What Is Hiding in the Energy Footprint of AI Planning? Initiating Energy Accountability

Ilche Georgievski[1]

[1]Service Computing Department, IAAS, University of Stuttgart, Germany

Abstract

The growing reliance on AI points out the critical need to research its energy consumption. While Green AI has primarily focused on profiling and optimising the energy use of machine learning models, AI planning remains unexplored in this regard despite its high runtime cost and broad applicability. Therefore, this paper initiates the study of energy accountability in AI planning. We make three contributions. First, we analyse factors that may influence the energy and carbon footprint of AI planning and propose ten hypotheses to guide a structured research agenda on energy-aware automated planning. Second, we introduce an energy measurement framework, PLANERGYM, tailored to the characteristics of AI planning systems, supporting hypothesis-driven and reproducible evaluations. Third, we apply the framework in a case study on classical planners, showing that while runtime correlates with energy consumption, it alone is insufficient to capture the full energy footprint due to variability in power draw. Our observations show the need for energy-aware design, evaluation, and reporting practices in AI planning.

Keywords

AI Planning, Energy Accountability, Classical Planners, Energy Profiling, Green AI

1. Introduction

Computation does not take time only, but also impacts the environment. As software systems increase in scale and complexity, so too does their energy and carbon footprint. The Information and Communication Technologies (ICT) sector was estimated to contribute up to 3.9% of global carbon emissions in 2021 [1], a share likely on the lower end today. This has led to calls for treating energy and carbon as first-class resources in software, e.g., [2], forming the areas of *green software* [3] and *sustainable computing* [4].

The environmental footprint of software is driven in large part by the rapid growth of AI. *Green AI* has emerged in response, focusing primarily on energy accountability and resource efficiency in machine learning [5]. Existing contributions are promoting the reporting of energy consumption alongside accuracy and encouraging methodological innovations that reduce computational cost [6]. In contrast, *AI planning* or *automated planning*, a fundamental area of AI concerned with generating action sequences to achieve given user goals, has thus far remained outside this conversation, despite being computationally intensive in nature.

Traditional evaluation criteria in AI planning have focused on *runtime* and *plan quality* as principal performance metrics. While runtime, which is the wall-clock time to solve a planning problem, has been treated as a proxy for computational efficiency, it offers only a partial picture. In contrast, *computational cost* encompasses the total resources consumed during execution, such as CPU cycles and memory, and when combined with power consumption, offers a more complete view of a system's energy and carbon footprint. Existing assumptions underlying comparative evaluation in AI planning are not supported empirically and seem to affect AI planners differently [7]. It should not be unexpected that AI planners with similar runtimes may draw different amounts of power due to differences in hardware utilisation

2nd Workshop on Green-Aware Artificial Intelligence, 28th European Conference on Artificial Intelligence (ECAI 2025), October 25–30, 2025, Bologna, Italy

*Corresponding author.

ilche.georgievski@iaas.uniÃ§stuttgart.de (I. Georgievski)

0000-0002-0877-7063 (I. Georgievski)

and reasoning dynamics, leading to distinct energy profiles. As such, despite this historical emphasis on efficiency, AI planning still lacks direct methods and metrics for assessing energy usage.

The environmental costs of AI planning extend beyond planner execution for performance evaluations. Standard practices, such as domain model engineering, iterative validation, debugging, and AI planner tuning, all involve repeated runs of AI planners. Yet, the environmental impact of these activities remains unexplored, and the cumulative experimentation that underlies reported performance results of AI planners is rarely made transparent. As a result, the full energy footprint of AI planning systems and workflows is currently hidden from view.

We therefore ask: What is hiding in the energy footprint of AI planning? In doing so, we initiate a call for *energy accountability* in AI planning, with the objective of uncovering and ultimately optimising energy usage alongside traditional metrics, thereby making a shift toward the development and evaluation of *energy-aware AI planning*. To this end, we make the following contributions:

- We frame energy efficiency as a critical and currently overlooked dimension in AI planning, and we propose ten hypotheses to guide future empirical investigations.
- We introduce a dedicated *energy measurement framework* tailored to the operational characteristics of AI planners, enabling systematic and reproducible energy analysis.
- We demonstrate the application of this framework through an illustrative case study on classical planners, giving initial insights into their energy behaviour.

The remainder of the paper is organised as follows. Section 2 presents the key background on AI planning. Section 3 introduces the energy dimension, including the hypotheses. Section 4 describes the proposed energy measurement framework. Section 5 presents the case study on classical planners, Section 6 concludes the paper.

2. Background on AI planning

AI planning is concerned with solving planning problems, where a basic planning problem consists of an initial state of the world from which planning would start, a goal state, which describes a user objective, and a set of actions that enable moving from one state to another, thus changing the world [8]. The set of actions is called a domain model and is a formal and templated representation of the world. Such a planning problem represents an input to an AI planner, which is a software tool concerned with the computation of plans, that is, sequences of actions whose execution in the initial state would lead to the goal state, thus satisfying the given user objectives.

AI planning development involves three main tasks: domain model engineering, algorithmic engineering, and systems engineering. The community's main focus is on theory and algorithm engineering, with a large body of research that focuses on making plan generation efficient (e.g., using domain-independent heuristics [9] and landmarks [10]) and capable of handling complex planning problems (e.g., probabilistic planning [11] and Hierarchical Task Network (HTN) planning [12, 13]). Domain model engineering focuses on the challenges of formulating domain models (e.g., domain model quality [14], and modelling languages such as the Planning Domain Definition Language (PDDL) [14]). Another body of work focuses on the challenges of engineering planning systems, placing attention on system architecting, interoperability, integration, reliability, and performance, e.g., [15, 16, 17, 18].

A standard practice for demonstrating the performance of AI planners is to run them on benchmarks composed of domain models and problem instances of varying difficulty. This practice has been strongly shaped by the International Planning Competitions (IPCs), held occasionally since 1998.[1] The IPCs have driven the development of benchmark sets and provided ranked comparisons of AI planners, thereby establishing standards for evaluating runtime, plan quality, and coverage.

Over the years, IPCs have diversified their focus by featuring tracks for different planning problems. Classical planning, which assumes deterministic actions and full observability, remains central, but

[1] https://www.icaps-conference.org/competitions/

additional tracks have been introduced for temporal, probabilistic, HTN, and reinforcement learning-based planning. Each track brings its own specification language, such as PDDL, and its own evaluation criteria and resource constraints.

