LLM-Assisted Parameter Optimization for Low-Power Wireless Protocols

Mohamed Hassaan Mohamed Hydher, Markus Schuß, Olga Saukh, Kay Römer, and Carlo Alberto Boano
Institute of Technical Informatics
Graz University of Technology
Graz, Austria
{m.h.mohamedhyder,markus.schus,saukh,roemer,cboano}@tugraz.at

Abstract

Careful parametrization of low-power wireless (LPW) protocols is essential to meet the stringent requirements of many IoT applications, yet remains notoriously difficult: tiny changes in configuration parameters can have significant and sometimes counterintuitive effects on system performance, and exploring these effects through real-world experiments can be prohibitively costly. State-of-the-art protocol optimization frameworks can minimize the amount of required experimental trials, but are constrained by rule-based test-point selection and lack the semantic understanding to interpret more complex experimental contexts. In this work, we explore the use of Large Language Models (LLMs) to build a reasoning-driven optimization framework capable of intelligently navigating a protocol's parameter space. After introducing LAPO, a two-stage reasoning architecture, we study the extent of guidance required for reliable decision-making (e.g., by means of refined system prompts), as well as the benefits of providing code-level access to the protocol's implementation. Our systematic evaluation shows that cloud-hosted proprietary models (e.g., OpenAI's GPT-5 and Anthropic's Claude Sonnet 4.5) can outperform traditional Bayesian optimization strategies in terms of faster convergence to the best-performing parameter set, especially when guided to use median-based decision making to account for the stochasticity of wireless experiments. We further evaluate the performance of locally-hosted open-weight models (e.g., Meta's Llama 3.1 and Google's Gemma 3) across model sizes spanning 8–70 billion parameters. Our results show that also these LLMs, which avoid API usage costs, can quickly converge to the best parameter set, especially at higher parameter counts. Finally, we showcase how the proposed LLM-assisted parametrization framework converges on average to the optimal parameter set of a popular LPW protocol within seven experimental trials on a public testbed across diverse network topologies and real-world conditions.

CCS Concepts

• **Computer systems organization** → *Embedded and cyber-physical systems*; *Sensor networks*; • **Networks** → *Network protocols*; *Network performance evaluation*; *Network performance modeling*; *Network performance analysis*; • **Computing methodologies** → *Natural language processing*.

Keywords

Low-power wireless, Protocol parametrization, Parameter optimization, Large language models, Testbed evaluation, IoT.

1 Introduction

Low-power wireless (LPW) systems have become a cornerstone of the IoT, enabling the development of key applications in the fields of smart homes and cities, precision agriculture, as well as industrial automation. Careful parametrization of the employed communication protocols is *crucial* for maximizing system performance and meeting stringent application requirements [6], e.g., on reliability, latency, and energy consumption, but is inherently complex.

Challenges in parametrizing LPW protocols. Identifying a suitable set of protocol parameters for a given application is indeed far from straightforward, and often requires a deep understanding of the protocol behaviour, the network topology and stack, as well as the nature and volume of transmitted data. For example, one would expect LPW protocols to exhibit a decrease in energy consumption when using a lower transmission (TX) power. In practice, the use of a lower TX power may worsen the quality of links across the network, increasing the chances of missing fundamental information such as synchronization beacons [10, 11], and resulting in a longer radio active time and higher energy consumption. This example exposes non-linear and unintuitive interactions between configuration parameters (e.g., TX power) and optimization objectives (e.g., energy consumption), which are difficult to anticipate without a profound understanding of the protocol's internals. Unfortunately, even expert knowledge—alone—is insufficient to guarantee accurate parametrization. Protocol performance is, in fact, often affected by environmental conditions and hardware features [23]: verifying whether application requirements are met and quantitatively comparing different parameter sets thus requires *real-world testing*. LPW *testbeds* allow evaluating protocol performance on actual hardware, capturing real-world complexity far more accurately than simulation [3]. However, they are shared resources and usually allow only short experimentation windows per day [1, 28]. Because of this, exploring all parameter combinations or performing an exploration giving a sufficient statistical significance may not be feasible: it is thus essential to find techniques able to identify an effective parameter set with *as few testbed trials as possible*. This is even more important in light of the *noisiness of wireless experiments*, which are inherently stochastic due to hardly-controllable factors such as external RF interference and multipath fading. As a result, a single exhaustive sweep of the parameter space is insufficient, as it provides only a performance snapshot. The latter can be heavily influenced by transient conditions (e.g., a "lucky" run without RF interference), and therefore lack statistical significance.

Limitations of existing approaches. Existing studies dealing with the parametrization of LPW protocols have rarely considered

the number of real-world experiments required, focusing instead on characterizing performance trade-offs [4, 7], optimizing parameters for specific scenarios and wireless technologies [5, 8, 14, 31], enabling runtime adaptation on-device [38], and on applying learning-based approaches for automated parametrization [13, 15]. A notable exception is APEX [10], a recently-introduced framework enabling an automatic parameter exploration of LPW protocols that is explicitly designed to minimize the number of required testbed trials. By using Gaussian Processes (GPs) to model protocol performance and by selecting testpoints that balance exploration and exploitation, APEX achieves efficient and robust parameter search even under noisy experimental conditions. However, its behaviour is constrained by predefined acquisition strategies: even though these allow for specific rule-based adjustments (e.g., confidence thresholds), they cannot cover every environmental contingency. APEX thus lacks the expert-like reasoning required to autonomously diagnose complex contexts and dynamically select an appropriate strategy when standard rules fall short.

The potential of LLMs. Large language models (LLMs) offer a new dimension of flexibility by introducing reasoning and contextual understanding into the optimization process. They have the potential to interpret experimental outcomes, adjust their decision-making strategy, and refine parameter choices in a manner similar to how an expert would guide an optimization procedure, considering both observed trends and application requirements. This capability opens the door to more adaptive and generalizable parametrization, where optimization is no longer confined to fixed rules or model structures, but instead guided by reasoning that can evolve with the problem context. This could allow a system to navigate the search space more intelligently, and identify optimal configurations with a lower number of experimental trials.

Our contributions. In this paper, we investigate how LLMs can assist in the parametrization of LPW protocols, and derive practical insights on how to design a reasoning-driven optimization framework that efficiently explores the protocol parameter space even when only few experimental trials are possible and their outcomes are noisy. After introducing related work in § 2, we make the following contributions:

- We design LAPO, a two-stage LLM-based reasoning architecture for LPW protocol parametrization, and describe its practical instantiation as a modular reasoning pipeline (§ 3).
- We introduce an experimental methodology enabling a controlled and robust evaluation of LLM-driven parameter optimization under noisy real-world feedback (§ 4).
- We evaluate the use of *cloud-hosted proprietary LLMs* in LAPO's reasoning pipeline and study the extent of guidance required for reliable decision-making (§ 5).
- Motivated by deployment needs (e.g., API costs, customizability, privacy), we evaluate the use of *locally-hosted open-weight LLMs* in the proposed reasoning pipeline with model sizes spanning 8–70 billion parameters, showing that they quickly converge to the best parameter set, especially at higher parameter counts (§ 6).
- We validate the proposed closed-loop optimization framework by showcasing LAPO in action on a public LPW testbed across diverse network topologies and real-world conditions, evaluating its convergence behaviour (§ 7).

2 Background and Related Work

This section reviews prior work relevant to our study. We begin with existing works focusing on the parametrization of LPW protocols (§ 2.1), and we then examine recent efforts that use LLMs for both optimization and reasoning tasks across wireless systems and other related domains (§ 2.2).

2.1 Parametrization of LPW Protocols

Research on LPW protocol parametrization has evolved in the last decade from manual trade-off characterizations to the design of automated optimization frameworks.

Early studies focused on analysing the impact of individual parameters through measurement studies in specific settings [4, 5, 22]. These efforts targeted specific protocols such as Bluetooth [16, 31], ZigBee [8], and LoRa [14], providing valuable insights into performance trade-offs but often lacking generality across complex multi-parameter interactions.

Subsequent efforts introduced joint multi-parameter exploration [7] and adaptive learning-based techniques, employing genetic algorithms [20] or on-device reinforcement learning [13, 15] to automate parameter selection. Additionally, modular frameworks have been proposed to facilitate runtime parameter adaptation [38] or to leverage simulation and hardware models to automate configuration [21, 23]. However, these approaches require extensive experimentation, rely on manual model definitions, or focus on device-level adaptation rather than holistic network performance.

In this context, APEX [10] represents the state-of-the-art for automated parameter optimization. Unlike other solutions, APEX is explicitly designed to identify optimal network-wide configurations with a minimal number of experimental trials. For this reason, we use it as a *baseline* in this work.

