

Decoupling Performance from Energy in AI-Based Business Optimization: An Error-Driven Control Loop for Sustainable Tabular Reasoning

Róbert Lakatos

University of Debrecen

Árpád Pándy

`pandy.arpad@inf.unideb.hu`

University of Debrecen

András Hajdu

University of Debrecen

Research Article

Keywords: Green AI, AI-based business optimization, Prompt optimization, Feedback control, Small language models, Energy efficiency, Tabular reasoning, Agentic AI

Posted Date: October 21st, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11435156/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Decoupling Performance from Energy in AI-Based Business Optimization: An Error-Driven Control Loop for Sustainable Tabular Reasoning

Róbert Lakatos¹, Árpád Pándy^{1*}, András Hajdu¹

^{1*}Department of Data Science and Visualization, Faculty of Informatics,
University of Debrecen, Kassai út 26, Debrecen, 4028, Hungary.

*Corresponding author(s). E-mail(s): pandy.arpad@inf.unideb.hu;
Contributing authors: lakatos.robert@inf.unideb.hu;
hajdu.andras@inf.unideb.hu;

Abstract

Background: Adapting large language models (LLMs) to domain-specific tasks usually relies on gradient-based fine-tuning, which has a high energy and carbon cost. AI-based business optimization (AIBO) in regulated sectors requires adaptation methods that are accurate, auditable, and energy-efficient. We formulate error-driven prompt optimization as a discrete-time feedback control loop. A frozen small language model (SLM) inside a Code Generation Agent (CGA) solves arithmetic questions from the TAT-QA financial benchmark. Mispredictions are clustered with HDBSCAN, a domain expert writes a natural-language rule for the selected error cluster, and the prompt is updated iteratively. We measured the energy of the loop directly at the wall outlet over 10 repeated runs and compared it with LoRA and full fine-tuning, which we estimated using a hardware-normalized scaling model.

Result: The optimized CGA reached 66.45% exact match (EM) on the held-out test set with Qwen3 4B, a gain of 9.76 percentage points, which is on par with a GPT-3.5 Turbo-based agent using manually written rules. The complete optimization consumed 351 Wh (coefficient of variation: 0.24%). This is 21.1 times less than the estimated cost of LoRA fine-tuning and 10.1 times less than the estimated cost of full fine-tuning for the same model.

Conclusion: The size of the advantage depends strongly on the base model’s inference speed. Error-driven prompt optimization is a precise, explainable, and provable alternative to energy-intensive training cycles.

Keywords: Green AI, AI-based business optimization, Prompt optimization, Feedback control, Small language models, Energy efficiency, Tabular reasoning, Agentic AI

1 Introduction

Large language models (LLMs) are becoming a core component of AI-based business optimization (AIBO). Companies use them to read reports, answer questions over financial tables, and support decisions in auditing, controlling, and customer service [1–3]. The capabilities of these models follow neural scaling laws: performance improves in a predictable way as the number of parameters, the size of the training data, and the amount of compute grow [4, 5]. The same laws also drive energy demand. Training a single large model can consume hundreds of megawatt-hours and emit tens to hundreds of tonnes of CO₂-equivalent [6–8]. Inference adds a further, continuous cost that grows with every deployed application [9–11]. Data centers already account for a measurable share of global electricity use, and AI is expected to be one of the main drivers of its growth [12, 13].

The Green AI movement argues that efficiency should be reported and evaluated together with accuracy [14]. Several authors have proposed standard methods for measuring and reporting the energy and carbon footprints of machine learning experiments [15, 16]. This view is also becoming a regulatory requirement. The European Climate Law sets a binding target of climate neutrality by 2050, and a 55% reduction of net greenhouse gas emissions by 2030 [17]. The revised Energy Efficiency Directive introduces energy reporting obligations for data centers [18], and the AI Act requires providers of general-purpose AI models to document the energy consumption of their models [19]. For companies, this means that an AI transformation must be planned together with their decarbonization goals. An adaptation method that reaches the required accuracy with less energy is therefore not only a technical improvement but also a business and compliance advantage.