To facilitate standardised and reproducible evaluations, the IPC introduced container-based execution environments (using Singularity, which is designed for high-performance computing that supports resource isolation and performance predictability) in recent editions. Each planner is distributed with a definition file that sets up all dependencies and builds the system. Resource limits, such as CPU time, memory, and number of cores, are strictly controlled per track.

The IPC infrastructure, benchmarks, and execution methodology provide a foundation for comparative evaluation. They are thus well-suited for extending evaluation dimensions to include energy consumption, which is not currently considered but can be studied by leveraging existing resources. In this work, we use IPC benchmarks and containerised planners as the basis for proposing energy-aware evaluation and applying our framework in a case study.

3. Energy Dimension in AP Planning

This section builds the case for recognising energy as a first-class dimension in AI planning. It begins by situating AI planning as an inherently computationally complex method, and then explores both established and plausible factors impacting resource use. Based on this analysis, we formulate a set of hypotheses intended to serve as a research roadmap for uncovering and understanding the energy and carbon footprint of AI planning systems.

3.1. Computational Characteristics

Three fundamental characteristics underlie the computational complexity of AI planning, making systems employing this technique likely to exhibit a non-negligible energy and carbon footprint.

Problem complexity. Planning problems are well known for their combinatorial complexity. Typical planning problems involve search spaces that grow exponentially with problem size; even small domains yield state spaces of 10^9 nodes, and real-world domains may involve thousands to millions of reachable states, high branching factors, and deep solution paths. This makes the theoretical complexity a strong indication of heavy computational demands; even classical planning is PSPACE-complete [19]. Such a combinatorial explosion directly suggests substantial energy usage.

Algorithmic complexity. Many existing planning approaches employ sophisticated techniques and algorithms designed to reduce the search space and improve plan quality. However, these techniques and algorithms are themselves computationally demanding, potentially requiring extensive processor cycles and memory. In this context, we distinguish three key computational intensities: preprocessing intensity, problem-solving intensity, and heuristics intensity. **Preprocessing intensity** refers to the computational load of AI planning before problem-solving (or plan generation). Many modern AI planners incorporate preprocessing for various reasons, including domain analysis, problem grounding (converting lifted problems into a propositional form), problem representation optimisation, and heuristic computation. These operations make preprocessing computation heavy, rivaling problem-solving in time (and, by extension, energy). **Problem-solving intensity** refers to the intensity of algorithms used to traverse the search space and generate plans. Two planners tackling the same planning problem can differ in memory and CPU cycles. Implementation choices for these algorithms (e.g., data structures) can also affect these consumptions. **Heuristics intensity** refers to the computational cost of heuristic computation and evaluation. One can observe that modern AI planners intentionally spend more processing power and keep larger data structures in memory to obtain stronger and sophisticated heuristics. This work inevitably manifests as additional energy per planning call.

Execution complexity. Planning systems are typically invoked repeatedly and dynamically in real-world contexts. Unlike in the machine learning lifecycle, where training a model is a one-off or infrequent job and inference is performed on demand, in the AI planning lifecycle, planning is executed on demand, each time a new goal or world state appears, which can happen many times per hour and even per minute. For example, invoking an AI planning system whenever a robot needs a new route or the game AI gets a new objective. This on-demand invocation pattern means energy costs compound over time, and even modest savings per planning call may be impactful at scale, especially for AI planning systems deployed in resource-constrained environments, such as edge devices.

3.2. Energy Hypotheses

Having established that the computational intensity of AI planning makes energy consumption an important dimension, we next identify factors that may influence energy efficiency. While no prior study has characterised energy use in AI planning, we can look at the extensive literature on runtime performance for indicators of where energy variation is likely to arise. As these factors may imply corresponding energy effects and are yet to be confirmed by research, we characterise them as hypotheses that may influence the energy and carbon footprint of AI planning.

It is often assumed that faster algorithms are more efficient overall, with the intuition that less runtime means less energy consumed. However, measuring time does not give a realistic view of energy consumption because of how instructions interact with the hardware [20]. That is, AI planners that finish faster may achieve that speed by running CPU-intensive heuristic evaluations or keeping large data structures in memory. Such optimisations increase power draw even when reducing runtime. This suggests that minimising runtime and minimising energy consumption may be fundamentally different objectives.

H 1 *Runtime and energy consumption are not positively correlated across different planning algorithms when solving the same problem instances on a fixed hardware platform.*

Optimal planning guarantees the best possible plan, such as shorter makespan or lower total cost, but does not necessarily require proportionally more search effort. AI planning systems that employ optimisation mechanisms, such as bounds, heuristics, and macros, may find near-optimal plans quickly, while exhaustive search may reuse earlier computations. Consequently, plan improvements can be gained with little or no additional energy, suggesting that the relationship between plan quality and energy consumption may not be straightforward.

H 2 *The energy required to find a plan is not monotonically increasing with respect to the plan's quality.*

AI planning is typically implemented as a pipeline consisting of distinct phases, such as domain model engineering, problem parsing, grounding, preprocessing and optimisation, problem-solving, and post-processing. These phases differ in their computational complexity and resource usage patterns, suggesting variability in their respective energy consumption profiles. Identifying and quantifying these differences can guide targeted energy optimisations.

Domain model engineering choices in AI planning have implications beyond plan quality. Existing work has shown that seemingly equivalent domain model formulations can lead to different performances of AI planners [21]. The configuration of domain models, including the choice of action schemas, predicate rearrangement, and constraints, can cause runtime variations on the same planning problems. The performance differences stem from how different domain configurations interact with the mechanisms of an AI planning system. These domain configuration decisions likely affect the energy footprint of AI planning systems, suggesting that energy-aware AI planning must also consider upstream decisions made in the pipeline.

H 3 *The energy consumption of AI planning systems varies when using semantically equivalent but structurally different domain model configurations when solving the same planning problem.*

Complex domain models can significantly impact computational requirements. Research has shown that more expressive domain features, while enabling more realistic and natural problem representations, often incur computational overhead. For example, derived predicates and axioms, which reduce domain modelling effort, can increase the complexity of plan generation due to the lack of compilation mechanisms in polynomial time [22]. The compilation approaches that transform complex domain features into simpler STRIPS representations[2] often result in larger problem descriptions. This computational burden translates directly to energy consumption through increased CPU cycles, memory access patterns, and extended runtimes.