APEX. Proposed by Hydher et al. [10], APEX is an open-source framework that automates the optimization of LPW protocol parameters while reducing the number of experimental trials needed to inform parameter exploration. It follows a four-step iterative optimization loop: (i) fitting a model, (ii) choosing the next test-point, (iii) running testbed trials, and (iv) terminating once certain criteria are met. The optimization process begins with the user providing the protocol and the parameters to be optimized as input, together with the application requirements to be satisfied. The latter are expressed as an optimization goal subject to specific constraints:

$$\text{maximize/minimize [metric(s)]} \quad \text{s.t. constraint(s) [metric]}. \tag{1}$$

For example, an application requiring to minimize energy consumption (E_c) while maintaining a Packet Reception Ratio (PRR) above a certain threshold can be defined as:

$$\text{minimize } E_c \quad \text{s.t. } PRR \geq 96\%. \tag{2}$$

In APEX, these performance metrics are correlated with protocol parameters using Gaussian Processes (GPs). A GP model captures non-linear behaviours and quantifies uncertainty from a limited number of observations, making it ideal for optimization involving costly experiments[1]. After each trial, APEX updates the GP model to

[1]Real-world trials are often costly in terms of both manpower and time. Similarly, experimentation on public testbeds is typically restricted due to their shared nature, leaving only a limited number of runs available per day.

find the mean performance and its variance for all possible parameter sets. This helps the framework to find not only configurations that are likely to work well (*exploitation*), but also areas where there is not enough information (*exploration*). APEX uses acquisition functions like the Lower Confidence Bound (LCB) and the Expected Improvement (EI) to balance exploration and exploitation and choose which set of parameters to test next. The process continues in cycles until certain predefined termination conditions are met, e.g., reaching a certain level of confidence or a limit in the number of testbed trials that can be performed. APEX also calculates confidence metrics capturing both the "optimality" (i.e., the likelihood that the current-best configuration is optimal) and the "robustness" (i.e., how well constraints are satisfied) of the solution.

With this structure, APEX enables systematic and efficient optimization, maintaining robustness even when experimental feedback is noisy[2]. Empirical evaluations on a real-world public testbed have demonstrated its ability to identify optimal parameter sets for LPW protocols such as Crystal [11] and RPL [34] with significantly fewer trials than exhaustive or heuristic methods. Consequently, APEX represents the state-of-the-art in model-based optimization for LPW systems, and will serve as *baseline* against which we evaluate the performance of LLM-based reasoning-driven approaches.

2.2 LLMs for Optimization and Reasoning

Recent progress in LLMs has opened a new direction in which optimization can be guided by reasoning rather than fixed acquisition strategies. LLMs are different from statistical optimizers in that they provide strong contextual understanding and adaptive reasoning, enabling them to interpret performance trends, identify trade-offs, and make informed decisions from structured or natural-language feedback.

General optimization. Recent studies have applied LLMs for general optimization tasks. For example, Liu et al. [18] have incorporated LLMs into Bayesian optimization pipelines, demonstrating that reasoning-guided in-context learning can enhance candidate selection. However, their framework does not account for noise in analytical evaluations and lacks constraint handling. Similarly, Peng et al. [24] and Zhou et al. [37] utilize LLMs to solve non-convex optimization problems or optimize power control through symbolic changes. Yet, these methods largely depend on analytical formulations or penalty-based reward structures that are unsuitable for noisy experiments where outliers can mislabel valid configurations, as often observed in wireless settings.

LLMs for wireless communication. Beyond general optimization, significant attention has been directed toward applying LLMs specifically to wireless networks. Sevim et al. [29] have investigated the use of reinforcement learning to deploy LLMs in wireless settings, finding that they converge faster than CNN-based agents. Thind et al. [32] have introduced OptimAI to convert natural-language optimization descriptions into solvable code, whereas Shao et al. [30] have proposed WirelessLLM to align general-purpose models with domain-specific wireless knowledge. Other works have explored environment classification through modular expert models [19] and common trade-offs in intermittent computing systems [26]. LLMs have also been applied to the runtime orchestration of secure wireless sensor networks [2] and to the prediction of channel states in satellite downlinks [35]. However, all these studies primarily target real-time control directives, code synthesis, or predictive modelling. None of them specifically addresses the *automated parameter optimization of LPW protocols*, which requires navigating complex, non-linear parameter interactions under strict application requirements and limited experimental budgets.

Summary and positioning. In summary, existing research highlights the growing potential of LLMs in optimization, reasoning, and wireless-system intelligence. However, most methods depend on analytical formulations, deterministic simulations, or unconstrained objective settings, none of which hold in our problem. At the same time, while APEX represents the state-of-the-art in automated LPW protocol parametrization, its behaviour is constrained by predefined acquisition strategies that cannot adapt when standard rules fall short. Our approach addresses both gaps: it operates directly under noisy real-world feedback, explicit application constraints, and a limited experimental budget, while replacing fixed acquisition rules with reasoning-driven flexibility that can interpret complex experimental contexts and adjust its strategy accordingly.

3 LLM-Assisted Parameter Optimization

We propose LAPO (**L**LM-**A**ssisted **P**arameter **O**ptimization), a framework to identify parameter configurations satisfying given application requirements under limited experimental budgets. LAPO uses LLMs to (i) extract protocol knowledge from static artifacts and to (ii) iteratively propose parameter settings by reasoning over stochastic experimental feedback.

3.1 Design Rationale and Architecture

The design of an LLM-based system for LPW protocol parameter optimization is governed by two requirements: (i) extracting the protocol *context*, i.e., establishing a mechanistic understanding of the target protocol (parameter semantics, plausible interactions, and protocol-specific pitfalls) from static sources such as documentation and implementation code; and (ii) active *optimization*, i.e., proposing new parameter configurations by reasoning over an evolving history of (possibly noisy) experimental observations.

Architectural choice: dual-stage reasoning. A straightforward approach is to employ a *single monolithic agent* to handle the entire workflow—analyzing the artifacts (e.g., source code or documentation), interpreting experimental results, and proposing new parameter configurations within a single context window. However, this design is suboptimal for two reasons. First, it combines two qualitatively different reasoning modes: the mechanistic understanding from static artifacts, and the iterative reasoning over noisy numerical feedback. Prior work suggests that separating such roles/phases improves reliability compared to a single undifferentiated agent [9]. Second, as the experimental history grows, keeping verbose static artifacts interleaved with the optimization trace increases *token overhead*, which not only raises costs and latency but also occupies a significant portion of the context window, leaving less capacity

[2]This is commonly the case with wireless experiments: to account for real-world uncertainties, it is hence crucial to increase the statistical robustness of the results—e.g., by testing a given parameter combination multiple times.

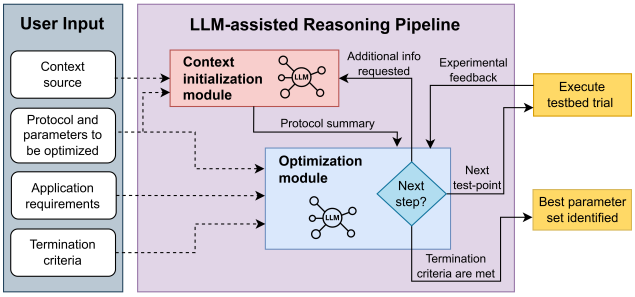

Figure 1: High-level architecture of the proposed LLM-assisted reasoning pipeline for LPW parametrization.

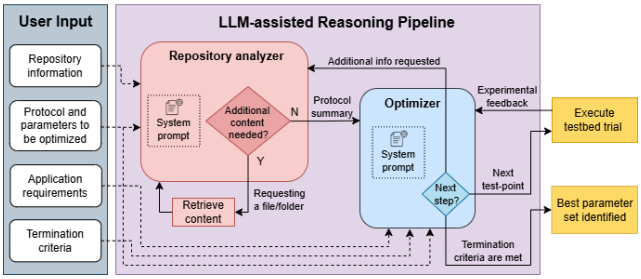

Figure 2: Instantiation of LAPO 's dual-stage architecture. The repository analyzer produces a protocol summary, which the optimizer uses—together with previous experimental feedback and the provided ARs—to propose the next parameter set (test-point) to be evaluated experimentally.

for reasoning over recent observations and thereby weakening focus on decision-relevant information, consistent with known long-context failure modes [17]. Complementary evidence on reasoning distraction suggests that mixing the main task with competing reasoning-heavy prompt content can substantially degrade model performance and induce strong recency effects, motivating designs that keep the optimization trace focused and avoid interleaving verbose static context during iterative decision-making [36].

Accordingly, we design LAPO to adopt a two-stage architecture, depicted in Fig. 1. It comprises: (i) a *context initialization module*[3] that extracts and compresses protocol knowledge from static artifacts into a *protocol summary* (e.g., parameter semantics and potential interactions), and (ii) an *optimization module* that consumes this summary together with the experimental observations to drive the next test-point selection (i.e., the next parameter set to be evaluated experimentally).