There are two main ways to adapt a pre-trained model to a domain task. The first is gradient-based fine-tuning. It updates either all weights (full fine-tuning) or a small set of additional parameters, as in parameter-efficient fine-tuning methods such as Low-Rank Adaptation (LoRA) [20], QLoRA [21], or prompt tuning [22]. The second way is gradient-free adaptation through the prompt. Here, the model weights remain frozen, and task knowledge is incorporated into the input [23]. Examples include chain-of-thought prompting [24], program-aided reasoning that delegates arithmetic to a code interpreter [25, 26], and automatic prompt optimization methods that search for better instructions [27–31]. Prompt-based adaptation preserves the base model’s general abilities and requires no training infrastructure. However, its energy cost is rarely measured, and it is rarely compared with fine-tuning under the same task and model.

This paper extends our previous work on tabular arithmetic reasoning. First, we introduced an arithmetic-aware Code Generation Agent (CGA) that improved exact match accuracy from 29% to 58% without fine-tuning [32]. Next, we showed that the CGA can run on small language models (SLMs) on-premises, with Qwen3 4B achieving 60% accuracy and surpassing GPT-3.5 Turbo [33]. Our third study added an error-driven optimization loop, which groups the failure modes into clusters and extends the prompt with rules aimed at them; this lifted the SLM accuracy to 66.45% [34]. In the conference version of the present study, we compared the energy of this framework with fine-tuning [35]. The present article extends that comparison in four directions: a

control-theoretic formulation of the method, a detailed description of the measurement protocol and software environment, a broader analysis of the results, and a translation of the energy figures into operational indicators for business decision makers.

The iterative optimization loop has a natural interpretation in control theory. The frozen model and the code interpreter form the plant; the exact match accuracy on a development set is the measured output, the set of mispredictions is the error signal, and the prompt is the control input. A domain expert, supported by error clustering, acts as the controller that turns the error signal into a new prompt rule. This structure is close to iterative learning control, where the input of the next trial is corrected by the error of the previous trial [36, 37], and to human-in-the-loop machine learning, where human judgment is part of the learning process [38]. The formulation makes the method’s assumptions explicit and connects it to the principles of precise, explainable, and provable AI (PEP-AI). The method is precise because it is evaluated with an exact match. It is explainable because every adaptation step is a readable natural-language rule. It is provable because each step is measured, repeated, and reproducible in a containerized environment.

The approach is also relevant for higher education. Business schools and informatics faculties increasingly teach the business applications of AI, and students need hands-on experience with systems that they can inspect and run [39]. On-premises SLMs with a transparent optimization loop are well-suited for this purpose. Their behavior can be demonstrated in practical laboratory sessions using cloud-based Python notebooks, and the error analysis can be taught as exploratory data analysis (EDA) with pandas [40]. Students can observe how a single rule changes a model’s accuracy and energy budget, linking technical skills with sustainability thinking.

The main contributions of this article are the following:

1. We formalize error-driven prompt optimization as a discrete-time feedback control loop with a human-in-the-loop controller.
2. We describe a direct, repeated energy measurement protocol for gradient-free adaptation and report its statistical stability.
3. We compare the measured energy with counterfactual estimates for LoRA and full fine-tuning of the same models, including sensitivity and break-even analyses.
4. We translate the energy results into operational cost and emissions indicators for AIBO decision-makers.
5. We provide a containerized, reproducible implementation of the full pipeline.

The rest of the article is organized as follows. Section 2 presents the dataset, the control-loop formulation, the optimization procedure, and the energy estimation and measurement methods. Section 3 reports accuracy, energy, and cost results. Section 4 discusses limitations, practical applicability, and future research directions, and concludes the article.

2 Methods

This section first describes the task and data 2.1, and the evaluation metrics 2.2. It then presents the CGA 2.3 and formalizes the optimization procedure as a feedback control

loop 2.4, 2.5, 2.6, 2.7. It describes how we estimated the energy of fine-tuning 2.8 and measured the energy of prompt optimization 2.9. Finally, it describes sensitivity and break-even analysis 2.10, reproducibility, and software environment 2.11.

2.1 Task and dataset

We use TAT-QA, a question answering benchmark built from real financial reports that combine tables and text [41]. We work with its direct tabular subset: questions whose answers are derived from the table and require arithmetic operations such as addition, subtraction, percentage change, and comparison. We apply the same filter as in our previous work [32]: we keep the questions whose answer type is *arithmetic* and whose answer can be computed from the table alone, without the accompanying text paragraphs. The filter reduces the TAT-QA development set from 278 to 215 tables and from 1,668 to 497 questions. Applying the same filter to the TAT-QA test set gives 471 questions. Fig. 1 shows the dataset structure and the applied filter.

