

GreenLangAI: A Conceptual AI-Assisted Framework for Generating Sustainability-Constrained Domain-Specific Languages from Natural Language

Amisha Ravindra Kanbargi¹, Dr. Asmita Deshpande²

¹(UG BCA Student) KLS Gogte College of Commerce BCA Department, Belagavi, Karnataka, India

²(Ph.D, MCA) International Student Conference on Artificial Intelligence, Sustainability and Green Finance

ABSTRACT

Artificial intelligence is increasingly being used to turn natural-language instructions into working software. However, most existing tools still generate code in general-purpose languages and usually consider sustainability only after the code has been produced. This study introduces **GreenLangAI**, a conceptual framework designed to create verified, sustainability-focused domain-specific languages (DSLs) for small sustainable-development applications. The aim is to make programming easier for people with limited coding knowledge while ensuring that the generated solutions are correct, safe, suitable for the target hardware, and efficient in their use of resources.

The study follows a conceptual design-and-evaluation approach supported by a focused review of research on natural-language program synthesis, grammar prompting, AI-assisted DSL development, code verification, green software engineering, low-code systems, and sustainable AI. Ideas from these areas were combined to design a modular framework that includes requirement clarification, minimal Backus–Naur Form grammar generation, grammar-constrained program creation, plain-language explanation, syntax and logic checking, safety validation, automatic repair, human approval, hardware-specific backend generation, and lifecycle sustainability assessment. A smart-classroom energy-management system is used as an illustrative case study. The proposed evaluation considers correctness, runtime, memory use, energy consumption, network activity, dependency size, usability, AI-generation overhead, environmental benefit, and operational break-even.

The review shows that previous studies often examine DSL generation, code verification, and energy-efficient programming separately. GreenLangAI brings these areas together and treats sustainability as part of the language design itself through local execution, limited sensing, reduced communication, sleep states, and protected safety rules. The study highlights that AI-generated software should not be called

sustainable simply because it is shorter or easier to read. Its sustainability must be demonstrated through testing on the target hardware, lifecycle measurement, and user-accessibility evaluation.

Keywords: GreenLangAI, Domain-Specific Languages, Sustainable Artificial Intelligence, Green Software Engineering, AI-Based Code Generation, Computational Sustainability, Hardware-Aware Code Generation, Sustainable Development

1. Introduction

1.1 Background

Sustainable development requires technological progress to support economic growth, social well-being and environmental protection at the same time. The United Nations 2030 Agenda presents sustainable development through 17 interconnected goals and stresses that progress should take place without harming nature or excluding vulnerable communities [11].

Artificial intelligence can support these goals in several ways, including resource optimisation, automation, forecasting and decision support. However, AI also has possible negative effects. These include high computational demand, unequal access to technology, unsafe automation and decisions that may be difficult to understand. Vinuesa et al. found that AI could support 134 Sustainable Development Goal targets while also potentially affecting 59 targets negatively [8]. This shows that AI cannot automatically be considered beneficial or sustainable.

Software plays an important role in modern sustainability solutions. Systems such as smart irrigation, energy-efficient buildings, waste monitoring and environmental sensing depend on software that converts real-world requirements into machine actions. However, people who understand these problems, such as teachers, farmers, sustainability professionals, facility managers and community organisations, may not have the programming knowledge needed to develop such systems.

General-purpose languages such as Python, Java and C are powerful and flexible. However, they also require users to understand syntax, libraries, hardware interfaces, dependencies and deployment methods. This level of complexity may be unnecessary for small and highly specific sustainability applications.

Developments in natural-language program synthesis, large language models and low-code platforms have made software development more accessible. Earlier research showed that users could describe a task in natural language and receive multiple possible programs written in a domain-specific language [1]. Grammar-prompting research later showed that formal Backus–Naur Form grammars could improve the structural correctness of generated outputs [2]. AI-assisted tools can now also create, modify and repair DSL grammars [5].

These developments suggest that AI can move beyond generating ordinary source code and instead help users create controlled, reusable and understandable domain-specific languages.

However, code generation and sustainability evaluation are still usually treated as separate activities. Studies on green code generation examine factors such as correctness, runtime, memory use, floating-point operations and energy consumption. Their findings show that AI-generated code is not always more efficient than human-written or expert-optimised code [4], [6]. The OECD also recommends evaluating

both the direct environmental cost of AI and the indirect positive or negative impact of the applications it supports [9].

GreenLangAI addresses this challenge by introducing sustainability constraints during DSL design and evaluating the generated backend code on the hardware where it is intended to run.

1.2 Problem Statement

Most AI coding assistants convert natural-language prompts into programs written in existing languages such as Python, Java or JavaScript. While this may improve developer productivity, it creates several problems when the user is building sustainability-focused applications.

First, generated source code may be difficult for non-programmers to understand, check or safely modify. Second, unrestricted AI-generated code may be syntactically correct but logically unsafe. Third, a solution described as environmentally friendly may still depend on excessive sensor checking, network communication, software libraries or powerful hardware. Finally, the energy consumed while using AI to generate and repair the application may be greater than the savings achieved by a small or rarely used system.