Required inputs. To operate as a well-posed optimization engine, LAPO requires the following inputs (cf. Fig. 1): (i) a *search space definition* specifying the parameters to be optimized and their allowable domains; (ii) *application requirements* defining the optimization goal subject to specific constraint(s) as in Eq. (1); (iii) *termination criteria* such as the maximum number of testbed trials that can be executed; and (iv) when available, a *context source*, such as the protocol source code and/or documentation from which the context initialization module derives protocol-aware guidance.

3.2 Module Design and Instantiation

We instantiate LAPO as a simple reasoning pipeline (Fig. 2) in which each module is implemented by a single LLM agent producing the module output. In this work, we assume the *context source* to be a repository containing the protocol source code. The context initialization module is thus realized as a *repository analyzer* agent that takes as input the parameters to be optimized, iteratively retrieves relevant files from the protocol repository, and compresses the extracted information into a *protocol summary*[4]. The optimization module is realized as an *optimizer* agent that consumes the protocol summary together with the application requirements (ARs), the parameter search space, and the experimental observations to propose the next test-point to be evaluated experimentally until certain termination criteria are met. Importantly, if the optimizer requires additional protocol-specific insight at any stage of the loop, it can request further info, which the repository analyzer retrieves and returns.

System prompt and guidance. The LLM's behaviour is governed by a *system prompt*, a high-priority instruction that defines its role, objectives, and relevant context, thereby shaping its decision-making throughout a session. In particular, the optimizer's prompt is critical, because it interprets the experimental outcomes and directly drives the optimization loop. Hence, it is important to study how much guidance the system prompt must provide for the optimizer agent to handle stochastic feedback effectively. We will evaluate the impact of prompt guidance on optimization stability in § 5. Note that, for completeness, the prompts we provided to the LLMs are available as online artefacts[4].

Implementation and interface. The system is configured through a YAML file specifying the parameters to be optimized and their value ranges, the optimization goal and constraints, a path to the protocol repository, and the iteration budget, along with testbed and execution settings. At every iteration, the optimizer produces a strictly-structured response: the proposed parameter set is returned as a JSON object, followed by a short plain-text rationale. Format validation and correction logic is enforced at runtime to ensure compliance across iterations. On the testbed side, each parameter set corresponds to a distinct firmware configuration. Rather than recompiling the firmware at every iteration, our implementation leverages the binary patching mechanism provided by D-Cube [28], which allows parameter values to be injected into a base firmware image prior to flashing at each testbed trial. This avoids the overhead of repeated compilation and simplifies deployment. For testbeds that do not support binary patching, an alternative approach is to pre-compile a firmware binary for each parameter combination in the search space and select the appropriate one at each iteration. The full implementation code, YAML configuration format, and optimizer reasoning produced at each iteration across all runs are publicly available on GitHub[5], in addition to the artefacts available on Zenodo[4].

[3]Here, "module" denotes a functional block in the architecture. Each module may be implemented by a single LLM agent or by multiple cooperating sub-agents (i.e., multiple LLM calls) that jointly produce a single output.

[4]An example of protocol summary generated by LAPO is available in Zenodo (click here). This repository also contains all the relevant artefacts of this study, including the prompts provided to the LLMs.

[5]http://iti.tugraz.at/LAPO

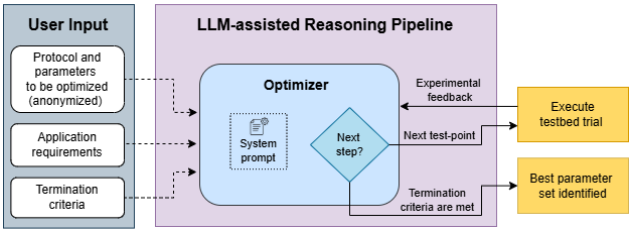

Figure 3: Instantiation of LAPO-A. The protocol and its parameters are anonymized (i.e., identifying cues are removed) and repository-derived context initialization is disabled.

Absence of context (LAPO-A). In certain scenarios, protocol artifacts may not be accessible (e.g., in the case of proprietary third-party implementations), preventing repository-derived context initialization. Likewise, limited or missing internal knowledge of the target protocol can leave the optimizer without sufficient protocol context. To study this case, we consider a context-limited variant of LAPO in which we *anonymize* the protocol (i.e., remove protocol-identifying cues) and *disable* repository-derived context initialization. This way, the optimizer operates solely based on the ARs, the parameter search space, and the accumulated history of experimental trials—proposing the next parameter set to be evaluated experimentally until the specified termination criteria are met. We refer to this variant as LAPO-A (**A** for "anonymous"), and illustrate it in Fig. 3. We will evaluate how the lack of protocol context affects performance in § 5.

4 Experimental Methodology

We describe next the experimental methodology that we follow to evaluate LAPO. We start by listing the proprietary and open-weight LLMs considered in our evaluations (§ 4.1). We then describe the evaluation procedure (§ 4.2), and outline the evaluation metrics used to quantify convergence (§ 4.3).

4.1 LLMs Under Study

We evaluate LAPO with both cloud-hosted proprietary LLMs and open-weight LLMs. Proprietary models represent the current frontier in general-purpose reasoning and tool use, and thus provide a practical reference point on what can be achieved when model quality is not the limiting factor. At the same time, many LPW deployments face constraints making cloud-hosted models undesirable, including recurring API costs, privacy and data-governance concerns (e.g., sharing details of proprietary protocols or experimental traces), and limited customizability. Open-weight models are thus attractive for on-premise deployment and scenarios requiring fine-tuning or tighter control over the inference stack.

Cloud-hosted proprietary LLMs. We hence adopt a two-step evaluation strategy. First, we conduct a design-oriented study using cloud-hosted proprietary models to identify the most effective configuration for LAPO (§ 5). Owing to their stronger reasoning capabilities, these models make the effects of design variants (e.g., prompt guidance level and protocol-context availability) easier to isolate and attribute. Table 1 lists the three proprietary LLMs considered in this study, together with their API pricing[6]. Costs are reported per one million tokens and are divided into *input* tokens—covering all text sent to the model, including system and user prompts, protocol summaries or contextual artifacts, and accumulated trial history—and *output* tokens, which account for all text generated by the model, such as the proposed parameter configurations and any accompanying rationale. This distinction is relevant because LAPO's pipeline incurs both costs at every iteration.

Table 1: Cloud-hosted proprietary LLMs used in § 5.

Provider	Model name	Release date	API pricing (1M tokens)
OpenAI	GPT-5	Aug. 7, 2025	Input: $1.25, Output: $10.00
Anthropic	Claude Sonnet 4.5	Sep. 29, 2025	Input: $3.00, Output: $15.00
Google	Gemini 2.5 Pro	Mar. 25, 2025	Input: $1.25, Output: $10.00

Table 2: Open-weight locally-hosted LLMs used in § 6.

Model ID	Provider	Family	Size	Weights
`gemma-3-12b-it`	Google	Gemma 3	12B	Full
`gemma-3-27b-it`	Google	Gemma 3	27B	Full
`llama-3.1-8b-instruct`	Meta	Llama 3.1	8B	Full
`llama-3.1-70b-instruct`	Meta	Llama 3.1	70B	Full
`llama-3.1-70b-instruct-awq`	Meta	Llama 3.1	70B	AWQ

Locally-hosted open-weight LLMs. We then evaluate LAPO's best-performing configuration with open-weight models, so to quantify the performance gap in a deployable, on-premise setting (§ 6). Table 2 lists the five open-weight models considered in this study. We focus on *instruction-tuned* models that are additionally trained to follow human instructions and produce structured, task-oriented responses: this makes them more suitable for agentic and decision-making tasks than base models that primarily optimize next-token prediction. To obtain a representative view across architectures and deployment trade-offs, we include multiple *model families* and, within the same family, several variants that differ in *parameter count* (column "Size") and *weight* format. This allows us to assess how reasoning performance scales with model capacity while also capturing practical memory/throughput trade-offs relevant to edge or on-premise inference. In particular, "Full" entries denote the original non-quantized weights, whereas "AWQ" entries denote 4-bit Activation-aware Weight Quantization, which reduces memory footprint and inference cost at the potential expense of reasoning accuracy. All open-weight models are evaluated locally on the *Multi-Site Computer Austria (MUSICA)* high-performance computing cluster, using NVIDIA H100 GPUs with vLLM [33], an open-source inference and serving engine designed for fast and memory-efficient LLM deployment. All models are evaluated using greedy decoding (`temperature=0`) to ensure deterministic behaviour and standard vLLM values (`top_p=1.0`, `top_k=-1`). For consistency, we cap the maximum output length at 8192 tokens per LLM call; in practice, the optimizer's per-iteration outputs are well below this limit, so this cap does not constrain reasoning.