The development set ($N = 497$) is used by the optimization loop: all global and cluster steps, error clustering and rule synthesis use only these items, and all energy measurements in this article refer to them. The test set ($N = 471$) is held out during optimization and used once to evaluate the final prompt. The 66.45% EM reported for the optimized Qwen3 4B agent is measured on the test set [34].

Each gold answer consists of a numerical value and a scale (billion, million, thousand, percent, or none). A prediction is correct only if both match.

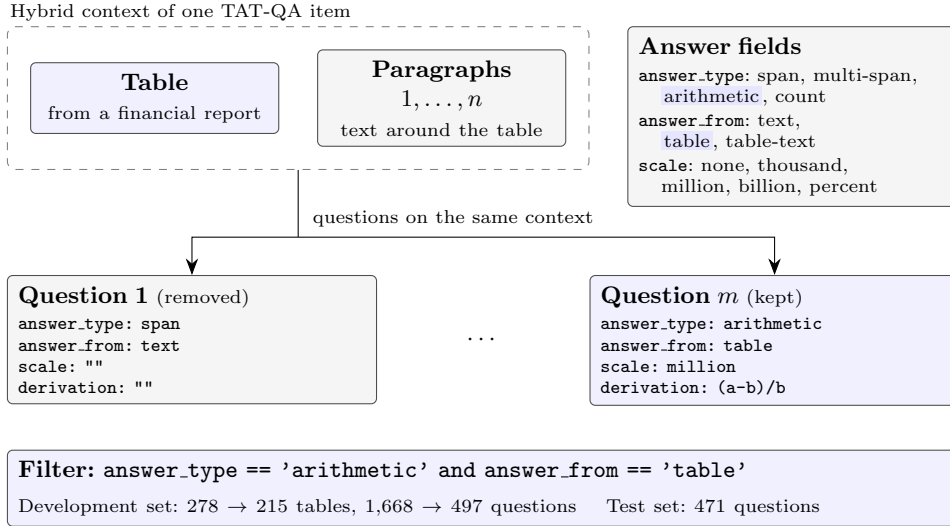


Fig. 1 Structure of a TAT-QA item and the filter applied to obtain the direct tabular subset. Each item consists of a table and the surrounding paragraphs of a financial report, and several questions refer to the same context. Every question carries its answer type, answer source, scale and, for arithmetic questions, the derivation of the gold answer. Only arithmetic questions answerable from the table alone are kept (blue)

2.2 Evaluation metrics

We use exact-match (EM) accuracy as the sole accuracy metric. A financial answer is only useful if its value, sign and scale are all correct. Word-overlap metrics such as F1 reward correct terminology even when the calculation is wrong, and therefore overestimate performance. EM is reported in percentage points (pp).

Energy is reported in watt-hours (Wh). To relate energy to accuracy, we use two derived indicators. The energy-to-accuracy ratio is

$$\eta_{\text{acc}} = \frac{E}{\text{EM}} \quad [\text{Wh/pp}], \quad (1)$$

and the energy-to-improvement ratio is

$$\eta_{\text{imp}} = \frac{E}{\text{EM} - \text{EM}_{\text{base}}} \quad [\text{Wh/pp}], \quad (2)$$

where E is the energy of the adaptation, EM is the accuracy after adaptation and EM_{base} is the accuracy of the base model before adaptation. Lower values are better for both indicators.

2.3 Code Generation Agent

The CGA does not ask the language model to compute the answer directly. Instead, the model receives the question, the table and a prompt, and generates a short Python program. The program is executed in an isolated interpreter, and its output is the predicted answer, including the value, the sign, and the scale. This design moves arithmetic from the language model to the interpreter, a well-known approach to reducing calculation errors in LLMs [25, 26]. It also makes every prediction traceable, because the generated code shows how the answer was obtained [32].

We use two SLMs with about 4 billion parameters: Qwen3 4B [42], and Gemma 3n [43]. Both run on local hardware, so no data leaves the organization. The models are served by the Ollama runtime (version 0.11.4). For Qwen3 4B, we use the `qwen3:4b` model (Q4_K_M 4-bit mixed quantization, 2.6 GB, digest `2bfd38a7daaf`); for Gemma 3n, the `gemma3n:e4b` model (E4B variant, 4-bit quantization, 7.5 GB, digest `15cb39fd9394`). We use the non-thinking (instruct) behavior of both models; for Qwen3 4B, this is enforced by appending the `/no_think` flag to the prompt. Decoding is greedy and deterministic: temperature 0, top- $p = 1$, no repetition, presence or frequency penalty, and at most 400 generated tokens per answer. The generated programs are executed with Python 3.9.5, and numerical outputs are rounded to two decimal places before evaluation.