H 4 *The energy consumption of AI planning systems positively correlates with the expressiveness features utilised in domain models.*

Certain operations during preprocessing incur computational overhead. Consider, for example, Fast Downward (FD), which is a state-of-the-art classical planning system [23]. In its preprocessing phase, it builds a decision tree right after grounding and uses a so-called successor generator to evaluate which actions would be applicable in a given state (leaves contain those actions). This design choice dictates how much work the planning system will do upfront as opposed to during problem-solving (i.e., search). In FD, the total duration of calls to the successor generator function accounts for the lion's share of CPU work (i.e., almost 90% of total runtime) [24]. Additionally, the successor generator triggers "out of memory" for over a thousand problem instances because the precomputed structures reach a huge number of nodes. Thus, implementations such as FD's successor generator invest in heavy preprocessing and large auxiliary data structures. In contrast, a naive successor generator, which would store a list of grounded actions and iterate through them to check their applicability in a given state, would require minimal processing and memory upfront [24], but would incur expensive applicability checks for each search node as it needs to scan the entire list of grounded actions and this is repeated thousands if not millions of times. So, its CPU time scales with the number of expanded states, and so does energy. Thus, the choice of successor generator shifts energy consumption between an upfront, memory-dominated spike and a longer, CPU-dominated tail. Selecting or tuning mechanisms, such as the successor generator, is a direct lever for energy-aware AI planning optimisation.

Certain operations during problem-solving also incur computational overhead. Consider, for example, HTN planning, where a mechanism is required to keep alternative decomposition branches independent. In our own HTN planning system, SH, which works without preprocessing [25], this is achieved using state cloning. SH keeps an immutable lifted representation, which requires a full clone of the state right inside the main search algorithm. The benefit is that we keep a fully lifted representation, meaning memory grows only with discovered objects rather than all ground atoms, but the cost is paid at runtime in repeated hash-map allocations and copies of lists. Consequently, state cloning is likely one of the dominant CPU and heap contributors in SH, and thus an energy hotspot, even though we forgo the heavy upfront optimisations typical of grounded AI planning systems.

H 5 *The energy consumption of AI planning systems is determined by the choice of computational mechanisms and their distribution across phases of the AI planning pipeline.*

H 6 *Different phases of the AI planning pipeline exhibit distinct energy consumption characteristics, with some phases contributing disproportionately to total energy usage.*

The most informative heuristics in modern AI planners are computationally heavy. LM-cut, for example, is an admissible heuristic that iteratively finds "landmarks", which are actions that must appear in any plan [26]. For each state, it computes h^{max}, builds a justification graph, then repeatedly finds a cut of actions that must be used to reach the goal, assigns costs to these actions while maintaining admissibility, and reduces the action costs. This continues until no more cuts can be found. Pattern databases (PDBs) are precomputed lookup tables storing exact distances to the goal for abstracted

[2] A STRIPS representation is a restricted form of classical planning that only allows positive preconditions and simple add/delete effects on atomic propositions, without conditional effects, quantifiers, or complex logical expressions.

versions of a planning problem [27]. The abstraction projects away some state variables, creating a smaller state space that can be solved optimally offline. During the search, the planner looks up the abstract state corresponding to each concrete state to get an admissible heuristic value. So, LM-cut performs significant computation during search, while PDBs shift computation to preprocessing but require memory lookups during search. This exemplifies the trade-off between online computation versus offline computation and storage, with different implications for energy consumption.

H 7 *Total energy consumption per problem instance increases monotonically with the computational complexity of the primary heuristic function.*

Performance evaluation shows that, once search begins, heuristic computation often dominates internal runtime costs of planning systems. For instance, it was reported that the main bottleneck in the HSP and HSP 2.0 planners is the computation of the heuristic values, taking more than 80% of the total runtime in both planners [28]. Likewise, the LAMA planner evaluates heuristic values by using deferred heuristic computation, where states are not evaluated upon generation but upon expansion using the heuristic value of their parent rather than their own value. This seems to lead to a substantial reduction in the number of heuristic computations, at the cost of losing heuristic accuracy [29]. Such evidence motivates treating heuristic evaluation as the single largest internal consumer of processing time, and, by implication, a primary energy hotspot in heuristic-based AI planning.

H 8 *Energy consumed by heuristic evaluations accounts for the single largest share of internal energy consumption, exceeding the energy consumption of any other individual operation.*

Recent studies in sustainable computing have shown that the energy consumption of software can vary significantly based on implementation details beyond algorithmic choices. The same algorithm implemented in different programming languages can have energy consumption differences of up to 50 times [30]. Furthermore, code refactoring can have an impact on energy usage, from decreasing energy consumption by about 4.6% to increasing energy consumption by 7.5% [31]. In our context, AI planners often rely on highly optimised compiled languages, such as C++, or interpreted languages, such as Python. Minor changes in compiler or interpreter versions can cause performance variations due to differences in memory handling and CPU efficiency. For example, switching from GCC 4.7 to GCC 4.8 or from Python 2.7.3 to 2.7.10 changes the performance rankings of AI planners that participated in IPC 2014 [32]. It has been shown that identical source code solved up to 35% fewer problems or took noticeably longer when recompiled under a newer version of the toolchain. Looking at energy-centered evidence on compilers and interpreters, one can notice that the same program can consume up to 8% more total energy under Clang than under ICC, with GCC in between [33], and that the same application code can consume 8% more energy when using CPython 3.10 than CPython 3.12 [34]. This evidence shows the sensitivity of programs to software toolchain versions, which likely leads to measurable differences in energy consumption for AI planning systems.

H 9 *The energy consumption of AI planning systems varies when compiled or interpreted with different versions of the same software toolchain.*

AI planners are rarely invoked in a uniform manner in real-world settings. Applications exhibit a variety of invocation patterns, from one-shot planning for fairly static problems to multiple invocations in dynamic environments, with or without replanning functionality. For example, mobile robots navigating dynamic environments may need to replan whenever obstacles appear and autonomous vehicles must adapt to traffic conditions. In a smart restaurant coordination system using SH [35], the planner was invoked 360 times per weekday, with a notable burst of invocation on some days (over 600 invocations). This translates to a planning call every 1-2 minutes during operating hours. On weekends, the invocation rate dropped to only 31 calls per day, reflecting a much more sporadic pattern with long idle gaps between calls. In such invocation patterns, the timing of AI planner invocations interacts non-trivially with energy usage. Even if two systems invoke an AI planner 300 times per

day, one doing so every five minutes (sporadic) and another in three short bursts (bursty) will have notably different energy profiles. Sporadic usage may pay more per call due to reinitialisation and idle wake-ups, while bursty usage may incur hardware-level penalties like thermal throttling. These observations suggest that, beyond optimising the efficiency of individual planning calls, the invocation pattern is an independent variable in the energy profile of AI planning systems and plays a crucial role in determining cumulative energy consumption.