4.2 Evaluation Procedure

To evaluate the effectiveness of LLM-assisted parameter optimization, we compare the performance of different variants of LAPO against that of APEX[7], the state-of-the-art framework for LPW protocol parametrization described in § 2.1. We focus primarily on how

[6]Prices reflect our usage as of Jan 25, 2026, and may vary across tiers/regions.

[7]We focus on APEX-EI, the variant recommended by the authors that uses the expected improvement as a next test-point selection strategy.

quickly can a framework converge to the best parameter set despite noisy experimental data.

Protocols and dataset. To this end, we adopt a *trace-based emulation* methodology leveraging the empirical dataset used to evaluate APEX, which was published by Hydher et al. [10]. This open-source dataset contains a large series of real-world measurements collected using the D-Cube[8] public testbed [28] for two well-known LPW protocols: Crystal [11] and RPL [34]. These two protocols are fundamentally different in nature: Crystal uses concurrent transmissions; RPL adopts a classical routing approach[9]. The dataset contains repeated measurements for all protocol parameters listed in Table 3. For Crystal, two parameters are varied, resulting in 16 distinct parameter combinations; for RPL, three parameters are explored, yielding 36 combinations. To capture the variability of wireless experiments, each parameter configuration is evaluated through six independent runs[10]: this leads to a dataset of 96 trials for Crystal and 216 trials for RPL.

Trace-based emulation. We leverage the collected dataset to enable controlled and repeatable comparisons across optimization strategies in § 5 and § 6 while preserving the stochastic nature of real wireless experiments. Instead of interacting with a live testbed, the evaluated optimization framework is connected to an emulator that replays the experimental traces from the dataset and exposes them through the same interface used to query the testbed API. From the optimizer's perspective, each interaction is therefore indistinguishable from executing a real experiment. Concretely, at each iteration of the optimization process, when a parameter configuration is selected for testing, the emulator returns the outcome of a single experimental trial from the dataset that is associated with that specific configuration. The returned trial is picked randomly across the six available repetitions of the selected parameter combination, and can be used only once within a given optimization run. Once all recorded trials for a configuration are consumed, that parameter combination is marked as exhausted and is no longer eligible for selection, forcing the optimizer to test the next test-point.

Repetition strategy. When comparing different approaches, it is important to account for the stochastic nature of the optimization process. An optimizer may occasionally identify the best parameter configuration within the first few trials purely *by chance*, which does not necessarily reflect its typical behaviour or overall effectiveness. To obtain a reliable estimate of performance, results must hence be averaged over multiple independent optimization runs. At the same time, repeating experiments too many times is *impractical* in our setting: we evaluate multiple LAPO variants across several LLMs, and proprietary cloud-hosted models incur *non-negligible API costs* (easily beyond 1000 $ in our case). We therefore limit proprietary-model experiments to *8 optimization runs* per experiment. For open-weight models, API cost is not a constraint, but evaluation time is bounded by available GPU resources; in this case, we increase the repetition count to *40 optimization runs* to obtain tighter estimates under stochastic feedback. To ensure fairness, we evaluate our baseline APEX using the same number of optimization runs (i.e., 8 or 40 depending on the type of experiment).

Table 3: LPW protocols considered in this work, along their parameters to be optimized and possible values.

LPW protocol	Parameter	Possible values
Crystal	Transmission power (dBm)	$\{-5, -3, -1, 0\}$
	Number of transmissions	$\{1, 2, 3, 4\}$
RPL	DIO interval (ms)	$\{2^4, 2^8, 2^{12}, 2^{16}\}$
	Rank threshold	$\{1.5, 3, 6\}$
	Max link metric	$\{16, 32, 64\}$

Table 4: Application requirements (ARs) used in this work. E_c: energy consumption; PRR: packet reception ratio.

ID	LPW protocol	Optimization goal	Constraint
AR_1	Crystal	Minimize E_c	$PRR \geq 65\%$
AR_2		Minimize E_c	$PRR \geq 96\%$
AR_3	RPL	Minimize E_c	$PRR \geq 65.5\%$
AR_4		Minimize E_c	$PRR \geq 93\%$

Testbed validation. To validate that LAPO's closed-loop optimization procedure also converges in an actual testbed environment, we complement our trace-based evaluations by parametrizing "live" a protocol on the D-Cube testbed across different network topologies and real-world conditions (§ 7).

Application requirements. Table 4 lists the optimization goals and constraints used to evaluate the various parametrization approaches: we include both loose and tight constraints as in [10] to assess performance under varying conditions.

4.3 Evaluation Metrics

Each optimization task is defined by an application requirement (AR) consisting of two parts: an objective to optimize (optimization goal, e.g., minimize E_c), and a constraint that must be satisfied (e.g., $PRR \geq 96\%$), as described in Eq. (1). Looking only at the objective can be misleading, because the selected configuration may yield a (near-) optimal goal value while still violating the imposed constraint. For this reason, we evaluate performance using two complementary metrics, both computed from the *"current-best" configuration* identified by the optimizer within a given trial budget, i.e., the parameter set (among those tested experimentally so far) that achieves the best goal value and satisfies the AR constraint (according to the received experimental feedback).

Optimality gap (G). This metric measures how far the optimizer is from the best feasible solution, with the latter being the best parameter set across the full dataset that also satisfies the AR constraint. Specifically, we compute G as the absolute difference between the median goal value[11] of the optimizer's "current-best" configuration and that of the true best configuration. Both median goal values are computed from the full empirical dataset, i.e., assuming *oracle* access to all recorded trials, and are used only for post-hoc evaluation; during optimization, methods observe only the outcomes of the

[8] Using D-Cube, a data collection application is run over an IEEE 802.15.4 mesh network, with one node acting as destination, and several nodes acting as data source. The identity of nodes depends on the chosen testbed layout.

[9] Crystal uses an implementation based on Baloo [12] running on TelosB devices; RPL employs Contiki-NG's RPL-Lite [25] running on nRF52840-DK devices. For both protocols, each source node transmits packets periodically—every second when using Crystal, and every 5 seconds when using RPL.

[10] Each testbed run lasts 10 minutes for Crystal and 18 minutes for RPL, plus 1 and 3 minutes to allow the network to settle, respectively.

[11] We use the median (computed across the six repetitions of each parameter configuration) in order to obtain a robust estimate that is less sensitive to outliers and random experimental fluctuations.

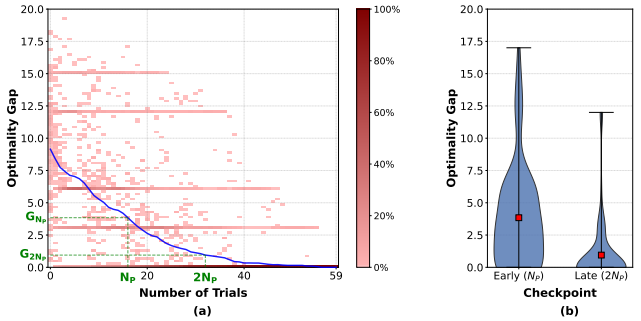

Figure 4: Illustration of convergence using the optimality gap metric (G) and its checkpoint-based summary. The green markers highlight the mean G at the early and late checkpoints (i.e., after N_p and $2N_p$ trials, with N_p being the number of distinct parameter configurations).

experimental trials. Note that G is only computed if the "current-best" configuration satisfies the AR constraint according to the data available to the *oracle*.

Constraint satisfaction rate (S). This metric measures how often the optimizer selects a configuration that satisfies the AR constraint. It is computed as the fraction of optimization runs for which the "current-best" configuration selected by the optimizer satisfies the constraint based on *oracle* knowledge. This metric thus well-complements G: a near-zero gap alone could be misleading, as the optimizer may converge to a configuration with an excellent goal value while still violating the constraint. A low S together with $G \approx 0$ can hence reveal unreliable or infeasible optimization behaviour.

Visualizing the metrics. To characterize how quickly an optimization framework converges to the best feasible solution, we can track both G and S as a function of the number of testbed trials. Fig. 4(a) illustrates an example on how we track and report G using synthetic data for 40 optimization runs. The heatmap shows the distribution of G as the number of performed trials increases, with color intensity indicating the fraction (%) of runs exhibiting a given G at each trial count. By inspecting the figure, we can observe that there are four main points (red horizontal lines) in which the optimization process tends to be trapped. However, most optimization runs have identified the best parameter set after 40 trials, as reflected by the darker red regions around $G \approx 0$, and by the convergence of the solid blue line—representing the mean G across runs—toward this value.

To enable easier comparison and provide a more compact presentation of our results, we summarize convergence by reporting both G and S at two evaluation *checkpoints*:

- *Early checkpoint*: computed after N_p trials, with N_p being the number of distinct parameter configurations in the search space. Accordingly, we use notation G_{N_p} and S_{N_p}.
- *Late checkpoint*: computed after $2 \cdot N_p$ trials, which reflects performance after additional experimental feedback had allowed refinement beyond the initial exploration.