2.4 Prompt optimization as a feedback control loop

We now describe the optimization procedure in control-theoretic terms. Table 1 summarizes the notation and the role of each quantity in the loop, and Fig. 2 shows the block diagram. The formulation describes the Prompt Extending Algorithm of our previous work [34], which was also used in all experiments of this article.

Let $\mathcal{D} = \{(x_i, a_i)\}_{i=1}^N$ be the development set, where x_i contains a question and its table and a_i is the gold answer. Let M_θ be the language model with frozen weights θ , and let $\text{Exec}(\cdot)$ denote the execution of the generated program, and let $\mathbf{1}[\cdot]$ be the indicator function that evaluates to 1 if its condition is true and 0 otherwise. For a prompt p , the match indicator of item i is

$$m_i(p) = \mathbf{1} [\text{Exec}(M_\theta(x_i; p)) = a_i], \quad (3)$$

and the output of the plant is the exact match accuracy

$$y(p) = \frac{100}{N} \sum_{i=1}^N m_i(p). \quad (4)$$

The error signal is not a single number but a set of mispredicted items

$$\mathcal{E}(p) = \{i \in \mathcal{D} : m_i(p) = 0\}. \quad (5)$$

This set carries structured information about what went wrong, which a scalar error $100 - y(p)$ would lose.

The control input is the prompt. The base prompt p_0 contains no rules. In iteration $k = 1, 2, \dots$ a new rule ρ_k may be appended to the current prompt p_{k-1} , which we write as $p_{k-1} \oplus \rho_k$. The sensor has two parts. First, a feature map ϕ describes each mispredicted item with a feature vector (see Section 2.5). Second, HDBSCAN groups these vectors into clusters:

$$\{\mathcal{C}_k^1, \dots, \mathcal{C}_k^{J_k}\} = \text{HDBSCAN}(\phi(\mathcal{E}(p_{k-1}))), \quad (6)$$

where noise points are not assigned to any cluster. The selection function S orders the clusters by decreasing size, $|\mathcal{C}_k^{(1)}| \geq |\mathcal{C}_k^{(2)}| \geq \dots$, and offers them to the controller in this order. At most two candidates are examined per iteration: the largest cluster $\mathcal{C}_k^{(1)}$ and, if its rule is rejected, the next one $\mathcal{C}_k^{(2)}$. When several clusters have the same size, the expert chooses among them.

In one iteration (Qwen3 4B, iteration 3), the second candidate was a cluster of 6 items, although three clusters of 7 items were also available. All three 7-item clusters contained change questions (calculation pattern $\# - \#$), the same pattern as the first candidate, whose rule had just been rejected, each with a different error type (scale, runtime and selection errors, respectively). The 6-item cluster addressed a different calculation, the selection of the operands of a two-value average $(\# + \#)/\#$.

The largest cluster is preferred because a rule that removes its root cause can be expected to correct the most items.

The controller is the rule synthesis step G , which produces a candidate rule for the examined cluster $\mathcal{C}_k^{(j)}$:

$$\rho_k^{(j)} = G(\mathcal{C}_k^{(j)}, p_{k-1}), \quad j \in \{1, 2\}. \quad (7)$$

In the current implementation, G is performed by a domain expert using the cluster summaries (Section 2.6). Before the rule is used on the whole development set, it is checked locally on the examined cluster:

$$y_k^C = \frac{100}{|C_k^{(j)}|} \sum_{i \in C_k^{(j)}} m_i(p_{k-1} \oplus \rho_k^{(j)}). \quad (8)$$