H 10 *The relationship between an AI planner's invocation frequency and cumulative energy consumption is linear due to inhibited power state transitions and repeated initialisation overhead, with continuous invocation consuming disproportionately more energy per planning episode than sporadic invocation.*

4. Energy Measurement Framework

As the ultimate objective of our work is to initiate energy accountability, we design a general framework for measuring the energy consumption of AI planning systems. The PLANning enERGY Measurement framework (PLANERGYM), which refines and extends the green software measurement model [36], is a structured approach that suggests starting by identifying goals and objects to be measured, followed by selecting appropriate metrics and a measurement procedure, and concluding with data evaluation. PLANERGYM assumes executing selected AI planners on multiple problem instances to ensure informative and significant energy consumption analysis. We first define necessary concepts and then explain the procedure and data evaluation.

4.1. Objects and Goals of Measurement

A *measured object* defines what is being assessed. In AI planning, the primary measured object is an AI planning system as a whole software product. This object could be further decomposed into other objects of measurement interest, that is, components and features that exhibit high resource load, such as those identified in our hypotheses.

A *measurement goal* defines what is being investigated. In our setting, the measurement goals focus on understanding relationships, (e.g., whether planning time and energy consumption are positively related (H1), variations (e.g., whether energy consumption varies across different domain model configurations (H3), and characteristics of energy consumption across these different system configurations and operational contexts (e.g., whether different phases exhibit distinct energy behaviour (H6). In all cases, comparison can be and should be a measurement goal, where an AI planning system is compared to itself or different AI planning systems to each other.

4.2. Measurements and Metrics

Measurement practices and metrics provide a foundational layer for energy accountability in AI planning. Therefore, we discuss each of these two components in detail.

Measurements refer to the data collection activities and processes used to gather information about the performance and resource consumption of AI planning systems (henceforth, we refer only to the primary measured object for simplicity). Reliable measurement requires several prerequisites, including suitable hardware, software logging tools, and a well-defined environmental setup. *Hardware requirements* specify the means of monitoring and measuring accurate energy and power data. These may include external instruments, such as standalone power meters, enterprise-grade distribution units, or internal interfaces integrated into computer hardware itself. For example, the Running Average Power Limit (RAPL) interface, available in Intel and some AMD processors, allows for precise monitoring of energy consumption at the level of CPU packages or cores [37]. Complementing hardware support, *software logging tools* can record resource usage and power consumption data. Tools, such as collectl, offer estimates of CPU and RAM. However, higher-precision measurement tools rely on access to hardware. For example, pyRAPL is a Python library that interfaces with RAPL to record energy data with high

temporal resolution, while CodeCarbon is a Python package that integrates multiple methods of energy measurements and also estimates carbon emissions, favoring hardware-level readings where available.

A well-defined *environmental setup* is essential to ensure that measurements are reproducible and interpretable. This includes defining a usage scenario, the notion of useful work, and a system baseline. A *usage scenario* specifies the experimental setting in which an AI planner is run. It includes the workload, invocation pattern, and measurement window. The *workload* defines the amount and nature of planning tasks assigned, which can vary along several dimensions, such as problem size (e.g., number of objects, actions, goals), encoding expressiveness (e.g., STRIPS, numeric features, durative actions), or task decomposition complexity in HTN planning. The *invocation pattern* is as described before, that is, how frequently an AI planner is called. Standard performance evaluations adopt a one-shot invocation, where an AI planner is invoked once per problem (this is different from experiment repetitions for statistical significance). However, in real-world applications, AI planners may be invoked sporadically, periodically, or in bursts. The *measurement window* defines the temporal boundaries for data collection and is closely related to the wall-clock time. It may be delimited by planner invocation and termination, by the time taken to find the first valid plan, or by timeouts. Also important is the definition of *useful work*, which captures the output or benefit delivered by running an AI planner. This is a critical component in characterising energy efficiency. Examples of useful work include the generation of a valid plan, plan quality, plan length, the number of planning problems solved, the number of invocations, etc. Finally, to isolate the energy consumption attributable to an AI planner itself, a *baseline* measurement is needed. The baseline represents the energy overhead introduced by the runtime environment in the absence of the measured objects and serves as a reference point to compute net energy usage.

Metrics are quantities that translate raw measurement data into interpretable indicators of energy usage, system efficiency, and performance bottlenecks. Common low-level metrics include the baseline consumption B (measured in watt-hours or joules), the runtime T (in seconds), CPU utilisation C (%), RAM usage R (in MB or %), and total energy consumed E, calculated by subtracting the baseline from the total power drawn during the measurement window. Mean power draw provides further insight into the intensity of resource usage and can be used to identify hotspots in planning code and phases, which can then be optimised.

Beyond these basic indicators, more advanced metrics offer a way to evaluate and compare the energy efficiency of AI planning systems. One such metric is the energy-efficiency factor, $EEF = \frac{W}{E}$, where W represents the useful work. A higher factor indicates greater energy efficiency. This metric can also be used to compare AI planners, domain/runtime configurations, or scenarios. For example, if the useful work means finding a valid plan and two planners can find such a plan for the same problem with different energy consumption, this factor would highlight the difference in their energy efficiency. In addition to EEF, a carbon-emission factor can be introduced to estimate the environmental impact of AI planning systems. It can be computed as the product of energy consumption and the carbon intensity of the power source, consistent with recent practice in reporting ML energy and carbon footprints [38]. This metric is especially relevant for sustainability-conscious deployment, as it contextualises energy efficiency in terms of real-world environmental cost.

4.3. Measurement Procedure

Having defined the key concepts, we now describe how to construct and execute a procedure in a valid, reproducible, and transparent manner. The procedure model we discuss encompasses the discussion from before and can therefore be easily aligned for testing the hypotheses. The procedure model should consist of five components, each discussed in the following.

