,
raw_base_folder)
97 observer = Observer()
98 observer.schedule(event_handler, path=raw_base_folder,
recursive=False)
99 observer.start()
100
101 print(f"Watching for new folders or tar.gz files in: {
raw_base_folder}")
102 try:
103     while True:
104         time.sleep(1)
105 except KeyboardInterrupt:
106     observer.stop()
107 observer.join()
```

Listing B.9: Python script for fetching images and performing insect recognition

```

1 import os
2 import base64
3 import json
4 import mysql.connector
5 import requests
6 import PIL.Image
7 from openai import OpenAI
8
9 # DB config
10 DB_CONFIG = {
11     'host': 'localhost',
12     'user': confidential,
13     'password': confidential,
14     'database': 'on_site_monitoring_system_data'
15 }
16
17 # API keys
18 OPENAI_API_KEY = confidential
19 GEMINI_API_KEY = confidential
20
21 def fetch_image(image_id, output_path):
22     """Fetches an image blob from DB and saves it to a file."""
23     connection = mysql.connector.connect(**DB_CONFIG)
24     cursor = connection.cursor()
25
26     cursor.execute("SELECT image_blob FROM ImageData WHERE
27 image_id = %s", (image_id,))
28     result = cursor.fetchone()
29
30     if result:
31         with open(output_path, 'wb') as f:
32             f.write(result[0])
33         print(f"Image {image_id} saved to {output_path}")
34     else:
35         print(f"No image found for image_id {image_id}")
36
37     cursor.close()
38     connection.close()
39
40 def fetch_location(image_id):
41     """Retrieves location string from RemoteMonitoringData and
42 returns [lat, lon] floats."""
43     conn = mysql.connector.connect(**DB_CONFIG)
44     cur = conn.cursor()
45
46     cur.execute(
47         "SELECT location FROM RemoteMonitoringData WHERE image_id
48 = %s", (image_id,))
49     result = cur.fetchone()

```

```

48     cur.close()
49     conn.close()
50
51     if not result:
52         raise ValueError(f"No location found for image_id {
image_id}")
53
54     loc_str = result[0] # e.g., "50.854642N,4.326576E"
55     # Parse latitude and longitude
56     match = re.match(r"([0-9.]+)([NS]),?\s*([0-9.]+)([EW])",
loc_str)
57     if not match:
58         raise ValueError(f"Invalid location format: {loc_str}")
59
60     lat, lat_dir, lon, lon_dir = match.groups()
61     lat = float(lat) * (1 if lat_dir == 'N' else -1)
62     lon = float(lon) * (1 if lon_dir == 'E' else -1)
63     return [lat, lon]
64
65 def analyze_with_openai(image_path):
66     """Sends the image to OpenAI Vision and retrieves the
classification response."""
67     client = OpenAI(api_key=OPENAI_API_KEY)
68     base64_image = encode_image(image_path)
69
70     response = client.chat.completions.create(
71         model="gpt-4o-mini",
72         messages=[
73             {"role": "user", "content": [
74                 {"type": "text", "text": (
75                     "Given the output of an insect recognition
model, list each identified species."
76                     "Extract the scientific name and, beside it,
the full taxonomic hierarchy"
77                     "from order down to species (Order- Family-
Genus-Species)."}
78                 )}],
79                 {"type": "image_url", "image_url": {"url": f"data
:image/jpeg;base64,{base64_image}"}}},
80             ]}
81     ],
82     )
83     return response.choices[0].message.content
84
85 def analyze_with_gemini(image_path):
86     """Sends the image to Gemini and retrieves the classification
response."""
87     client = genai.Client(api_key=GEMINI_API_KEY)
88     img = PIL.Image.open(image_path)
89
90     prompt = (

```