The central research problem is therefore the absence of an integrated framework that can:

- convert a sustainability problem into a task-specific language;
- include safety and resource-efficiency rules in that language;
- generate alternative implementations for different hardware devices;
- verify the correctness and safety of the generated program;
- measure the computational cost of the software;
- compare that cost with the environmental benefit of the application.

1.3 Research Gap

The reviewed studies provide valuable foundations, but they mainly address separate parts of the problem. Natural-language program synthesis generates programs in predefined DSLs and ranks possible outputs [1]. Grammar prompting uses specialised grammars to guide structured generation [2]. LLM4PLC combines language models with external validators to improve the reliability of industrial-control programs [3]. *Learn to Code Sustainably* evaluates generated code through correctness, runtime, memory and energy metrics [4]. DSL Assistant supports grammar creation, versioning, examples and automatic repair [5]. Apsan et al. show that code efficiency changes across hardware platforms and that code written by a green-software expert may outperform LLM-generated code [6].

However, these areas are generally studied independently.

Existing DSL-generation systems do not usually treat environmental sustainability as a central grammar-design requirement. Green-code studies normally evaluate code after it has already been generated instead of defining a language whose permitted commands encourage sustainable execution.

Current approaches also rarely combine all of the following within one lifecycle:

- requirement clarification;
- minimal grammar generation;
- safety verification;
- automatic repair;

- hardware-aware backend selection;
- AI-generation overhead;
- environmental benefit assessment;
- operational break-even analysis.

GreenLangAI is proposed to address this integrated gap.

1.4 Aim and Objectives

The aim of this study is to design a conceptual AI-assisted framework for generating accessible, verified and measurable sustainability-constrained DSLs for small, low-resource sustainable-development applications.

The objectives are:

1. To convert natural-language sustainability requirements into structured information about users, sensors, conditions, actions, timings and restrictions.
2. To generate a minimal BNF grammar containing only the commands required for the selected application.
3. To include sustainability policies such as local execution, controlled sensing frequency, sleep states, limited network communication and hardware limits within the DSL.
4. To generate and rank multiple candidate programs instead of accepting a single unrestricted AI response.
5. To validate generated programs through syntax, meaning, functionality, sustainability and safety checks.
6. To generate alternative backend implementations and select an efficient verified implementation for the target hardware.
7. To evaluate computational footprint, environmental handprint, social accessibility, AI-generation overhead and break-even.
8. To demonstrate the framework using a smart-classroom energy-management case study.

1.5 Research Questions

- **RQ1:** Can GreenLangAI convert natural-language sustainability requirements into syntactically valid and logically appropriate DSL programs?
- **RQ2:** Does minimal grammar prompting improve validity and reduce irrelevant commands compared with unrestricted generation?
- **RQ3:** How much can verification, automatic repair and human confirmation improve the reliability of generated programs?
- **RQ4:** Can sustainability rules included in the grammar reduce unnecessary computation, memory usage and network communication?
- **RQ5:** How does the efficiency of GreenLangAI-generated backends differ across hardware platforms?
- **RQ6:** Can hardware-aware backend selection perform better than directly generated AI code on the selected device?
- **RQ7:** After including AI-generation overhead, when does GreenLangAI reach computational break-even?

- **RQ8:** Can GreenLangAI improve understanding and program-modification performance among users with limited programming knowledge?

1.6 Research Contributions

This study proposes:

1. A complete lifecycle architecture for converting sustainability requirements into verified domain-specific programs.
2. A sustainability-constrained grammar model in which green execution policies are formal language rules rather than optional comments.
3. A candidate-ranking method based on requirement coverage, correctness, safety and predicted resource efficiency.
4. A closed-loop validation process combining parser checks, semantic rules, functional tests, safety contracts, repair and human approval.
5. A hardware-aware backend selector that evaluates alternative implementations on the intended device.
6. A GreenLang Sustainability Score that retains raw measurements and prevents incorrect programs from receiving a positive sustainability rating.
7. A dual-impact evaluation that separates computational footprint from environmental handprint.
8. A break-even model that includes one-time AI-generation costs and repeated execution costs.
9. A smart-classroom case study and reusable validation templates for future implementation.

2. Literature Review

2.1 AI-Based Code Generation

Natural-language program synthesis aims to reduce the difficulty of converting human instructions into formal computer programs.

Desai et al. proposed a framework that accepts a predefined DSL together with natural-language and DSL examples. It generates several possible programs and ranks them according to coverage, mapping and structural similarity. Their intended solution appeared first in more than 80% of benchmark cases and within the top three in more than 90% [1].

The study also highlighted the importance of previews and reverse explanations because natural-language instructions may be ambiguous. GreenLangAI adopts these ideas by generating multiple candidate programs and explaining them in plain language. However, instead of translating instructions only into an existing DSL, GreenLangAI aims to create or specialise a sustainability-focused DSL.