Measured object operationalisation. The first component involves the selection and preparation of AI planning systems or components to be measured. The AI planning systems selected must support a common input format, such as PDDL, to ensure consistent workload execution. Moreover, the selected AI planning systems must support sufficient expressiveness to handle the domain models and problem instances included in the workload. In this context, executables, containers, and source code availability

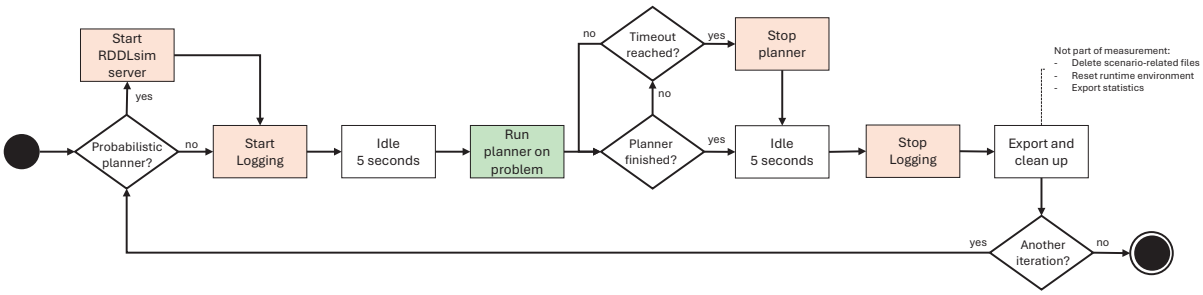

Figure 1: Execution Protocol for Collecting Data in a Measurement Scenario for Probabilistic AI Planners.

are critical for enabling the measurement. Where the goal is to attribute energy consumption to specific components, mechanisms, or phases, an AI planner must have source code available, expose a modular architecture, or be sufficiently documented to allow the isolation of these components. Finally, AI planners must be compatible with the hardware and software infrastructure used for running them and measuring their energy consumption.

Measurement method definition. Once AI planners or components have been selected, the next step is to define how measurements will be taken. One approach is to treat an AI planner as a single unit, and aggregate metrics are recorded. Another approach is to attribute resource consumption to specific components, mechanisms, or phases, requiring instrumentation or logging at a finer granularity. Independent of the chosen measurement approach, the number of repetitions per scenario must also be defined. The general recommendation is to have at least 30 repetitions to achieve statistical significance unless some constraints are in place [39], such as resource limitations or when the usage scenarios involve long measurement time (e.g., some types of planners may need to reach the predefined timeout to output a plan). Finally, wherever possible, the measurement runs should be automated to minimise measurement noise and ensure consistent execution conditions.

Scenario design. A measurement scenario defines the context in which an AI planner or component is evaluated. Each scenario must specify both its type and structure. Common types include the baseline, standard usage, where an AI planner solves a planning problem under normal conditions, and load, where an AI planner is subjected to stress, e.g., large planning problems. The measurement window plays a crucial role here as it specifies when logging begins and ends, typically, from the moment of planner invocation to the moment a plan is returned or a timeout occurs. Scenarios must therefore include an execution protocol, such as the one in Figure 1, which describes the sequence of operations, such as log initialisation, input preparation, planner invocation, planner termination, log termination, and cache flushing. Such an execution protocol must be consistently applied across repetitions and between AI planners.

Workload specification. Workload specification defines the tasks given to an AI planner and directly influences energy consumption footprints. A workload includes a set of planning problems, which are characterised by their domain features (e.g., expressiveness, object count, goal structure) and complexity. Workloads must be chosen to align with the capabilities of the selected AI planners and measurement goals, and by implication, the hypotheses under investigation. One can choose between benchmark and real-world planning problems. When using benchmarks, additional care is required. Problem instances in benchmarks are often generated using randomised generators with parameters that control problem size and difficulty. The domain chosen, the generator settings, and the distribution and number of problem instances all affect performance and energy profiles. Benchmark selection should also consider the type of planning (e.g., classical and HTN planning), the diversity of domain structures, the intended planning objective (e.g., satisficing, optimal), and different domain configurations.

Execution and post-processing. The final step in the measurement procedure involves executing the defined scenarios and analysing the resulting data. Execution must be automated where possible to ensure consistency and minimise external interference. Before analysis, baseline consumption must be subtracted to isolate planner-induced energy usage. The metrics can then be computed from the raw data. Statistical processing of the results, such as averaging across runs, identifying outliers, and computing confidence intervals, helps validate the reliability of findings. These outputs serve both as descriptive indicators and as evidence for testing the energy hypotheses.

4.4. Measurement Setup

The measurement setup specifies the physical and software environment in which energy data of AI planning systems is captured. It instantiates the previously defined measurement concepts and measurement procedure model by specifying the hardware, operating system, measurement tools, logging infrastructure, and runtime environments needed to execute AI planners under controlled and reproducible conditions.

The hardware platform should be selected to reflect both the experimental goals and practical constraints. Systems should have sufficient memory and CPU capacity to avoid unintended bottlenecks (e.g., swapping), but unnecessary background processes should be minimised to reduce measurement noise. To ensure accurate baseline and runtime measurements, the idle power characteristics of the hardware should be stable and well understood. The choice of operating system affects idle power draw. This choice is often limited by the requirements of selected AI planners, as most of them are optimised to run on Linux distributions. In any case, the operating system should be configured to minimise background activity during measurements.

AI planners depend on complex software stacks, including specific programming language runtimes, solver libraries, or domain model interpreters. To standardise these dependencies and ensure reproducibility, it is recommended that AI planners are encapsulated in containers. Containers also make it easier to configure consistent CPU and memory limits across AI planners, avoiding variability introduced by heterogeneous runtime configurations.

Logging infrastructure typically includes scripting for orchestrating AI planner runs, recording timestamps, and collecting low-level metrics. In other fields, bash or Python scripts are commonly used to coordinate measurements and enforce repeatability. In some scenarios, additional runtime services may be required. For example, probabilistic AI planners based on relational Markov decision processes (RMDPs) may need to interact with simulation servers (e.g., RDDLSim) to evaluate policy performance. In such cases, these external services must be included in the measurement setup (see Figure 1) and properly accounted for in the baseline.

4.5. Data Evaluation

The data evaluation model defines the methods used to analyse recorded measurement data from AI planners and translate raw logs into interpretable insights. It should align with the measurement goals and usage scenarios, and should support both descriptive and comparative analysis of energy consumption and performance. Once data is collected, the first step is to compute the defined low-level and advanced metrics per measurement instance.

For summary statistics, mean values and standard deviations are calculated across repetitions for each planner, problem instance, and scenario. These statistics help identify consistent energy patterns, highlight energy consumption variability, and ensure statistical robustness. In comparative contexts, inferential statistics should be used to assess whether observed energy differences between AI planners are statistically significant. To facilitate interpretation, results should be visualised through plots, such as boxplots (to represent distribution), bar charts (to compare aggregate measures), and tables (e.g., for detailed planner-by-domain comparisons). These visual tools support both AI planner comparisons and component evaluations (e.g., per phase, per domain, per mechanism). Special attention should be given to component-level or resource-specific analysis, where available. For instance, the contribution of RAM

energy usage relative to CPU power draw can be analysed to understand memory-related inefficiencies. Similarly, analysing energy expenditure across planner phases can help localise optimisation targets.