Fig. 4(b) shows the compact visualization used throughout § 5 — § 6, showing the gap distribution at N_p and $2N_p$ trials.

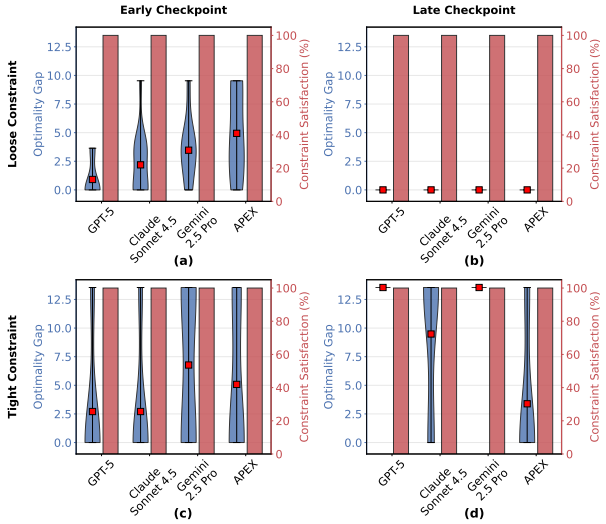

Figure 5: Optimization of Crystal using the dual-stage architecture (LAPO), proprietary LLMs, and minimal prompt specification. Blue violins show the optimality gap G, with the red square denoting the mean. Red bars show the constraint satisfaction rate S.

5 Performance of Cloud-Hosted Proprietary LLMs

Using the methodology described in § 4, we evaluate LAPO's performance using the proprietary LLMs listed in Table 1.

5.1 Impact of Prompt Guidance

We start by evaluating the impact of prompt guidance on optimization performance. First, we use a *minimal specification* in which the Optimizer's prompt instructs it to identify the best-performing feasible parameter set as quickly as possible, but provides no explicit guidance on handling stochasticity. We later use a *median-informed specification*, in which we add explicit statistical guidance to deal with noisy feedback.

Minimal specification. Fig. 5 shows LAPO's performance when parametrizing Crystal under loose (AR_1) and tight (AR_2) constraints (see Table 4) at the early (N_p) and late ($2N_p$) checkpoint, comparing it with that of our baseline, APEX. Under the looser constraint (Fig. 5(a–b)), all three proprietary LLMs reach $G = 0$ by the late checkpoint, matching APEX. At the early checkpoint, instead, they seem to converge faster to $G = 0$ than APEX, with GPT-5 exhibiting the best performance. Under the tighter constraint (Fig. 5(c–d)), the picture changes completely: while performance is comparable across methods at the early checkpoint, the optimality gap of proprietary LLMs is much higher at the late checkpoint, indicating unstable convergence. We attribute this instability to how the optimizer interprets stochastic outcomes without explicit aggregation guidance. Given the stochastic nature of the results, the optimizer can overreact to individual samples: rare constraint violations can trigger premature rejection, while outlier goal values can trigger selection, even when repeated trials would suggest the opposite. More generally, the system treats individual observations as decisive rather than aggregating evidence across repeated trials, which destabilizes convergence under tight constraints.

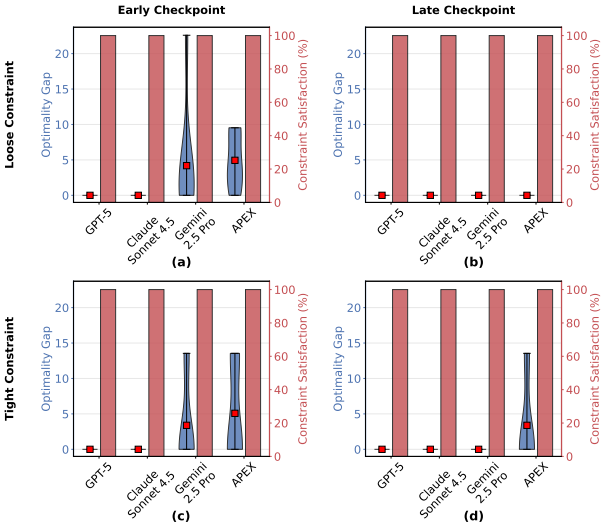

Figure 6: Optimization of Crystal using the dual-stage architecture (LAPO), proprietary LLMs, and median-informed prompt specification.

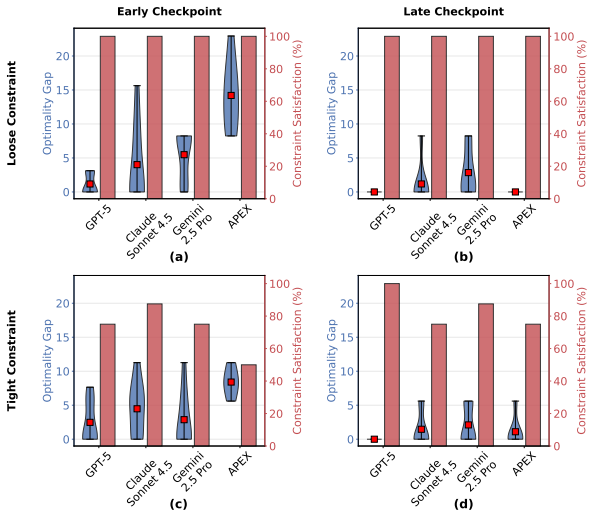

Figure 7: Optimization of RPL using the dual-stage architecture (LAPO), proprietary LLMs, and median-informed prompt specification.

Median-informed specification. We hence modify the prompt and instruct the optimizer to use the *median* over repeated evaluations of a parameter set when assessing both the goal value and constraint satisfaction, and to repeat tests when needed to reduce outlier effects. We also advise de-prioritizing configurations that are clearly unlikely to satisfy the constraint based on the evidence observed so far. Fig. 6 shows how LAPO benefits significantly from the introduction of the median-informed prompt specification. Under the loose constraint (AR_1), all three proprietary LLMs converge again to $G = 0$ by the late checkpoint, matching APEX, but two of them (GPT-5 and Claude Sonnet 4.5) already do so by the early checkpoint. Under the tight constraint (AR_2), the statistical guidance removes the instability observed with the minimal prompt: both GPT-5 and Claude Sonnet 4.5 achieve $G = 0$ at both early and late checkpoints, while Gemini 2.5 Pro does so after $2N_p$ trials. Since the constraint satisfaction rate is 100% across all methods, we can conclude that median-based aggregation prevents outlier-driven rejections of otherwise feasible configurations and stabilizes the optimizer's comparisons under stochastic feedback.

RPL. So far, we have evaluated LAPO's performance by optimizing Crystal. We study next whether the same trends apply when considering RPL, a protocol more stochastic in nature than Crystal. We focus on the median-informed prompt specification, and optimize RPL under both loose (AR_3) and tight (AR_4) constraints. Fig. 7 shows our results: GPT-5 performs best across all scenarios at both early and late checkpoints. All proprietary models outperform APEX at the early checkpoint, whereas performance is broadly on-par at the late checkpoint—with APEX being more competitive than Claude Sonnet 4.5 and Gemini 2.5 Pro for both constraints.

Compared to Crystal, the RPL traces exhibit higher variance: due to the noisy experimental feedback, the optimizer is thus unable to fully exploit prior knowledge. Indeed, we can observe that the increased stochasticity often biases the optimizer toward early samples—for example, overestimating a configuration after an unusually good outcome or prematurely discarding it after a rare violation. Consequently, the optimizer spends additional iterations re-evaluating such configurations to correct these initial biases before convergence stabilizes. This also results in a lower constraint satisfaction (red bars) when optimizing RPL under the tight AR constraint: candidate configurations that appear promising can occasionally violate the constraint across repeated trials, requiring additional iterations to steer the search back into the feasible region. Yet, our results confirm that median-based guidance helps proprietary LLMs to improve their results and converge faster to the best parameter configuration, a consistent behaviour observed across both protocols.

5.2 Impact of Protocol Context

We next evaluate whether LAPO remains effective in absence of protocol-specific context. To this end, we use the LAPO-A variant defined in § 3.2, in which repository-derived context initialization is disabled and protocol-identifying cues are removed. We optimize both Crystal and RPL, focusing on the application requirements with the tighter constraints (AR_2 and AR_4, respectively), and using the median-informed prompt specification, due to its superior performance.

Crystal. Fig. 8 reports the performance of LAPO-A when optimizing Crystal for AR_2. At the early checkpoint, all models exhibit non-zero optimality gaps and a slower initial convergence when protocol context is unavailable (compared to Fig. 6(c)). By the late checkpoint, GPT-5 and Claude Sonnet 4.5 close the gap to zero while maintaining full constraint satisfaction, whereas Gemini 2.5 Pro retains a small residual gap (in contrast, it achieved $G = 0$ when provided with protocol context, as shown in Fig. 6(d)). Overall, these results suggest that strong proprietary models can recover good solutions even in the absence of protocol context, but the lack of context primarily manifests as slower convergence.