Modern LLM-based coding assistants can generate programs from open-ended prompts, but reliability remains a serious concern. LLM4PLC shows that one-step generation is not sufficient for specialised and safety-related systems. Its process combines prompt engineering, fine-tuning, grammar checking, compiler feedback, formal verification and user interaction. This improved generation success from 47% to 72% [3].

GreenLangAI therefore does not rely only on the confidence of the AI model. It uses independent validators, functional tests and iterative repair.

Unlike ordinary coding assistants, GreenLangAI does not treat Python or Java code as its main user-facing output. It first creates a restricted vocabulary and grammar for the selected sustainability task. The

resulting DSL can then be translated into different backend languages without changing the program understood by the user.

2.2 Domain-Specific Languages

A domain-specific language is designed for a limited field or activity. It uses words and commands that are familiar to users in that domain and removes unrelated general-purpose programming features.

For example, a smart-classroom DSL may use commands such as:

MONITOR

EMPTY_FOR

TURN_OFF

KEEP_ON

These commands may be easier to understand than sensor APIs, loops, timer objects and hardware libraries.

Wang et al. introduced grammar prompting, where an LLM first predicts a specialised BNF grammar for a particular task and then generates an output that follows that grammar. Their research showed improvements across semantic parsing, planning and molecular-generation tasks. Grammar-constrained decoding also guaranteed syntactic validity, although it required additional computation [2].

GreenLangAI adapts this idea by using minimal grammar not only for correct structure but also for safety and sustainability.

Mosthaf and Wąsowski's DSL Assistant demonstrates that generative AI can support grammar generation, grammar modification, example creation and automatic repair. Their system treats DSL development as an iterative process and includes versioning and context management. It automatically repaired 83 faulty grammars, representing an 86% repair rate in their evaluation [5].

GreenLangAI adopts grammar repair, versioning and the joint development of grammar and examples. Its additional contribution is the integration of sustainability contracts, hardware profiling and lifecycle impact assessment.

2.3 Green Software Engineering

Green software engineering aims to provide the required software function while reducing energy use, hardware demand and carbon impact.

The Software Carbon Intensity specification evaluates software emissions per functional unit and encourages improvements through energy efficiency, hardware efficiency and carbon awareness [10]. A functional unit may be defined as:

- one classroom-day monitored;
- one sensor event processed;
- one generated application deployed.

Vartziotis et al. evaluate AI-generated code using correctness, runtime, memory use, floating-point operations and energy consumption. They introduce Green Capacity as a correctness-filtered relative measure [4]. Their findings show that sustainability-oriented prompts do not consistently improve every program or model.

Apsan et al. also demonstrate that code efficiency depends on hardware. In their study, human-written code was more energy-efficient on a server and Raspberry Pi, while LLM-generated code performed better on the tested PC. Code written by a green-software expert was more efficient across all tested devices [6]. These findings show why GreenLangAI must evaluate generated code on the actual target hardware. GreenLangAI does not assume that shorter code is automatically greener. A simple DSL may still produce an inefficient backend, and its interpreter may add extra processing. Therefore, the framework reports correctness, runtime, energy, memory, network communication, dependencies and hardware information separately.

2.4 Low-Code and No-Code Systems

Low-code and no-code systems reduce the need for manual programming through visual interfaces, models, demonstrations and natural-language instructions.

Liu et al. distinguish between traditional low-code systems and LLM-supported low-code development. These approaches differ in their capabilities, limitations and use throughout the software-development lifecycle [7].

Low-code platforms improve accessibility, but they may also:

- restrict users to fixed components;
- create dependence on one vendor;
- reduce portability;
- hide how the program behaves internally.

GreenLangAI is positioned between no-code interaction and formal programming. Users may begin with natural language, but the system produces an explicit and readable DSL.

This intermediate language allows users and reviewers to:

- inspect the rules;
- compare different backend versions;
- simulate behaviour;
- maintain version history;
- retain the program separately from the original AI conversation.

2.5 AI for Sustainable Development

AI can support sustainable development through demand prediction, resource management, smart buildings, environmental monitoring and accessible automation.

However, the environmental benefits of an AI application must be compared with the environmental cost of creating and operating it.

The OECD distinguishes between:

- the direct environmental footprint of developing, running and disposing of AI systems;
- the indirect positive or negative effects of the applications enabled by AI [9].

This distinction is important for GreenLangAI.

The electricity used for AI generation and program execution forms the **computational footprint**. The electricity saved by switching off unused classroom devices forms the **environmental handprint**.

GreenLangAI may potentially support:

- **SDG 4:** Quality Education;
- **SDG 7:** Affordable and Clean Energy;
- **SDG 9:** Industry, Innovation and Infrastructure;
- **SDG 10:** Reduced Inequalities;
- **SDG 12:** Responsible Consumption and Production;
- **SDG 13:** Climate Action.

These are potential contributions and should not be presented as completed achievements without experimental evidence.