Because every item of an error cluster is mispredicted by p_{k-1} , the local accuracy before the rule is zero, and y_k^C is also the local improvement. To check that the improvement is consistent rather than accidental, we apply the exact (binomial) McNemar test [44] to the paired outcomes $m_i(p_{k-1})$ and $m_i(p_{k-1} \oplus \rho_k^{(j)})$ on the cluster. All discordant pairs are corrections, so with c_k corrected items the two-sided exact p -value is $\pi_k = \min(1, 2^{1-c_k})$. A rule is accepted if

$$A_k^{(j)} = \mathbf{1} [y_k^C \geq \tau \wedge \pi_k \leq \alpha] = 1, \quad (9)$$

with $\tau = 50\%$ and $\alpha = 0.0625$. The second condition is equivalent to $c_k \geq 5$, so a rule must correct at least half of its cluster and at least five items. The update law of the controller is

$$p_k = \begin{cases} p_{k-1} \oplus \rho_k^{(1)}, & \text{if } A_k^{(1)} = 1, \\ p_{k-1} \oplus \rho_k^{(2)}, & \text{if } A_k^{(1)} = 0 \text{ and } A_k^{(2)} = 1, \\ p_{k-1}, & \text{otherwise.} \end{cases} \quad (10)$$

A rejected rule is discarded and never enters the prompt; the gate in Eq. (9) is therefore applied before any global evaluation of the rule.

The loop stops when both candidates of an iteration are rejected:

$$A_k^{(1)} = A_k^{(2)} = 0 \Rightarrow p^* = p_{k-1}. \quad (11)$$

In other words, the loop terminates when the two largest remaining error clusters no longer yield a rule that passes the local gate. No maximum number of iterations was needed in our experiments: the loop stopped after four global steps for Qwen3 4B and after three for Gemma 3n (Section 3.1). Since only accepted rules are added, the returned prompt p^* is the last accepted prompt.

The analogy with iterative learning control is useful but has limits [36]. The input space is discrete text, the controller is partly human, and no gradient is available. For this reason, we do not claim formal stability or convergence guarantees. The local gate in Eq. (9) does not guarantee that the global accuracy increases: a rule can help its cluster but interfere with items outside it. In our experiments, every accepted rule increased the development accuracy, but some rejected candidates would have decreased it (Section 3.1). The main benefit of the formulation is that it makes the sensor, the controller, the update law, and the stopping rule explicit, so that each can be measured, audited, and improved separately.

Table 1 Notation and the role of each quantity in the feedback control loop

Symbol	Meaning	Role in the loop
$\mathbf{1}[\cdot], m_i(p)$	Indicator function and match indicator of item i	Item-level measurement
M_θ, Exec	Frozen SLM and Python interpreter	Plant
p_k	Prompt after iteration k (p_0 : base prompt)	Control input
$\rho_k^{(j)}$	Natural-language rule proposed in iteration k	Control action
$y(p)$	Exact match accuracy on $\mathcal{D}$ (Eq. 4)	Measured output
y^*	Maximization of y (no fixed target)	Reference
$\mathcal{E}(p)$	Set of mispredicted items (Eq. 5)	Error signal
$\phi, \text{HDBSCAN}$	Feature map and density-based clustering	Sensor / observer
$\mathcal{C}_k^{(j)}, S$	Examined error cluster (j -th largest), selection	Filtered error signal
G	Rule synthesis by a domain expert	Controller
y_k^C, π_k	Local EM and McNemar p -value on the cluster	Local verification
τ, α	Acceptance thresholds (50%, 0.0625; Eq. 9)	Controller settings
E_k^G, E_k^C	Energy of the global and cluster steps	Cost of one iteration

2.5 Error clustering

The sensor groups mispredictions that are likely due to the same root cause. Following [34], each mispredicted item is described by five categorical features derived from the structure of the tabular calculation:

- **calc_pattern**: the reference calculation from the *derivation* field of TAT-QA, with numbers replaced by a placeholder, e.g., $(\#-\#)/\#$ for a relative change; patterns occurring fewer than three times are merged into an *other* category;
- **code_calc_pattern**: the calculation actually performed by the generated program, extracted from its abstract syntax tree in the same format;
- **scale_mismatch**: whether the predicted and gold scales differ;
- **value_match**: equal value, sign error only, missing value (usually a runtime error), or different value;
- **error_type**: selection, calculation, scale, sign, syntax, or runtime error.

The features are one-hot encoded into a binary vector.