RPL. Fig. 9 shows the performance of LAPO-A when optimizing RPL for AR_4. Overall, the optimality gaps and constraint satisfaction rates remain similar to those observed with context enabled (i.e., in

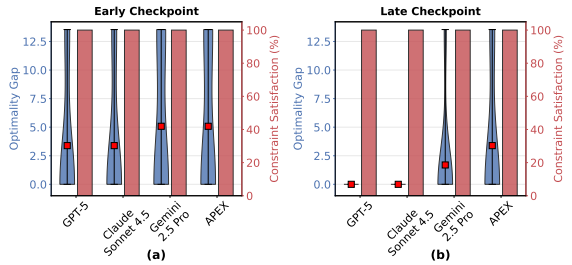

Figure 8: Optimization of Crystal using proprietary LLMs, no context (LAPO-A), and median-informed prompt specification. We focus on a tight constraint (AR_2).

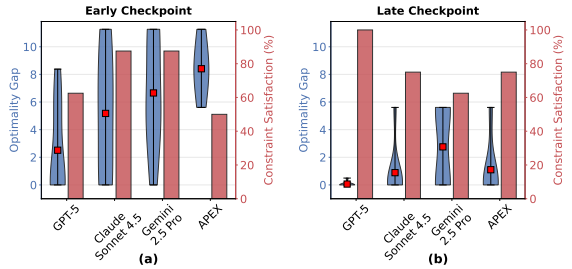

Figure 9: Optimization of RPL using proprietary LLMs, no context (LAPO-A), and median-informed prompt specification. We focus on a tight constraint (AR_4).

Fig. 7), except for a higher G by Gemini 2.5 Pro at the early checkpoint. We attribute this to the higher variance exhibited by RPL: in this regime, the optimization trajectory is dominated by stochastic samples and the need to repeat evaluations for robust median estimates, so additional protocol context yields only marginal—and often hard to separate—gains within the limited number of trials.

6 Performance of Open-Weight LLMs

We next evaluate LAPO with open-weight LLMs to assess how the best configuration identified with proprietary models in §5 (i.e., context enabled and median-informed prompt specification) translates to deployable, on-premise inference.

Crystal. Fig. 10 reports the performance of LAPO when optimizing Crystal. Overall, open-weight models of the same family with a higher parameter count consistently achieve smaller optimality gaps than their smaller counterparts at both early and late checkpoints. Under the loose constraint (AR_1, Fig. 10(a–b)), the *larger* open-weight models achieve much smaller optimality gaps than APEX at the early checkpoint, indicating faster initial progress. However, this advantage does not persist at the late checkpoint, where APEX reaches $G = 0$ in contrast to all open-weight models. A plausible explanation is that, as the prompt grows with the amount of experimental feedback, the open-weight models struggle to retain and consistently exploit the full optimization context, which can lead to degraded reasoning quality in later iterations and hence weaker refinement. Under the tight constraint (AR_2, Fig. 10(c–d)), both Gemma 3 models achieve smaller gaps than APEX, with the larger one (27B parameters) reaching $G = 0$ by the late checkpoint. Although Llama 3.1 70B AWQ also reaches $G = 0$, its constraint satisfaction rate is lower, and hence not directly comparable. Comparing the two Llama 3.1 70B variants, the full-precision model

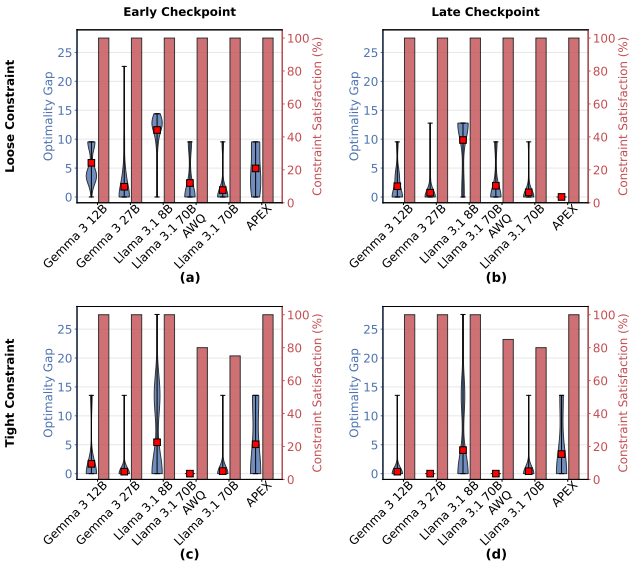

Figure 10: Performance of LAPO when optimizing Crystal using locally-hosted open-weight LLMs.

achieves lower S than AWQ and also attains a slightly higher gap. We attribute this to quantization, which mildly regularizes the optimizer's proposals.

RPL. Fig. 11 reports the performance of LAPO when optimizing RPL. Under the loose constraint (AR_3, Fig. 11(a–b)), most open-weight models show smaller mean optimality gaps than APEX at the early checkpoint, suggesting faster initial progress. However, by the late checkpoint, APEX outperforms all open-weight models. This mirrors the trends observed when optimizing Crystal, indicating that the optimizers leveraging open-weight LLMs make progress, but do not fully match APEX' refinement as the number of experimental trials increases.

Under the tight constraint (AR_4, Fig. 11(c–d)), constraint satisfaction drops substantially at the early checkpoint, reflecting RPL's higher stochasticity and the difficulty of maintaining feasibility under a strict AR. The optimality gap trends match those observed for the loose constraint, whereby open-weight models perform on-par or better than APEX at the early checkpoint, but are outperformed in the long run, where APEX converges to optimality faster.

Model size and quantization. For Crystal, Fig. 10 hints a capacity trend: when comparisons are made at similar constraint satisfaction rates S, higher-parameter optimizers tend to achieve smaller mean optimality gaps G. In a few settings, we observe a gap–satisfaction trade-off, where a larger model reduces the mean gap but attains a lower S, making the overall advantage ambiguous. Still, across comparable cases, the larger quantized models (AWQ) generally outperform the *smaller* full-precision models.

For RPL, Fig. 11 shows no clear winner when jointly considering G and S. Under the loose constraint, S is near-saturated, and larger models are not consistently better in G; under the tight constraint, gap improvements often come with lower satisfaction (or higher variability). Thus, overall, model size does not yield a consistent advantage for RPL within our evaluation budget. Finally, we observe

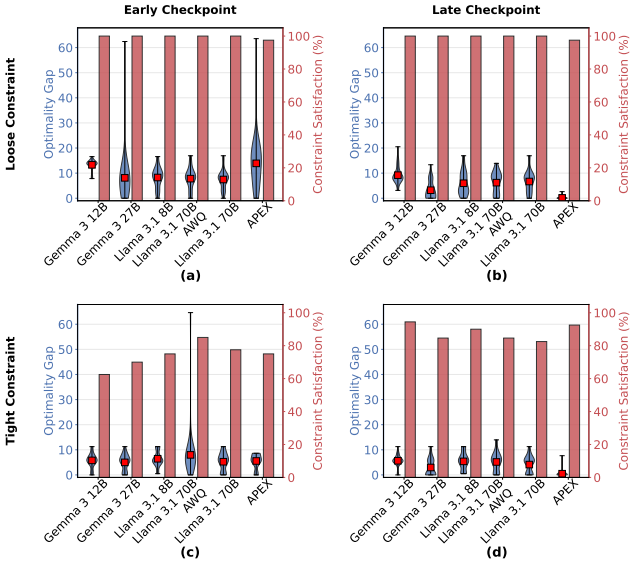

Figure 11: Performance of LAPO when optimizing RPL using locally-hosted open-weight LLMs.

that Llama 3.1 70B AWQ performs similar to the full-precision variant for both Crystal and RPL: this suggests that AWQ does *not* degrade the reasoning-driven optimization behaviour studied here.

7 Validation on a Real-World Testbed

Finally, as a validation of our earlier results, we evaluate how the proposed LLM-based optimization framework converges when executed "live" on a public LPW testbed. We select D-Cube due to its popularity and its ability to generate controlled RF interference, which allows to assess LAPO under real-world dynamics and harsh environmental conditions.

We consider three scenarios (called *layouts* in D-Cube), spanning two diverse network topologies and both absence/presence of RF interference[12]. All our experiments use LAPO with context enabled and the median-informed prompt specification. We select Gemini 2.5 Pro as LLM, since it showed the *least* stable performance among the proprietary models in § 5 (cf. Fig. 6), and we aim to stress-test the proposed framework. We use AR_1 in all three scenarios and track convergence using the optimality gap G as a function of the number of executed testbed trials[13]. Fig. 12 reports the results we measured across *six* "live" optimization runs for each scenario. Each curve shows the gap between the median goal value of the *"current-best" configuration* identified by the optimizer (among those tested experimentally so far) and the median goal value of the best parameter set across 60 iterations of the optimization loop (i.e., testbed trials) within each run.