2.6 Identified Research Gap

Approach	Main contribution	Gap relative to GreenLangAI
Natural-language synthesis [1]	Ranked natural-language-to-DSL programs	Integrates grammar, verification, hardware profiling, footprint, handprint and break-even
Grammar prompting [2]	Minimal BNF grammar and constrained decoding	Sustainability is not the main grammar-design goal
LLM4PLC [3]	Verifier-guided program generation	Focused on industrial PLCs without a green-impact model
Learn to Code Sustainably [4]	Correctness, runtime, memory, FLOPs and energy metrics	Evaluates generated code rather than a generated sustainable DSL
DSL Assistant [5]	Grammar creation, repair, versioning and examples	Does not primarily focus on sustainable hardware deployment
Energy-efficient code through LLMs [6]	Cross-hardware testing and green-expert baseline	Does not create a sustainability-constrained DSL
GreenLangAI	Integrates grammar, verification, hardware profiling, footprint, handprint and break-even	Remains conceptual and requires prototype validation

The research gap is not that AI has never been used to generate DSLs. The gap is the absence of an end-to-end system where sustainability is treated as a formal language constraint and generated programs are verified, tested on target hardware and evaluated according to both computational cost and real-world benefit.

3. Research Methodology

3.1 Research Design

This study follows a conceptual design-and-evaluation approach. It develops a new framework from gaps identified in existing literature and explains how a future prototype should be evaluated.

The study does not claim that GreenLangAI has already produced experimentally verified energy savings. Numerical examples are clearly presented as assumptions and must later be replaced with actual measurements.

The unit of analysis is a GreenLangAI-generated sustainability application containing:

1. a natural-language requirement;
2. a specialised grammar;
3. one or more DSL programs;
4. generated backend code;
5. verification results;
6. a sustainability report.

The first demonstration area is smart-classroom energy management.

3.2 Literature Review Method

A targeted narrative review was conducted using six major reference papers and authoritative sustainability sources.

The literature covers:

- natural-language program synthesis [1];
- grammar-constrained DSL generation [2];
- verified specialised code generation [3];
- green-code evaluation [4];
- AI-assisted DSL development [5];
- cross-hardware energy measurement [6];
- low-code development [7];
- AI and Sustainable Development Goals [8];
- environmental impact measurement for AI [9];
- software carbon-intensity measurement [10];
- the United Nations sustainable-development agenda [11].

Studies were included when they contributed to at least one GreenLangAI design area, such as:

- requirement translation;
- grammar generation;
- constrained output;
- verification;
- repair;
- accessibility;
- energy measurement;
- hardware dependence;
- lifecycle assessment;
- sustainable-development impact.

The concepts identified in the literature were connected to GreenLangAI components and evaluation criteria. Since the review is focused, a broader systematic literature review would be required for future journal publication.

3.3 Framework Development Method

The framework was developed by converting each limitation identified in previous research into a design requirement.

For example:

- ambiguity in user instructions led to the requirement analyser;
- invalid structured output led to minimal BNF grammar and constrained generation;
- unsafe code led to syntax, semantic, functional and safety validators;
- changing DSL requirements led to versioning and automatic repair;
- hardware-dependent energy use led to target-device profiling;
- the direct and indirect impacts of AI led to separate footprint and handprint evaluation;
- AI-generation cost led to the break-even model.

The framework is modular. An initial prototype may use predefined smart-classroom grammar templates and a rule-based interpreter. Future versions may use LLM-assisted grammar creation, formal verification and multiple backend languages.

3.4 Case-Study Selection

Smart-classroom energy management was selected because it is relevant to educational institutions, easy for non-specialists to understand and suitable for testing with simulated occupancy data.

The case includes clear:

- inputs;
- conditions;
- time rules;
- actions;
- safety restrictions.

It also allows the study to separate two different ideas:

1. electricity saved by controlling classroom devices;
2. resources used by the GreenLangAI software itself.

3.5 Evaluation Criteria

Dimension	Indicator	Purpose
Functional correctness	Requirements and tests passed	Mandatory acceptance condition
Syntactic validity	Parser acceptance rate	Measures grammar quality
Semantic validity	Compliance with domain rules`	Measures logical correctness
Safety	Safety-rule violations	Indicates deployment readiness
Runtime	Median execution time	Measures computational efficiency
Memory	Median and peak memory	Tests low-resource compatibility

Energy	Joules or watt-hours per functional unit	Measures operational footprint
Network	Requests and bytes transferred	Measures communication cost
Dependencies	Number and installed size	Measures deployment complexity
Usability	Time, errors, completion and understanding	Measures social accessibility
AI overhead	Generation requests, time and available energy estimate	Supports lifecycle accounting
Handprint	Avoided device use and electricity saved	Measures application benefit
Break-even	Time or executions needed to recover overhead	Measures net sustainability

4. Proposed GreenLangAI Framework

4.1 Framework Overview

GreenLangAI is an AI-assisted development framework that converts a natural-language sustainability requirement into a controlled domain-specific language and a verified executable implementation.

It does not create an unrelated language for every individual user. Instead, it combines:

- a standard core grammar;
- reusable domain modules;
- task-specific customisation.