We use HDBSCAN [45, 46], a density-based hierarchical clustering algorithm. It does not require the number of clusters in advance and labels isolated points as noise rather than forcing them into a cluster. Both properties fit error analysis, because the number of failure modes is unknown and many errors are unique. We set the minimum cluster size to `min_cluster_size` = 2 and the minimum number of samples to `min_samples` = 1, so that even small groups of related errors can be found. These values were chosen by a grid search over `min_cluster_size` $\in \{2, 3, 4, 5, 6, 10, 15\}$ and `min_samples` $\in \{1, \dots, 5\}$, which selects the setting with the lowest noise ratio among those whose cluster count is close to the first quartile of the error set size [34]; the same pair was selected in every iteration. As distance, we use the normalized Hamming distance, i.e., the fraction of vector positions in which two items differ. On binary vectors,

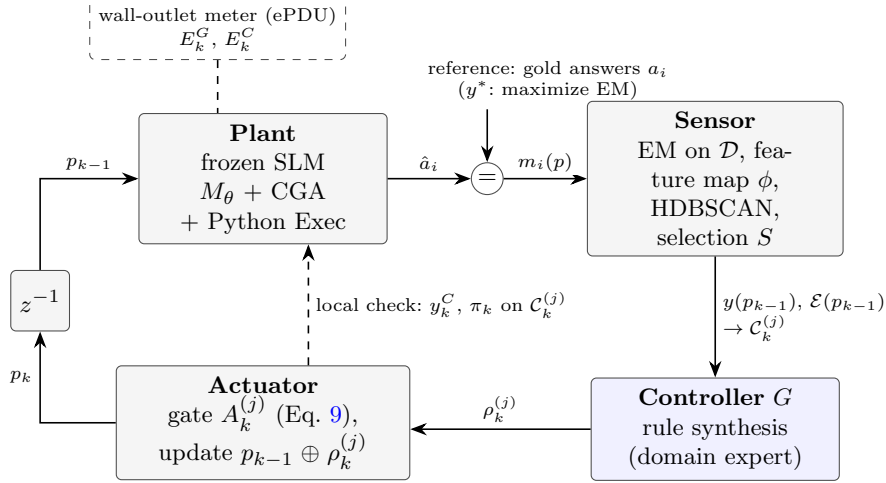


Fig. 2 The prompt optimization procedure as a feedback control loop (notation in Table 1). In each iteration, the plant answers the development set with prompt p_{k-1} ; the comparator matches the executed answers with the gold answers, and the sensor turns the mispredicted items into error clusters. The controller writes a rule for the examined cluster, and the actuator adds it to the prompt only if it passes the local gate, which is evaluated by running the plant on the cluster alone (dashed). The delay z^{-1} separates the iterations. The energy of every global and cluster step is measured at the wall outlet of the workstation (Section 2.9)

it is proportional to the Manhattan (L_1) distance, so the two give the same clustering up to a scaling of the distance threshold. The distance has a direct interpretation: it counts the error attributes in which two mispredictions differ.

2.6 Rule synthesis with a human in the loop

During rule synthesis, a domain expert studies the representative items of the selected cluster and formulates one short rule in natural language that targets their shared cause. The error attributes usually reveal this cause directly. A typical case is a cluster in which every calculation is a percentage change but the predicted scale is not percent; the corresponding rule tells the agent to express such results as percentages. When the model lacks a business formula, for example how an average over several years is calculated, the expert states the formula in the rule. Most clusters had a single cause; for a cluster with two causes, we wrote one rule for each.

This step takes about 0.4 to 0.5 person-hours per candidate rule, which gives about 2 to 3 person-hours for a full optimization run with 5 to 6 candidate rules. Human effort is not included in the energy figures, but it is included in the operational cost analysis in Section 3.6. Because every rule is written in plain language, the complete adaptation of the model can be read and reviewed by an auditor, which is an important requirement in regulated sectors.

2.7 Adaptive iterative optimization procedure

Fig. 3 and Algorithm 1 show how the loop is executed in practice. Each iteration consists of one global step and one or two cluster steps, with very different costs.