Across all scenarios, LAPO quickly converges to $G = 0$, i.e., it takes few iterations (7 in average) to match the best feasible median goal value observed among all evaluated configurations in that run.

[12]We use D-Cube's data collection layouts 1 and 4 without added RF noise; as well as layout 1 with Wi-Fi interference generated using JamLab-NG [27] with ≈ 5 ms bursts in periods of 13 ms using a TX power of ≈ 30 mW.

[13]We omit the constraint satisfaction rate, as it is ≈ 100% across all scenarios.

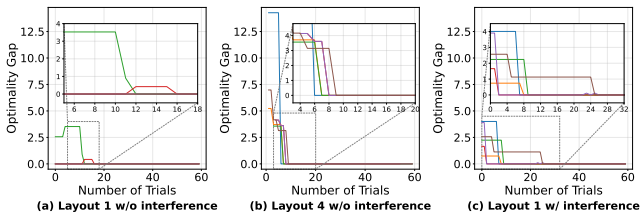

Figure 12: Performance of LAPO when optimizing Crystal for AR_1 *live* on a real-world testbed.

Specifically, $G = 0$ is reached after an average of 3 trials in D-Cube's *layout 1* (without RF interference), 8 trials in *layout 4* (without RF interference), and 11 trials in *layout 1* (but in the presence of RF interference). Finally, we evaluate the LLM inference overhead, which turns out to be negligible compared to the duration of a testbed trial (which takes several minutes): across all live runs, prompt execution takes on average 6.81 ± 5.37 s per iteration.

8 Discussion and Future Work

LLMs can serve as effective decision-makers in a closed-loop parameter optimization pipeline, but their reliability hinges on explicitly accounting for the stochasticity of wireless experiments. In our study, a median-informed task specification proves crucial: without aggregation guidance, the optimizer can overreact to individual trials, prematurely rejecting feasible configurations or chasing outlier-driven improvements. Median-based assessment (and repeating evaluations when needed) stabilizes convergence under tight constraints, underscoring the need for statistically grounded decision rules when reasoning over noisy feedback.

Decision pattern analysis. To better understand how the optimizer behaves across iterations, we analyzed over 60,000 optimizer iterations across more than 1,000 optimization runs. A first distinction emerges from the decision type at each iteration, which can be determined programmatically by comparing the proposed parameter set against those tested previously. Proprietary models distribute their iterations roughly evenly between proposing new configurations (≈ 36%), revisiting previously tested ones (≈ 37%), and deliberately re-testing the immediately preceding configuration (≈ 27%). Open-weight models, in contrast, spend the majority of their iterations revisiting previously tested configurations (≈ 70%) and rarely re-test the immediately preceding one (≈ 4%), with the remaining iterations devoted to proposing new configurations (≈ 25%).

To quantify the cost of this repetition behaviour, we measure the *redundant repeat rate*: the fraction of iterations in which the optimizer re-tests a configuration that has already been evaluated more than k times while at least m other non-exhausted, constraint-satisfying configurations with strictly better running median goal values are available at that point. Fig. 13 reports the redundant repeat rate for proprietary and open-weight LLM models parametrizing both Crystal and RPL across a range of $m = [3, 10]$ values with a fixed $k = 2$. Open-weight models exhibit consistently higher redundant repeat rates than proprietary models across both protocols and all threshold values, with the gap being particularly pronounced for RPL, where the larger parameter space offers more alternatives that the optimizer could explore instead of re-testing already well-evaluated configurations. Moreover, an examination of the rationale

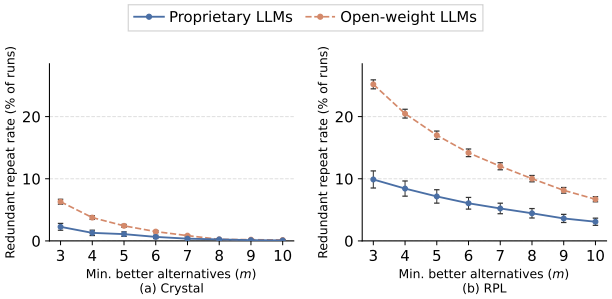

Figure 13: Redundant repeat rate: fraction of iterations repeating a configuration already evaluated more than k times while at least m better feasible alternatives exist. k=2 fixed; error bars show standard error of the mean across runs.

text reveals that fewer than one in ten repeat decisions by open-weight models acknowledge the repetition as intentional statistical refinement[14]. We also observe that the reasoning provided by open-weight models when repeating a parameter set is often framed as if a given configuration is being proposed for the first time, i.e., without referencing that the same parameter set was already evaluated. This could hint that the open-weight models lost the context related to the previous run of that parameter set or that the system prompt emphasizing the median-based reasoning is overlooked at that point. Proprietary models, instead, show the opposite pattern: more than nine in ten repeat decisions are explicitly motivated by median-driven reasoning, framing the choice as deliberate evidence gathering.

Open-weight models also frequently base their decisions on the outcome of the single most recent trial, without referencing results from earlier evaluations of the same configuration. Proprietary models, in contrast, consistently reference accumulated evidence across multiple trials when justifying their choices. The following two excerpts, drawn from optimizer logs under the same protocol and application requirement, illustrate this contrast concretely. A proprietary model (GPT-5) repeating a configuration writes:

> *"31/2 currently satisfies reliability with the lowest observed median energy among feasible sets. To adhere to the median-based evaluation and ensure robustness, repeat this configuration to refine and confirm the medians."*

An open-weight model (Llama 3.1 8B), re-proposing an already tested configuration, writes:

> *"Next set to test: Increasing tx_power to 23 to balance energy consumption and reliability."*

These patterns suggest that the performance gap between model categories could be partly explained by how consistently models internalize the statistical guidance provided in the prompt: proprietary models develop a structured evidence-gathering strategy, while open-weight models tend to react to individual observations without building a cumulative picture of the search space. These findings also point to concrete guardrails that could improve reliability under noisy feedback, such as requiring the optimizer to explicitly reference accumulated evidence before accepting or rejecting a configuration, enforcing conservative fallbacks near the constraint boundary, and triggering automatic repetitions when the number of observations for the current-best configuration falls below a minimum threshold.

Prompting and multi-agent extensions. Prompt specification substantially affects the convergence stability, suggesting room for more principled designs. The reasoning patterns described in the previous paragraph point to specific failure modes that a multi-agent architecture could address. For instance, a repetition-awareness agent could flag when a proposed configuration has already been tested, preventing the unacknowledged repetitions that dominate open-weight repeat decisions. A median-enforcement agent could require the optimizer to reference accumulated statistics before declaring a configuration optimal, counteracting the recency bias observed in open-weight models. More broadly, separating feasibility assessment, uncertainty estimation, and candidate generation into distinct steps, each handled by a dedicated agent, may improve consistency and make optimization traces easier to interpret. These extensions remain compatible with the current LAPO architecture, as each agent can be realized as an additional module within the existing pipeline.

Practical guidance on optimizer choice. When access to high-capability frontier models is available, an LLM-based optimizer is generally the better choice, as it tends to make faster progress and achieve stronger overall performance. When limited to smaller or less-capable LLMs, performance is broadly comparable to APEX, making the choice less critical in typical operating conditions. In highly stochastic scenarios—where experimental outcomes are noisy and constraint violations are likely—APEX is often the safer option, since its statistically grounded optimization is more robust than smaller LLMs, whose decisions can exhibit higher variability.

Accessibility and expert-guided use. LAPO removes the burden of requiring deep protocol expertise for systematic parameter optimization, making it accessible to a broader range of developers. This is achieved through the repository analyzer, which automatically extracts and compresses protocol knowledge from source code, thereby removing the need to manually encode parameter semantics or interaction effects. At the same time, experienced developers can equally benefit: rather than exploring the full parameter space, an expert can restrict the search to a filtered subset of configurations they consider most promising, reducing the number of required trials and stopping once the results are satisfactory. In both cases, real-world testing remains unavoidable regardless of expertise level, as protocol performance is shaped by environmental conditions such as RF interference, network topology, and hardware characteristics that cannot be captured by simulation or prior knowledge alone. Given the stochastic nature of wireless experiments, even an expert must repeat evaluations to rule out outlier results, making efficient trial budget management a practical necessity that LAPO directly addresses.

Deployment and model adaptation. Open-weight models enable on-premise deployment but can be less stable under stochastic feedback. Promising directions include lightweight adaptation, hybrid pipelines where a surrogate proposes candidates while the

[14]The qualitative observations on repeat awareness and recency bias are based on keyword-based pattern matching over the optimizer's rationale text and should be interpreted as indicative of relative behavioural differences between model categories.