```

91         "Given the output of an insect recognition model, list
each identified species."
92         "Extract the scientific name and, beside it, the full
taxonomic hierarchy"
93         "from order down to species (Order- Family-Genus-Species)
."
94     )
95
96     response = client.models.generate_content(
97         model="gemini-2.0-flash-exp",
98         contents=[prompt, img]
99     )
100     return response.text
101
102 def analyze_with_obsidentify(image_path, location):
103     """Sends the image to ObsIdentify API and retrieves the
classification response."""
104     endPoint = "https://multi-source.identify.
biodiversityanalysis.eu/v2/observation/identify"
105     lat, lon = location
106
107     myImage = {'image': open(image_path, 'rb')}
108     response = requests.post(url=endPoint, files=myImage)
109
110     return json.dumps(response.json(), indent=2)
111
112 def insert_taxonomic_data(image_id, openai_response,
obsidentify_response):
113     connection = mysql.connector.connect(**DB_CONFIG)
114     cursor = connection.cursor()
115
116     sql_query = """
117         INSERT INTO TaxonomicData
118         (image_id, openai_response, deepseek_response,
gemini_response, obsidentify_response, biologist_review)
119         VALUES (%s, %s, NULL, %s, %s, NULL)
120     """
121     cursor.execute(sql_query, (image_id, openai_response,
obsidentify_response))
122     connection.commit()
123
124     cursor.close()
125     connection.close()
126     print(f"Taxonomic data inserted for image_id {image_id}")
127
128
129 def encode_image(image_path):
130     """Encodes an image to Base64 string."""
131     with open(image_path, "rb") as img_file:
132         return base64.b64encode(img_file.read()).decode('utf-8')
133

```