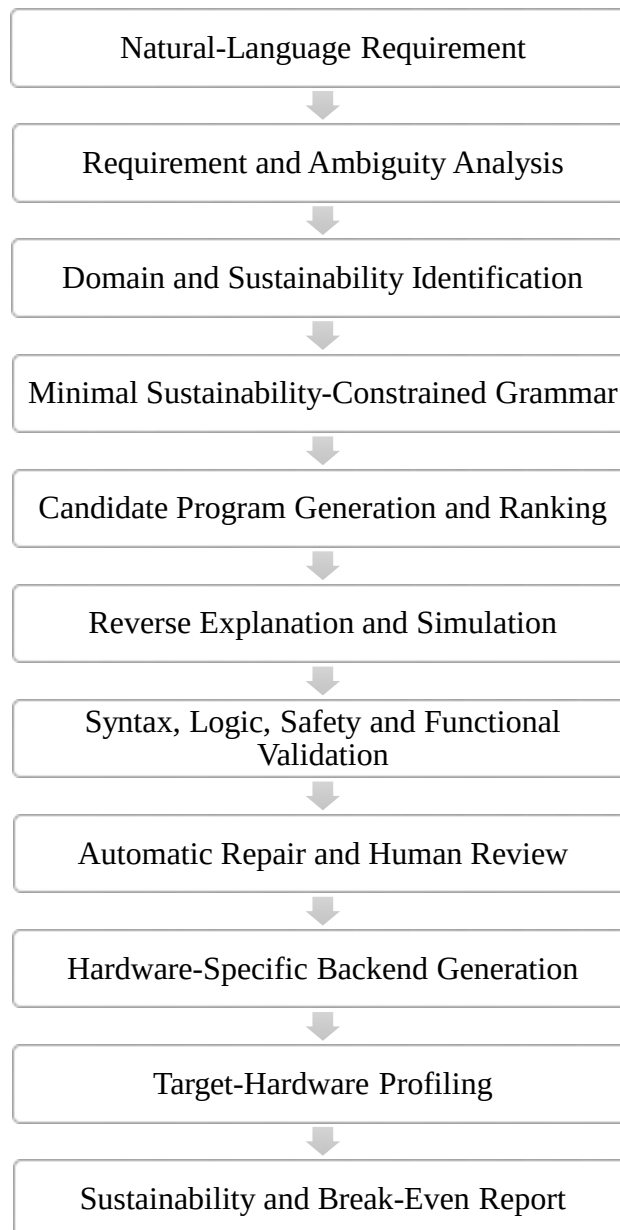
This approach supports consistency, maintenance, version control and interoperability.

The main design principle is **sustainability by construction**. Rather than generating ordinary code first and requesting optimisation later, GreenLangAI limits the language to approved constructs and includes sustainability policies directly within the program.

The final backend implementation is still measured before deployment because a sustainability policy alone does not prove that the application is efficient.

4.2 Architecture

Figure 1. Proposed GreenLangAI architecture and verification lifecycle



4.3 Major Components

Natural-Language Interface

Accepts the user's requirement in simple language and records the domain, intended device and user constraints.

Requirement and Ambiguity Analyser

Identifies sensors, resources, conditions, actions, durations, recipients and exceptions. It asks questions when essential information is missing.

Domain and Sustainability Intent Identifier

Recognises the application area and intended environmental benefit without assuming that the final system will automatically be sustainable.

Minimal Grammar Generator

Selects or creates the smallest relevant part of a validated BNF grammar.

Candidate Program Generator and Ranker

Produces several DSL solutions and ranks them according to requirement coverage, logical suitability, safety, simplicity and predicted resource demand.

Reverse Explanation and Simulation Preview

Explains each program in normal language and shows how it behaves with sample inputs.

Validation and Repair Engine

Checks syntax, logic, safety, functional requirements and resource policies. Errors are returned to the AI for a limited number of correction attempts.

Version and Co-evolution Manager

Stores versions of grammars, examples and programs. When the grammar changes, affected programs are tested again.

Backend Generator

Converts the DSL into executable code, initially in Python and later in languages such as C or C++.

Hardware Profiler and Selector

Tests verified implementations on the intended device and selects the best version according to measured performance.

Sustainability Assessor

Reports raw resource measurements, computational footprint, application handprint, GreenLang Sustainability Score and break-even.

Human Approval and Audit Log

Requires a person to approve deployment and stores the generated program, test results, changes and reviewer information.

4.4 Operational Workflow

1. The user describes the sustainability problem and target device.
2. The analyser extracts requirements and asks for missing information.
3. A validated domain module is selected.
4. The module is converted into a minimal task-specific grammar.
5. Sustainability and safety rules are included.
6. Several DSL programs are generated.
7. Programs are ranked and explained in simple language.
8. The selected program is parsed, simulated and tested.
9. Failed tests trigger limited automatic repair.
10. Unresolved errors are sent for human review.
11. Alternative backend versions are generated.

12. Correct versions are tested on the intended hardware.
13. Performance, AI overhead, environmental benefit and break-even are reported.
14. A reviewer approves, rejects or modifies the final system.