LLM manages repetition and constraint handling, and *retrieval-augmented generation* (RAG) to ground decisions in protocol documentation, code snippets, and prior trial traces. An additional direction is *online learning*, where the optimizer leverages feedback from completed runs to update its preferences and improve stability and sample efficiency over time.

Context quality and representation. In general, the optimizer's decisions are fundamentally limited by the context it has access to. A promising direction is to enrich the prompt with structured, deployment-specific information such as the testbed layout, network topology, and presence of RF interference. Providing such grounded context could help the LLM reason more efficiently about feasible regions, avoid implausible configurations, and reduce back-and-forth behaviour across iterations under noisy feedback. Another open question is how the *form* of context affects performance: for example, whether protocol details expressed as natural language, configuration tables, or source-code snippets lead to different optimization behaviour and stability.

Testing a broader set of protocols. In this work, we focus on Crystal and RPL as two fundamentally-different LPW protocols, enabling a controlled yet representative evaluation: Crystal relies on concurrent transmissions while RPL adopts a classical routing approach, covering two distinct ends of the LPW protocol design spectrum. This choice deliberately stress-tests the optimizer across contrasting operational regimes, from a relatively deterministic synchronous flooding protocol to a more stochastic multi-hop routing protocol, providing a meaningful basis for assessing generality across fundamentally different protocol designs. We plan next to verify the generality of our results across additional protocol stacks and larger parameter spaces.

Acknowledgments

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) [10.55776/DFH5]. For the purpose of open access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission. The computational results have been achieved using the Austrian Scientific Computing (ASC) infrastructure.

References

[1] C. Adjih et al. 2015. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *Proc. of the 2nd World Forum on Internet of Things*.

[2] S. Bal and L. Mandal. 2025. *Conversational LLM Framework for Secure and Energy-Efficient Wireless Sensor Networks.* Technical Report Preprint, 10.21203/rs.3.rs-7714424/v1. Research Square.

[3] C.A. Boano et al. 2018. IoTBench: Towards a Benchmark for Low-Power Wireless Networking. In *Proc. of the 1st CPSBench Workshop*.

[4] F. Despaux, Y.-Q. Song, and A. Lahmadi. 2013. Measurement-Based Analysis of the Effect of Duty Cycle in IEEE 802.15.4 MAC Performance. In *Proc. of the 10th IEEE MASS Conference*.

[5] W. Dong, Y. Liu, Y. He, T. Zhu, and C. Chen. 2014. Measurement and Analysis on the Packet Delivery Performance in a Large-Scale Sensor Network. *IEEE/ACM Transactions on Networking* 22, 6 (Dec. 2014).

[6] F. Foukalas et al. 2019. Dependable Wireless Industrial IoT Networks: Recent Advances and Open Challenges. In *Proc. of the 24th IEEE ETS*.

[7] S. Fu et al. 2015. Experimental Study for Multi-layer Parameter Configuration of WSN Links. In *Proc. of the 35th ICDCS Conference*. IEEE.

[8] J. Han. 2009. Global Optimization of ZigBee Parameters for End-to-End Deadline Guarantee of Real-Time Data. *IEEE Sensors* 9, 5 (2009).

[9] S. Hong et al. 2024. MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. In *Proc. of the 12th ICLR Conference*. 1–29.

[10] M.H.M. Hydher, M. Schuss, O. Saukh, K. Römer, and C.A. Boano. 2025. APEX: Automated Parameter Exploration for Low-Power Wireless Protocols. *ACM Transactions on Sensor Networks (TOSN)* 21, 6 (2025).

[11] T. Istomin, A.L. Murphy, G.P. Picco, and U. Raza. 2016. Data Prediction + Synchronous Transmissions = Ultra-Low Power Wireless Sensor Networks. In *Proc. of the 14th SenSys Conference*.

[12] R. Jacob, J. Baechli, R. Da Forno, and L. Thiele. 2019. Synchronous Transmissions Made Easy: Design Your Network Stack with Baloo. In *Proc. of the 16th EWSN Conference*. Junction Publishing, 106–117.

[13] P.N. Karunanayake, A. Könsgen, T. Weerawardane, and A. Förster. 2023. Q-Learning-Based Adaptive Protocol Parameters for WSNs. *Journal of Communications and Networks* 25, 1 (Feb. 2023), 76–87.

[14] G. Kaur, S.H. Gupta, and H. Kaur. 2022. Optimizing the LoRa Network Performance for Industrial Scenario Using a Machine Learning Approach. *Computers & Electrical Engineering* 100 (May 2022), 107964.

[15] A. Li et al. 2022. *A Lightweight Transmission Parameter Selection Scheme Using Reinforcement Learning for LoRaWAN.* Technical Report arXiv:2208.01824. Cornell University.

[16] A. Liendo et al. 2018. BLE Parameter Optimization for IoT Applications. In *Proc. of the IEEE ICC Conference*.

[17] N.F. Liu et al. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Trans. Assoc. Comput. Linguistics* 12 (2024).

[18] T. Liu, N. Astorga, N. Seedat, and M. van der Schaar. 2024. *Large Language Models to Enhance Bayesian Optimization.* Technical Report arXiv:2402.03921. Cornell University.

[19] Z. Liu and H. Du. 2025. *Model Context Protocol-Based Internet of Experts for Wireless Environment-Aware LLM Agents.* Technical Report arXiv:2505.01834. Cornell University.

[20] N. Mazloomi, M. Gholipour, and A. Zaretalab. 2022. Efficient Configuration for Multi-Objective QoS Optimization in Wireless Sensor Network. *Ad Hoc Networks* 125 (Feb. 2022), 102730.

[21] L. Mottola et al. 2019. makeSense: Simplifying the Integration of Wireless Sensor Networks into Business Processes. *IEEE Transactions on Software Engineering* 45, 6 (June 2019), 576–596.

[22] C. Noda, S. Prabh, M. Alves, and T. Voigt. 2013. On Packet Size and Error Correction Optimisations in Low-Power Wireless Networks. In *Proc. of the 10th IEEE SECON Conference*. 212–220.

[23] F.J. Oppermann, C.A. Boano, M.A. Zúñiga, and K. Römer. 2015. Automatic Protocol Configuration for Dependable Internet of Things Applications. In *Proc. of the 10th IEEE SenseApp Workshop*. 742–750.

[24] X. Peng, Y. Liu, Y. Cang, C. Cao, and M. Chen. 2025. *LLM-OptiRA: LLM-Driven Optimization of Resource Allocation for Non-Convex Problems in Wireless Communications.* Technical Report arXiv:2505.02091.

[25] RPL-Lite in Contiki-NG. [Online] Last access: 2026-02-01. https://docs.contiki-ng.org/en/develop/doc/programming/RPL.html#rpl-lite

[26] A.I. Sayyid-Ali et al. 2025. CheckMate: LLM-Powered Approximate Intermittent Computing. In *Proc. of the 23rd ACM SenSys Conference*.

[27] M. Schuß et al. 2019. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *Proc. of the 16th EWSN Conference*. Junction Publishing.

[28] M. Schuß, C.A. Boano, and K. Römer. 2018. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the 1st CPSBench Workshop*. IEEE, 30–35.

[29] N. Sevim, M. Ibrahim, and S. Ekin. 2024. Large Language Models (LLMs) Assisted Wireless Network Deployment in Urban Settings. In *Proc. of the 100th Vehicular Technology Conference (VTC-Fall)*. IEEE.

[30] J. Shao et al. 2024. WirelessLLM: Empowering Large Language Models Towards Wireless Intelligence. *Journal of Communications and Information Networks* 9, 2 (June 2024), 99–112.

[31] M. Spörk et al. 2020. Improving the Reliability of Bluetooth Low Energy Connections. In *Proc. of the 17th EWSN Conference*.

[32] R. Thind, Y. Sun, L. Liang, and H. Yang. 2025. *OptimAI: Optimization from Natural Language Using LLM-Powered AI Agents.* Technical Report arXiv:2504.16918. Cornell University.

[33] vLLM. [Online] Last access: 2026-02-01. https://vllm.ai/

[34] T. Winter et al. 2012. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.* RFC 6550. IETF.

[35] H. Yang, S. Lambotharan, and M. Derakhshani. 2025. *FAS-LLM: Large Language Model-Based Channel Prediction for OTFS-Enabled Satellite-FAS Links.* Technical Report arXiv:2505.09751. Cornell University.

[36] Z. Zhang, W. Xu, S. Cui, and C.K. Reddy. 2025. *Distractor Injection Attacks on Large Reasoning Models: Characterization and Defense.* Technical Report arXiv:2510.16259. Cornell University.

[37] H. Zhou et al. 2025. *Prompting Wireless Networks: Reinforced In-Context Learning for Power Control.* Technical Report arXiv:2506.06526.

[38] M. Zimmerling et al. 2012. pTUNES: Runtime Parameter Adaptation for Low-power MAC Protocols. In *Proc. of the 11th IPSN Conference*.