Finally, evaluation tools and automation scripts should be used to ensure consistency, especially when processing large numbers of runs of AI planners. Whether implemented using scientific computing environments or domain-specific tools, automated analysis ensures scalability and reduces human error.

5. A Case Study in Energy Profiling Classical Planners

This case study presents an illustrative instance of PLANERGYM. Its purpose is not to offer comprehensive performance rankings but rather to demonstrate how energy accountability in AI planning can be approached.

5.1. Energy Measurement

The measurement goal of the case study is to provide the first empirical insights into the relationship between computation time and energy use in classical planners. Thus, it targets hypothesis H1. The measured objects are full classical AI planners selected from the IPC2018 Agile Track.[3] This track focuses on planners that prioritise rapid plan generation over optimality. To reflect a spectrum of runtime performance, three planners were selected: LAPKT-BFWS-Preference [40], which ranked first in the track; Cerberus [41], ranked ninth; and FS-blind [40], ranked thirteenth. These choices were made to explore variation in energy usage across planners with different runtime performance profiles. Each planner is treated as a black-box entity, meaning that internal components are not considered.

The experiments were conducted on a laptop machine with an Intel Core i7-4770 processor clocked at 3.40GHz, equipped with 4 physical cores, 8 threads, and 24 GB of RAM. Each planner execution was constrained to a single core and allowed access to the full RAM, ensuring uniform resource availability. Energy measurements were obtained using Intel's RAPL interface, accessed through the pyRAPL Python library. Further, we used the psutil library to manipulate files, and Python's standard time module to control the measurement window and timeouts. We use bash scripts to build and run the planners.

The environmental setup includes a Ubuntu Desktop version 22.04 was used as the base operating system. To isolate the AI planners and manage their dependencies, each planner was executed within its own Singularity container, built using (updated) Singularity files provided by IPC 2018.

We collected low-level metrics, including a planner's runtime, CPU energy usage, and RAM energy usage. These were collected at a sampling rate of approximately once per second. No derived metrics were computed in this stage, keeping the focus on basic metrics and the initial hypothesis.

The measurement procedure followed a scenario where each planner was run on individual problem instances from three benchmark domains featured in IPC2018: Data Network, Nurikabe, and Snake. For each domain, the first ten instances from the competition repository were selected to construct the workload. The measurement window for each run consisted of an idle phase before planner invocation, followed by planner execution (with a timeout of 350 seconds), and concluded with an additional idle phase. The metrics were logged continuously throughout this window. The system's baseline energy consumption was measured separately and amounts to 6.7 joules. However, it was not integrated into the current visualisations as we decided to represent the energy consumption per component (CPU and RAM) and not as a total. Each planner-domain-instance combination was executed only once, which is a limitation acknowledged in this initial exploration. However, given the exploratory and demonstrative nature of the case study, single-run measurements were deemed acceptable to illustrate how the framework can be applied and where variability might arise.

Results were collected in tabular form. Data analysis was conducted using Jupyter Notebook, where recorded logs were parsed and visualised using standard plotting libraries (i.e., matplotlib, numpy, and pandas). The output includes per-instance runtime, CPU energy, and RAM energy, while our discussion

[3] https://ipc2018-classical.bitbucket.io/

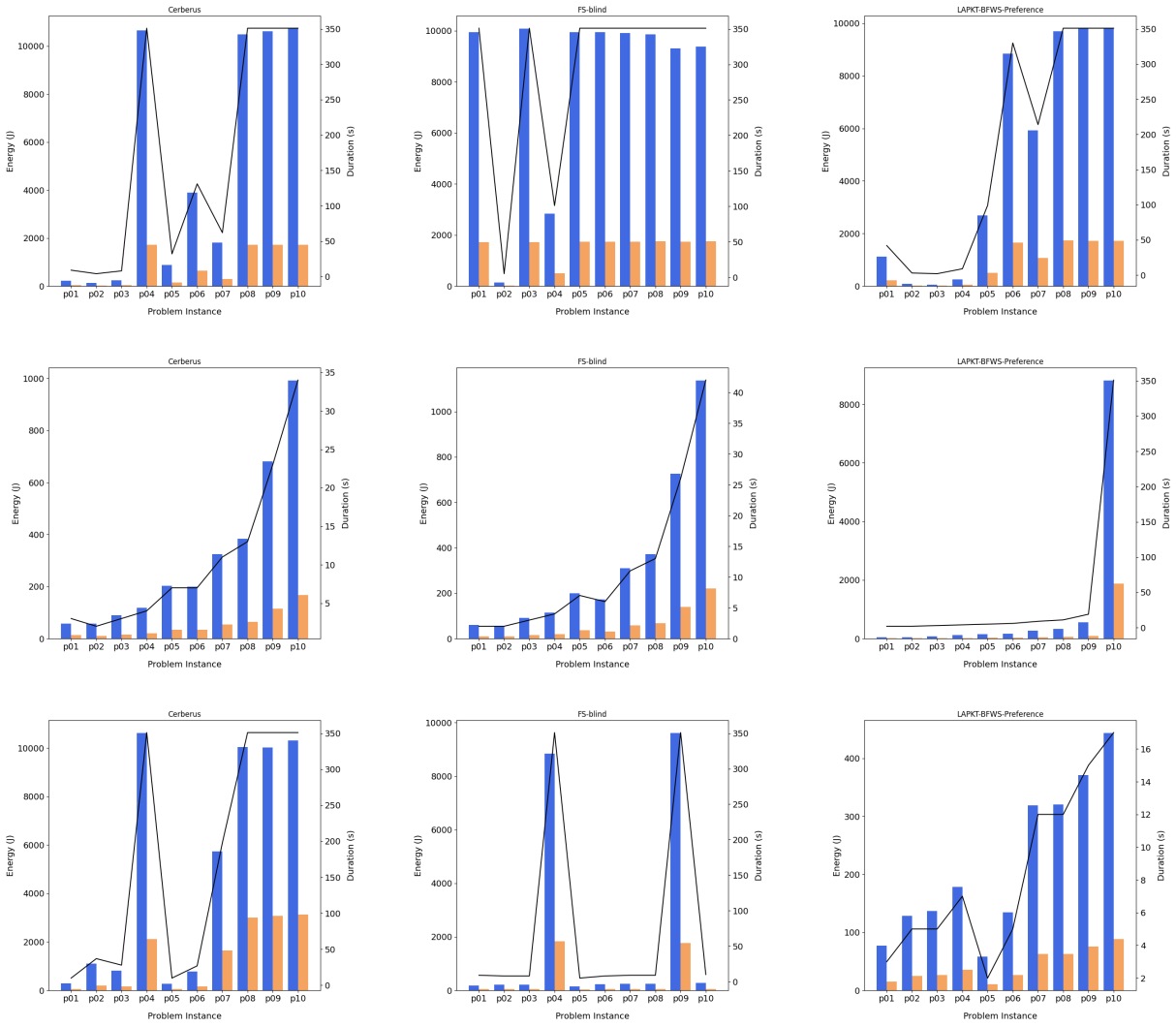

Figure 2: Performance Results of Selected Classical Planners in Three Benchmark Domains (Top to Bottom Row: Data Network, Nurikabe, SNake