4.5 Sample DSL

APPLICATION SmartClassroom

TARGET RaspberryPi

MONITOR classroom_occupancy EVERY 30 SECONDS

WHEN classroom_occupancy IS EMPTY FOR 10 MINUTES

TURN_OFF lights

TURN_OFF fans

SEND_ALERT TO maintenance_staff

END

WHEN classroom_occupancy IS OCCUPIED

KEEP_ON lights

KEEP_ON fans

END

POLICY

RUN_LOCAL

SLEEP_BETWEEN_READINGS

SEND_ONLY_WHEN_STATUS_CHANGES

KEEP emergency_light ON

MAX_NETWORK_REQUESTS 2 PER HOUR

END

A simplified BNF grammar is:

<program> ::= <application> <target> <monitor_rule>

<control_rule>+ <policy_block>

<monitor_rule> ::= "MONITOR" <sensor> "EVERY" <duration>

<control_rule> ::= "WHEN" <condition> <action>+ "END"

<action> ::= "TURN_OFF" <device>

| "KEEP_ON" <device>

| <alert>

<policy_block> ::= "POLICY" <policy>+ "END"

The policy section converts sustainability and safety intentions into rules that the system can check automatically. However, these rules must still be validated through measurement.

4.6 Safety and Validation Mechanisms

GreenLangAI uses five levels of validation:

1. **Syntactic validation:** Checks whether the program follows the DSL grammar.
2. **Semantic validation:** Checks whether the commands are meaningful for the declared sensors and devices.
3. **Functional validation:** Checks whether the program produces the required output.
4. **Safety validation:** Protects mandatory rules such as emergency lighting and anti-oscillation delays.
5. **Sustainability validation:** Checks resource budgets, communication limits and actual measured performance.

Human approval remains necessary before deployment. Users are shown the DSL, its explanation, test results, warnings, backend details and sustainability report.

GreenLangAI is not intended to deploy unrestricted code automatically in safety-critical systems.

5. Smart Classroom Case Study

The case study considers a classroom with occupancy sensing, six lights and four fans. Devices may remain switched on even when the room is empty, leading to unnecessary electricity use.

The user requirement is:

Monitor the classroom. If nobody is present continuously for ten minutes, switch off the normal lights and fans, keep the emergency light on and notify maintenance only when the state changes.

The requirement analyser identifies:

- occupancy sensor;
- ten-minute empty-room threshold;
- normal lights;
- fans;
- emergency-light exception;
- maintenance recipient;
- change-only notification.

The system then asks for the target hardware and sensor-checking interval. Assuming Raspberry Pi deployment and a thirty-second interval, only the required occupancy, timing, control, alert and policy commands are included in the grammar.

GreenLangAI may generate three candidates:

1. periodic local monitoring;
2. event-based local monitoring;
3. cloud-connected monitoring.

The cloud option may be ranked lower if continuous internet communication is unnecessary. The event-based option may be preferred when reliable hardware events are available. Otherwise, controlled periodic monitoring may be selected.

When the classroom is occupied, lights and fans remain on. When it becomes empty, a timer starts. If someone returns before ten minutes, the timer resets. If the room remains empty for ten minutes, the normal lights and fans are switched off and one notification is sent.

A single false sensor reading does not trigger shutdown because the ten-minute condition has not been completed. The emergency light remains protected.

Element	Case-study specification
Input	Occupancy state
Main condition	Classroom continuously empty for ten minutes
Actions	Switch off normal lights and fans and send one notification
Safety rule	Emergency light must remain on
Sustainability rules	Local execution, controlled sensing, sleep states and limited communication
Hardware	Raspberry Pi or equivalent edge device
Functional unit	One classroom-day
Baselines	Manual Python, direct AI Python and green-expert-optimised code

The application may save electricity by reducing unnecessary device operation. However, GreenLangAI's unique contribution must be evaluated separately through software efficiency, accessibility, verification and maintainability.

6. Proposed Evaluation Framework

6.1 Functional Test Cases

ID	Test condition	Expected output	Purpose
TC-01	Classroom remains occupied	Lights and fans remain on	Normal operation
TC-02	Classroom remains empty for ten minutes	Devices switch off once	Energy-saving action
TC-03	Empty for eight minutes and then occupied	Timer resets	Prevents incorrect shutdown
TC-04	Empty for two hours	One shutdown and one alert	Avoids repeated operations
TC-05	One false empty reading	No shutdown	Sensor-noise tolerance
TC-06	Attempt to switch off emergency light	Command rejected	Safety protection

TC-07	Internet unavailable	Local control continues	Offline operation
TC-08	Invalid DSL command	Parser error and no execution	Syntax protection
TC-09	Duration missing	System requests clarification	Ambiguity handling
TC-10	One hundred classrooms simulated	Independent states handled correctly	Scalability

Correctness is compulsory. A program that fails a functional or safety test cannot receive a positive sustainability rating.