```
134 if __name__ == "__main__":
135     # Choose an existing image_id
136     image_id =
137     temp_path = "./temp_image.jpg"
138
139     # Step 1: fetch image blob
140     fetch_image(image_id, temp_path)
141
142     # Step 2: retrieve and parse location
143     location = fetch_location(image_id)
144     print(f"Using location: {location}")
145
146     # Step 3: analyze with different APIs
147     openai_resp = analyze_with_openai(temp_path)
148     gemini_resp = analyze_with_gemini(temp_path)
149     obs_resp = analyze_with_obsidentify(temp_path)
150
151     # Step 4: store responses
152     insert_taxonomic_data(image_id, openai_resp, gemini_resp,
153                           obs_resp)
154
155     # Cleanup temp file
156     os.remove(temp_path)
```

Bibliography

- [1] Adafruit Industries. *AHT20 Temperature & Humidity Sensor Breakout Board*. <https://www.adafruit.com/product/4566>. Accessed: 2025-05-19. 2023.
- [2] Arduino. *Arduino Uno Rev3 Datasheet*. <https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>. Accessed: 2025-05-19. 2025.
- [3] J. Bolliger, J. Haller, B. Wermelinger, S. Blum, and M. Obrist. “Contrasting effects of street light shapes and LED color temperatures on nocturnal insects and bats”. In: *Basic and Applied Ecology* (2022).
- [4] J. Bolliger, T. Hennet, B. Wermelinger, S. Blum, J. Haller, and M. Obrist. “Low impact of two LED colors on nocturnal insect abundance and bat activity in a peri-urban environment”. In: *Journal of Insect Conservation* (2020).
- [5] J. Bolliger, T. Hennet, B. Wermelinger, R. Bösch, R. Pazur, S. Blum, J. Haller, and M. Obrist. “Effects of traffic-regulated street lighting on nocturnal insect abundance and bat activity”. In: *Basic and Applied Ecology* (2020).
- [6] D. Boyes, D. Evans, R. Fox, M. Parsons, and M. Pocock. “Street lighting has detrimental impacts on local insect populations”. In: *SCIENCE ADVANCES* 7.35 (Aug. 2021). ISSN: 2375-2548. DOI: [10.1126/sciadv.abi8322](https://doi.org/10.1126/sciadv.abi8322).
- [7] L. De Causmaecker, A. Mentens, S. Aerts, B. Vanschoenwinkel, R. Moortgat, P. Van den Bossche, and V. A. Jacobs. “DEVELOPMENT OF A CONTROL SYSTEM TO EVALUATE THE IMPACT OF ARTIFICIAL LIGHT MODULATION ON INSECTS”. en. In: *Proceedings of the CIE Symposium on Advances on the Measurement of Temporal Light Modulation*. International Commission on Illumination (CIE), Oct. 2022. ISBN: 978-3-902842-70-1. DOI: [10.25039/x49.2022.P07](https://doi.org/10.25039/x49.2022.P07).
- [8] L. De Causmaecker, A. Mentens, L. Segers, P. Van den Bossche, B. Vanschoenwinkel, and V. A. Jacobs. “TOWARDS PUBLIC LED LIGHTING WITH MINIMAL IMPACT ON INSECT MOVEMENT”. en. In: *Proceedings of the 30th Session of the CIE, Ljubljana, Slovenia, September 15 –*

- 23, 2023, Volume 1. Ljubljana, Slovenia (hosted by NC Slovenia): International Commission on Illumination, CIE, Dec. 2023, pp. 291–300. ISBN: 978-3-902842-77-0. DOI: [10.25039/x50.2023.OP044](https://doi.org/10.25039/x50.2023.OP044).
- [9] J. Deichmann, C. Gatty, J. Navarro, A. Alonso, R. Linares-Palomino, and T. Longcore. “Reducing the blue spectrum of artificial light at night minimises insect attraction in a tropical lowland forest”. In: *INSECT CONSERVATION AND DIVERSITY* 14.2 (Mar. 2021), pp. 247–259. ISSN: 1752-458X. DOI: [10.1111/icad.12479](https://doi.org/10.1111/icad.12479).
- [10] Gordon Lyon. *Nmap: Network Mapper*. <https://nmap.org/>. Accessed: 2025-05-23. 2024.
- [11] R. H. A. van Grunsven, J. Becker, S. Peter, S. Heller, and F. Hölker. “Long-Term Comparison of Attraction of Flying Insects to Streetlights after the Transition from Traditional Light Sources to Light-Emitting Diodes in Urban and Peri-Urban Settings”. In: *Sustainability* 11.22 (2019). ISSN: 2071-1050. DOI: [10.3390/su11226198](https://doi.org/10.3390/su11226198).
- [12] T. Igoe. *Making Things Talk: Practical Methods for Connecting Physical Objects 3rd Edition*. Maker Media, 2017. ISBN: 978-1-68045-215-0.
- [13] International Commission on Illumination (CIE). *CIE 015:2018 – Colorimetry*. Tech. rep. Tables 11, 12.1, 12.2. CIE Central Bureau, 2018.
- [14] International Commission on Illumination (CIE). *Relative spectral power distributions of illuminants representing typical LED lamps*. <https://cie.co.at/datatable/relative-spectral-power-distributions-illuminants-representing-typical-led-lamps>. Accessed July 31, 2025. 2025.
- [15] libcamera Project. *libcamera: Open Source Camera Stack and Framework*. <https://libcamera.org/>. Accessed: 2025-05-23. 2025.
- [16] C. Liechti. *pySerial: Python Serial Port Extension*. <https://github.com/pyserial/pyserial>. Accessed May 26, 2025. 2025.