also includes power, enabling a first comparison of computational and energy profiles across the selected classical planners.

5.2. Results

Figure 2 shows the visualisation of the metrics per planner per domain. In the following, we interpret these results.

Data Network Domain. Across the ten instances in the data network domain, every planner maintained an almost constant power between 33 and 35 W. Consequently, the principal determinant of energy was runtime. LAPKT-BFWS-Preference solved seven instances in less than 100 s, incurring about 3 kJ, but timed out in the remaining three, each of which cost almost 11 kJ. Cerberus displayed the same pattern: the solutions to the first problem instances consumed less than 0.3 kJ, whereas four time-outs also reached almost 11 kJ. FS-blind, whose blind breadth-first search rarely terminates early, reached the timeout on nearly every problem instance; its energy usage therefore clustered around 11 kJ irrespective of eventual success. In this domain, energy differences arose almost exclusively from differences in termination time, not from variations in instantaneous power draw.

Nurikable Domain. Nurikabe proved easier to solve. Cerberus and FS-blind completed every instance, while LAPKT-BFWS-Preference failed only on the last problem instance. The power again lay in the 30-35 W band, so energy scaled linearly with runtime. LAPKT-BFWS-Preference required at most 19 s on the first nine tasks (less than 0.7 kJ each) but exhausted the window on the last problem instance, spending about 10.7 kJ. Cerberus, typically 3–5 s slower than LAPKT-BFWS-Preference, consumed approximately 20% more energy per instance, with its longest run (34 s) costing about 1.2 kJ. FS-blind added a further few seconds per task and thus recorded the highest cumulative energy. Although the planners' power profiles were similar, modest runtime differences translate directly into systematic energy gaps.

Snake Domain. Snake constituted the most demanding workload. LAPKT-BFWS-Preference solved nine problems in less than 17 s (costing about 0.7 kJ) and never approached the timeout; its per-run power remained approximately 31 W. Cerberus solved six instances quickly (less than 198 s, less than 5.9 kJ) but timed out on four, each failure costing 12–13 kJ. FS-blind's behaviour was analogous: seven short successes (about 230-340 J) contrasted with two exhaustive failures (almost 10.7 kJ each). Thus, under the most difficult instances, single time-outs dominated the total energy budget for Cerberus and FS-blind, whereas LAPKT-BFWS-Preference's informed search avoided such spikes.

Cross-Domain Insights. In all three domains, the planners operated at near-constant power; RAM contributed only a minor, proportional increment. Differences among planners were driven by how rapidly they produced a plan or timed out. LAPKT-BFWS-Preference was energy-minimal because it typically finds a plan quickly; Cerberus was intermediate, and FS-blind incurred the highest cost owing to frequent near-timeout executions. Equal runtimes do not imply equal energy across planners, nor do equal energy implies comparable runtimes. This hints at the role of search strategy in impacting the energy use.

Implications for Hypothesis H1 The empirical evidence supports Hypothesis 1. Within a selected classical planner, energy and runtime are indeed proportional; however, runtime alone is an unreliable predictor of energy across the selected planners. A twenty-second LAPKT-BFWS-Preference run (less than 1 kJ) is cheaper than a twenty-second Cerberus or FS-blind run, and a single timeout (in other words, failure) can outweigh the energy cost of many rapid successes. Thus, energy efficiency in agile planning may be governed less by reducing instantaneous power, which remains essentially constant, and probably more by internal components, such as the search strategy and termination behaviour.

6. Concluding Remarks

We initiate a shift toward energy-aware AI planning by proposing a set of hypotheses and a dedicated measurement framework to guide future research. These hypotheses serve not only as a conceptual lens but also as a roadmap for systematic investigation of energy implications of algorithmic design choices, heuristic strategies, domain mode configurations, and system design. Their empirical exploration may uncover hotspots in planning systems and inform the development of energy-aware techniques that do not compromise performance. By defining how energy can be measured, attributed, and assessed in AI planning systems, we contribute the necessary framework for systematically accounting for energy and exploring energy-performance trade-offs. This departs from the traditional practice in planning research, where the focus is on runtime and plan quality, as exemplified by the IPCs, and brings energy accountability as a first-class evaluation concern. The illustrative case study shows the feasibility of this approach. While a general correlation between runtime and energy consumption was expected, the case study reveals deviations: classical planners with similar runtimes may exhibit different energy profiles. This observation indicates that runtime alone may not be a reliable proxy for energy use in AI planning. While the case study comes with several limitations that hinder the generalisability of the findings, it points to key challenges and considerations that future studies should address.

We suggest that IPCs consider expanding their scope to include tracks focused on energy and carbon efficiency. Such an initiative could catalyse the development of sustainable AI planning systems, foster methodological contributions, and raise community awareness of energy as a critical dimension of algorithmic performance. More generally, we encourage researchers and practitioners to adopt, adapt, and extend the proposed framework and hypotheses to promote empirical rigor, transparency, and sustainability in AI planning research and practice.

Acknowledgments

I thank Andreas Glinka for conducting the experiment with the selected classical planners.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT 4o to polish self-authored text. Afterward, the author reviewed and edited the content as needed and takes full responsibility for the work's content.