6.2 Computational Sustainability

Four implementations should be compared:

- manually written Python;
- directly AI-generated Python;
- green-expert-optimised Python or C;
- GreenLangAI-generated backend.

Each version must perform the same task, use the same dataset and run on the same hardware.

The following should be measured:

- execution time;
- CPU usage;
- memory consumption;
- energy per functional unit;
- dependency count and size;
- network requests and data transferred;
- executable size;
- performance on both a personal computer and target edge device.

Percentage improvement may be calculated as:

$$\text{Improvement (\%)} = \frac{\text{Baseline} - \text{GreenLangAI}}{\text{Baseline}} \times 100$$

Raw measurements must always be reported with the percentage.

When direct power measurement is unavailable, CPU, memory and runtime may be used as efficiency indicators, but they should not be presented as exact carbon emissions.

6.3 Environmental Impact

The classroom application’s environmental benefit can be estimated through avoided operating time.

$$\text{Energy saved} = \text{Device power} \times \text{Avoided time}$$

For illustration:

- four fans at 70 W each = 280 W;
- six lights at 20 W each = 120 W;
- combined load = 400 W or 0.4 kW.

Avoiding two unnecessary operating hours per day would save:

$$0.4 \times 2 = 0.8 \text{ kWh per day}$$

For 25 working days:

$$0.8 \times 25 = 20 \text{ kWh per month}$$

These are illustrative calculations and not experimental findings.

A manually written Python program could produce the same device savings. Therefore, GreenLangAI's own value must be demonstrated through improved accessibility, verification, repeatability or lower software resource use.

6.4 Social Sustainability

A usability study may include participants with limited, basic and intermediate programming knowledge. Participants can complete equivalent Python and GreenLang tasks such as:

- explaining the program;
- changing the empty-room duration;
- adding an alert;
- finding an unsafe command;
- adding a new condition.

The study may record:

- task-completion time;
- success rate;
- number of errors;
- assistance required;
- understanding score;
- ease-of-use rating;
- ability to identify safety or logical mistakes.

Participant data should be anonymised, and appropriate consent procedures should be followed.

6.5 AI Generation Overhead

AI-assisted generation also consumes computational resources.

The evaluation should record:

- number of AI requests;
- failed generations;
- repair attempts;
- generation time;
- available energy or carbon estimates.

Total GreenLangAI energy may be represented as:

$$E_{\text{total}} = E_{\text{generation}} + E_{\text{validation}} + N \times E_{\text{execution}}$$

Reusing approved grammars, caching templates and running the final application without continuous AI use can reduce repeated costs.

6.6 Break-Even Analysis

GreenLangAI reaches computational break-even only when its initial generation and validation cost is recovered through lower repeated execution costs.

$$\text{Break-even units} = \frac{E_{\text{generation}} + E_{\text{validation}}}{E_{\text{baseline per unit}} - E_{\text{GreenLangAI per unit}}}$$

The denominator must be positive. If GreenLangAI uses more execution energy than the baseline, energy-based break-even will not occur.

Other benefits, such as safety and usability, may still remain.

7. Discussion

GreenLangAI treats AI-assisted programming as more than a one-prompt code-generation task. It considers language design, validation, sustainability and governance together.

Its strongest conceptual contribution is the use of sustainability-constrained grammar. Commands such as `RUN_LOCAL` and `SEND_ONLY_WHEN_STATUS_CHANGES` are easier to audit when they are formal DSL rules rather than informal statements included only in a prompt.

The framework also recognises that accessibility and computational efficiency may not always improve together. A DSL may be easier for users to understand even if its interpreter is slightly slower than optimised code. This would support social sustainability but not necessarily computational sustainability. Similarly, a highly efficient backend may not be appropriate if users cannot understand or safely modify it.

Hardware profiling is essential because no single model, prompt or program can be considered universally green [6]. Results measured on a laptop cannot automatically be applied to a Raspberry Pi, server or microcontroller.

Multiple candidate programs and reverse explanations can reduce ambiguity, but they cannot remove it completely. Domain experts are still required to define safety rules, assumptions and acceptable operating conditions.

GreenLangAI may support local capability building by allowing students, facility managers and domain experts to work with understandable commands. However, genuine digital inclusion also depends on affordability, language support, accessibility design, training and trust.

8. Practical and Theoretical Contributions

8.1 Practical Contributions

GreenLangAI provides:

- a reusable workflow for developing small sustainability applications;
- a readable DSL that can be reviewed separately from the AI conversation;
- a validation checklist covering syntax, meaning, functionality, safety and sustainability;
- a comparison method using manual, AI-generated and expert-optimised baselines;
- a deployment report separating software metrics, application savings and AI overhead;
- a smart-classroom template that can initially be tested using simulated data.

8.2 Theoretical Contributions

The study introduces:

- sustainability-constrained grammar as a central concept in AI-assisted DSL development;
- a lifecycle model connecting DSL generation with green-software evaluation;
- a dual-impact model separating footprint and handprint;
- a correctness-gated GreenLang Sustainability Score;
- a hardware-dependent understanding of generated-code sustainability;
- a break-even model connecting AI-generation cost with long-term execution.