- [17] M. C. F. Lima, M. E. D. de Almeida Leandro, C. Valero, L. C. P. Coronel, and C. O. G. Bazzo. “Automatic Detection and Monitoring of Insect Pests—A Review”. In: *Agriculture* (2020).
- [18] M. T. Lockett, T. M. Jones, M. A. Elgar, K. J. Gaston, M. E. Visser, and G. R. Hopkins. “Urban street lighting differentially affects community attributes of airborne and ground-dwelling invertebrate assemblages.” In: *Journal of Applied Ecology* 58.10 (Oct. 2021), pp. 2329–2339. ISSN: 00218901. DOI: [10.1111/1365-2664.13969](https://doi.org/10.1111/1365-2664.13969).
- [19] B. Luo, R. Xu, Y. Li, W. Zhou, W. Wang, H. Gao, Z. Wang, Y. Deng, Y. Liu, and J. Feng. “Artificial light reduces foraging opportunities in wild least horseshoe bats.” In: *Environmental Pollution* 288 (Nov. 2021), N.PAG–N.PAG. ISSN: 02697491. DOI: [10.1016/j.envpol.2021.117765](https://doi.org/10.1016/j.envpol.2021.117765).

- [20] E. J. McNaughton, J. R. Beggs, K. J. Gaston, D. N. Jones, and M. C. Stanley. “Retrofitting streetlights with LEDs has limited impacts on urban wildlife.” In: *Biological Conservation* 254 (Feb. 2021), N.PAG–N.PAG. ISSN: 00063207. DOI: [10.1016/j.biocon.2020.108944](https://doi.org/10.1016/j.biocon.2020.108944).
- [21] Mean Well Enterprises Co., Ltd. *HDR-30 Series Datasheet*. <https://docs.rs-online.com/050f/0900766b815d9418.pdf>. Accessed: 2025-05-19. 2024.
- [22] A. Méndez, L. Martín, J. Arines, R. Carballeira, and P. Sanmartín. “Attraction of Insects to Ornamental Lighting Used on Cultural Heritage Buildings: A Case Study in an Urban Area”. In: *Insects* (2022).
- [23] Minicom Team. *Minicom: Serial Communication Program*. <https://salsa.debian.org/minicom-team/minicom>. Accessed: 2025-05-23. 2018.
- [24] ModemManager Project. *ModemManager: Mobile Broadband Modem Management*. <https://modemmanager.org/>. Accessed: 2025-05-23. 2025.
- [25] NetworkManager Project. *NetworkManager: Network Configuration Tool Suite*. <https://www.networkmanager.dev/>. Accessed: 2025-05-23. 2025.
- [26] OpenSSH Developers. *OpenSSH: Portable Secure Shell*. <https://www.openssh.com/>. Accessed: 2025-05-23. 2024.
- [27] Oracle Corporation. *MySQL*. Accessed: 2025-06-12. Oracle. 2025.
- [28] A. C. S. Owens and S. M. Lewis. “The impact of artificial light at night on nocturnal insects: A review and synthesis”. en. In: *Ecology and Evolution* 8.22 (Nov. 2018), pp. 11337–11358. ISSN: 2045-7758, 2045-7758. DOI: [10.1002/ece3.4557](https://doi.org/10.1002/ece3.4557).
- [29] W. Perrin, M. Moretti, A. Vergnes, D. Borcard, and P. Jay-Robert. “Response of dung beetle assemblages to grazing intensity in two distinct bioclimatic contexts.” In: *Agriculture, Ecosystems & Environment* 289 (Feb. 2020), N.PAG–N.PAG. ISSN: 01678809. DOI: [10.1016/j.agee.2019.106740](https://doi.org/10.1016/j.agee.2019.106740).
- [30] PuTTY Contributors. *PuTTY: A Free Telnet/SSH Client*. <https://putty.org/>. Accessed: 2025-05-23. 2024.
- [31] Raspberry Pi Foundation. *Raspberry Pi 4 Model B Datasheet*. <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf>. Accessed: 2025-05-19. 2025.
- [32] Raspberry Pi Ltd. *Camera Module 3 NoIR (Wide Angle)*. <https://www.raspberrypi.com/products/camera-module-3/>. Accessed: 2025-05-19. 2023.
- [33] RealVNC Ltd. *RealVNC: Remote Access Software*. <https://www.realvnc.com/>. Accessed: 2025-05-23. 2024.

- [34] D. Russo, F. Cosentino, F. Festa, F. De Benedetta, B. Pejic, P. Cerretti, and L. Ancillotto. “Artificial illumination near rivers may alter bat-insect trophic interactions.” In: *Environmental Pollution* 252 (Sept. 2019), pp. 1671–1677. ISSN: 02697491. DOI: [10.1016/j.envpol.2019.06.105](https://doi.org/10.1016/j.envpol.2019.06.105).
- [35] B. Seymoure, T. Parrish, K. Egan, M. Furr, D. Irwin, C. Brown, M. Crump, J. White, K. Crooks, and L. Angeloni. “Better red than dead: Plasticine moths are attacked less under HPS streetlights than LEDs”. In: *BASIC AND APPLIED ECOLOGY* 74 (Feb. 2024), pp. 66–73. ISSN: 1439-1791. DOI: [10.1016/j.baae.2023.11.008](https://doi.org/10.1016/j.baae.2023.11.008).
- [36] Tailscale Inc. *Tailscale: Zero-Config VPN for Secure Networks*. <https://tailscale.com/>. Accessed: 2025-05-23. 2025.
- [37] WaveShare. *SIM7600G-H 4G HAT (B) – Dial-up Networking with Raspbian*. https://www.waveshare.com/wiki/SIM7600E-H_4G_HAT. Accessed: 2025-05-19. 2025.
- [38] N. Willmott, J. Henneken, M. Elgar, and T. Jones. “Guiding lights: Foraging responses of juvenile nocturnal orb-web spiders to the presence of artificial light at night”. In: *Ethology* (2019).