References

[1] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, A. Friday, The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations, Patterns 2 (2021).
[2] T. Anderson, A. Belay, M. Chowdhury, A. Cidon, I. Zhang, Treehouse: A case for carbon-aware datacenter software, SIGENERGY Energy Inform. Rev. 3 (2023) 64–70.
[3] C. Calero, M. Piattini, Introduction to green in software engineering, Springer, 2015.
[4] A. Pazienza, G. Baselli, D. C. Vinci, M. V. Trussoni, A holistic approach to environmentally sustainable computing, Innov. Syst. Softw. Eng. 20 (2024) 347–371.
[5] R. Schwartz, J. Dodge, N. A. Smith, O. Etzioni, Green AI, Comm. of the ACM 63 (2020) 54–63.
[6] R. Verdecchia, J. Sallou, L. Cruz, A systematic review of Green AI, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 13 (2023) e1507.
[7] A. E. Howe, E. Dahlman, A critical assessment of benchmark comparison in planning, Journal of Artificial Intelligence Research 17 (2002) 1–33.
[8] M. Aiello, I. Georgievski, Introduction to AI Planning, Technical Report 2412.11642, arXiv, 2024.
[9] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (2001) 5–33.
[10] J. Porteous, L. Sebastia, J. Hoffmann, On the extraction, ordering, and usage of landmarks in planning, in: European Conference on Planning, 2001, pp. 37–48.
[11] M. L. Littman, J. Goldsmith, M. Mundhenk, The computational complexity of probabilistic planning, Journal of Artificial Intelligence Research 9 (1998) 1–36.
[12] I. Georgievski, M. Aiello, HTN planning: Overview, comparison, and beyond, Artificial Intelligence 222 (2015) 124–156.
[13] E. Alnazer, I. Georgievski, M. Aiello, Risk Awareness in HTN Planning, Technical Report 2204.10669, arXiv, 2022.
[14] T. L. McCluskey, T. S. Vaquero, M. Vallati, Engineering Knowledge for Automated Planning: Towards a Notion of Quality, in: Knowledge Capture Conference, 2017.
[15] K. L. Myers, CPEF: A Continuous Planning and Execution Framework, AI Magazine 20 (1999) 63–69.
[16] F. André, E. Daubert, G. Nain, B. Morin, O. Barais, F4Plan: An Approach to build Efficient Adaptation Plans, in: International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, 2010, pp. 386–392.
[17] S. Fratini, N. Policella, A. Donati, A service oriented approach for the interoperability of space mission planning systems, in: Workshop on Knowledge Engineering for Planning and Scheduling, 2013, pp. 39–43.

[18] I. Georgievski, PlanX: A Toolbox for Building and Integrating AI Planning Systems, in: IEEE International Conference on Service-Oriented System Engineering, 2023, pp. 130–134.
[19] T. Bylander, The computational complexity of propositional strips planning, Artif. Intell. 69 (1994) 165–204.
[20] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, V. Boeva, How to measure energy consumption in machine learning algorithms, in: International Workshop on Energy Efficient Data Mining and Knowledge Discovery at ECML and PPKDD, 2018, pp. 243–255.
[21] M. Vallati, F. Hutter, L. Chrpa, T. L. McCluskey, On the effective configuration of planning domain models, in: International Conference on Artificial Intelligence, 2015, pp. 1704–1711.
[22] S. Thiébaux, J. Hoffmann, B. Nebel, In defense of PDDL axioms, Artif. Intell. 168 (2005) 38–69.
[23] M. Helmert, The fast downward planning system, J. Artif. Int. Res. 26 (2006) 191–246.
[24] Y. Zutter, Implementing and Evaluating Successor Generators in the Fast Downward Planning System, Master's thesis, University of Basel, 2020.
[25] I. Georgievski, A. V. Palghadmal, E. Alnazer, M. Aiello, SH: Service-oriented HTN Planning system for real-world domains, SoftwareX 27 (2024) 101779.
[26] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: what's the difference anyway?, in: International Conference on Automated Planning and Scheduling, 2009, pp. 162–169.
[27] S. Edelkamp, Symbolic pattern databases in heuristic search planning., in: International Conference on Artificial Intelligence Planning Systems, 2002, pp. 274–283.
[28] B. Bonet, H. Geffner, Planning as heuristic search, Artif. Intell. 129 (2001) 5–33.
[29] S. Richter, M. Westphal, The LAMA planner: guiding cost-based anytime planning with landmarks, J. Artif. Int. Res. 39 (2010) 127–177.
[30] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. a. P. Fernandes, J. a. Saraiva, Energy efficiency across programming languages: how do energy, time, and memory relate?, in: ACM SIGPLAN International Conference on Software Language Engineering, 2017, pp. 256–267.
[31] C. Sahin, L. Pollock, J. Clause, How do code refactorings affect energy usage?, in: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2014, pp. 1–10.
[32] C. Fawcett, M. Vallati, A. E. Gerevini, H. H. Hoos, Performance Robustness of AI Planners to Changes in Software Environment, Technical Report, Scispace, 2019.
[33] N. Schmitt, J. Bucek, K.-D. Lange, S. Kounev, Energy Efficiency Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite, in: Companion of the ACM/SPEC International Conference on Performance Engineering, 2020, pp. 38–41.
[34] R.-H. Pfeiffer, On the Energy Consumption of CPython, in: International Conference on the Quality of Information and Communications Technology, Springer, 2024, pp. 194–209.
[35] I. Georgievski, T. A. Nguyen, F. Nizamic, B. Setz, A. Lazovik, M. Aiello, Planning meets activity recognition: Service coordination for intelligent buildings, Pervasive and Mobile Computing 38 (2017) 110–139.
[36] A. Guldner, R. Bender, C. Calero, G. S. Fernando, M. Funke, J. Gröger et al., Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM), Future Gener. Comput. Syst. 155 (2024) 402–418.
[37] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, Z. Ou, RAPL in Action: Experiences in Using RAPL for Power Measurements, ACM Trans. Model. Perform. Eval. Comput. Syst. 3 (2018).
[38] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, J. Pineau, Towards the systematic reporting of the energy and carbon footprints of machine learning, JMLR 21 (2020) 1–43.
[39] E. Kern, L. M. Hilty, A. Guldner, Y. V. Maksimov, A. Filler, J. Gröger, S. Naumann, Sustainable software products—towards assessment criteria for resource and energy efficiency, Future Gener. Comput. Syst. 86 (2018) 199–210.
[40] G. Frances, H. Geffner, N. Lipovetzky, M. Ramiréz, Best-first width search in the IPC 2018: Complete, simulated, and polynomial variants, in: IPC 2018 – Classical Tracks, 2018, pp. 23–27.
[41] M. Katz, Cerberus: Red-black heuristic for planning tasks with conditional effects meets novelty heuristic and enchanced mutex detection, in: IPC 2018 – Classical Tracks, 2018, pp. 47–51.