9. Limitations

This study has several limitations.

- GreenLangAI is currently conceptual and has not been implemented as a complete system.
- The literature review is focused and does not replace a full systematic review.
- The smart-classroom example may not represent agriculture, healthcare, transport or industrial systems.
- Automatic grammar generation may produce hallucinations, underfitting, overfitting or logical errors.
- Correct syntax does not guarantee correct or safe behaviour.
- Interpreter overhead may make the DSL less efficient than optimised general-purpose code.
- Reliable energy measurement requires controlled hardware.
- Carbon estimates depend on electricity sources, system boundaries and hardware assumptions.
- Results involving BCA students may not represent farmers, NGOs, facility managers or other target users.
- Repeated generation and repair may increase AI overhead.
- Too many incompatible DSLs could create maintenance problems.
- The framework should not be used in safety-critical environments without expert supervision and institutional approval.

10. Future Scope

The next stage should involve developing a limited smart-classroom prototype.

Future work may include:

- creating a GreenLang core grammar;
- building a Python interpreter and simulator;
- developing a benchmark of requirements and expected programs;
- comparing unrestricted, complete-grammar and minimal-grammar prompting;
- generating Python and C/C++ backends;
- testing on computers and Raspberry Pi devices;
- integrating hardware power measurement;
- developing reusable modules for water, irrigation, waste and renewable energy;
- supporting regional-language input;

- using formal methods for safety verification;
- conducting usability testing with actual domain users;
- developing a Green Deployment Certificate;
- using smaller local language models;
- publishing code, datasets, grammars and raw results.

11. Conclusion

This paper proposes GreenLangAI, a verified and hardware-aware framework for generating sustainability-constrained domain-specific languages from natural-language requirements.

The framework combines requirement clarification, minimal grammar generation, candidate ranking, plain-language explanation, automatic validation, repair, version control, hardware-specific backend generation and lifecycle sustainability reporting.

GreenLangAI's originality does not come from claiming that AI-generated DSLs are entirely new. Its contribution lies in combining sustainability-oriented language design with verification, hardware testing and measurable lifecycle impact.

The framework rejects the assumption that shorter or AI-generated code is automatically greener. Correctness, safety, energy use, memory, communication, hardware, AI overhead, usability and environmental benefit must all be evaluated.

The smart-classroom case study shows how the system could create understandable energy-management rules while protecting emergency lighting and supporting local operation.

A working prototype and controlled experiment are still required before claiming energy or carbon reductions. If validated, GreenLangAI could connect domain knowledge, accessible programming and evidence-based sustainable software development.

References

1. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, S. R., and S. Roy, “Program Synthesis using Natural Language,” arXiv:1509.00413, 2015.
2. B. Wang, Z. Wang, X. Wang, Y. Cao, R. A. Saurous, and Y. Kim, “Grammar Prompting for Domain-Specific Language Generation with Large Language Models,” in Advances in Neural Information Processing Systems, vol. 36, 2023.
3. M. Fakih, R. Dharmaji, Y. Moghaddas, G. Quiros Araya, O. Ogundare, and M. A. Al Faruque, “LLM4PLC: Harnessing Large Language Models for Verifiable Programming of PLCs in Industrial Control Systems,” arXiv:2401.05443, 2024.
4. T. Vartziotis, I. Dellatolas, G. Dasoulas, M. Schmidt, F. Schneider, T. Hoffmann, S. Kotsopoulos, and M. Keckeisen, “Learn to Code Sustainably: An Empirical Study on LLM-based Green Code Generation,” arXiv:2403.03344, 2024.
5. M. M. Mosthaf and A. Wąsowski, “From a Natural to a Formal Language with DSL Assistant,” arXiv:2408.09766, 2024.

6. R. Apsan, V. Stoico, M. Albonico, R. Dhar, K. Vaidhyanathan, and I. Malavolta, “Generating Energy-Efficient Code via Large-Language Models—Where are we now?” in Proc. 48th IEEE/ACM International Conference on Software Engineering (ICSE), 2026, doi: 10.1145/3744916.3764522.
7. Y. Liu, J. Chen, T. Bi, J. Grundy, Y. Wang, J. Yu, T. Chen, Y. Tang, and Z. Zheng, “An Empirical Study on Low-Code Programming using Traditional vs Large Language Model Support,” arXiv:2402.01156, 2025 revision.
8. R. Vinuesa et al., “The role of artificial intelligence in achieving the Sustainable Development Goals,” Nature Communications, vol. 11, art. 233, 2020, doi: 10.1038/s41467-019-14108-y.
9. OECD, Measuring the Environmental Impacts of Artificial Intelligence Compute and Applications: The AI Footprint. Paris, France: OECD, 2022.
10. Green Software Foundation, Software Carbon Intensity (SCI) Specification, ISO/IEC 21031:2024, version 1.1, accessed July 2026.
11. United Nations General Assembly, Transforming our world: the 2030 Agenda for Sustainable Development, A/RES/70/1, 2015.