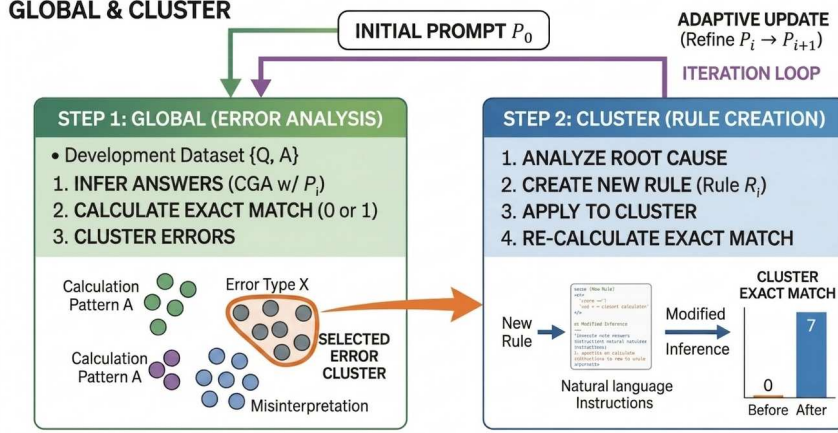


Fig. 3 The adaptive iterative prompt optimization procedure: each iteration consists of a global step (G) and one or two cluster-level steps (C). Reproduced from the conference version of this study [35].

In the *global step* (G), the CGA answers every question of the development set with the current prompt p_{k-1} . We compute the EM value of each item, the global accuracy $y(p_{k-1})$, and the error set $\mathcal{E}(p_{k-1})$, and we cluster the errors. In the *cluster step* (C), the candidate rule $\rho_k^{(j)}$ is added to the prompt, and the CGA answers only the questions of the examined cluster. This gives the local accuracy y_k^C and the McNemar p -value π_k . The global step processes all $N = 497$ items, while a cluster step processes only 6 to 34 items in our experiments. A global step is run only for an accepted prompt, because a rejected rule never enters the prompt. Iterations continue until the stopping criterion is met.

The total energy of the prompt optimization is the sum of the energy of all steps:

$$E_{PO} = \sum_{k=1}^K \left(E_k^G + \sum_{j=1}^{J'_k} E_{k,j}^C \right), \quad (12)$$

where K is the number of global steps and $J'_k \in \{1, 2\}$ is the number of cluster steps examined in iteration k .

Algorithm 1 Error-driven prompt optimization loop

Require: development set $\mathcal{D}$, frozen model M_θ , base prompt p_0 , feature map ϕ , thresholds τ, α

Ensure: optimized prompt p^* and energy $\log \{E_k^G, E_{k,j}^C\}$

```
1:  $k \leftarrow 1$ 
2: loop
3:   Global step  $G_k$ : run the CGA with  $p_{k-1}$  on  $\mathcal{D}$ ; compute  $y(p_{k-1})$  and  $\mathcal{E}(p_{k-1})$ ;
   record  $E_k^G$ 
4:   cluster  $\phi(\mathcal{E}(p_{k-1}))$  with HDBSCAN and sort the clusters by decreasing size
5:    $accepted \leftarrow \text{false}$ 
6:   for  $j = 1, 2$  do
7:     Rule synthesis: the expert writes a candidate rule  $\rho_k^{(j)}$  for  $\mathcal{C}_k^{(j)}$ 
8:     Cluster step  $C_{k,j}$ : run the CGA with  $p_{k-1} \oplus \rho_k^{(j)}$  on  $\mathcal{C}_k^{(j)}$ ; compute  $y_k^C$ 
   and  $\pi_k$ ; record  $E_{k,j}^C$ 
9:     if  $y_k^C \geq \tau$  and  $\pi_k \leq \alpha$  then
10:        $p_k \leftarrow p_{k-1} \oplus \rho_k^{(j)}$ ;  $accepted \leftarrow \text{true}$ ; break
11:     end if
12:   end for
13:   if not  $accepted$  then
14:     return  $p^* \leftarrow p_{k-1}$   $\triangleright$  stopping criterion, Eq. (11)
15:   end if
16:    $k \leftarrow k + 1$ 
17: end loop
```

2.8 Energy estimation for gradient-based fine-tuning

We did not fine-tune Qwen3 4B/30B or Gemma 3n ourselves. To compare the two adaptation paradigms on the same models, we estimated the energy that fine-tuning would have required. The estimate combines the way training compute grows with model size [4] with a model of the power drawn by the training hardware [7]. The estimate is anchored to a known measurement: the LoRA-based TAT-LLM [47], built on Llama 2 (13B, 70B) and MATATA model, built on Phi-3-mini (3.8B parameters) [48], was trained on TAT-QA for 10 epochs in 16 hours on one NVIDIA A100 40 GB GPU [49]. This gives the reference values $H_{\text{ref}} = 16 \text{ h}$, $P_{\text{ref}} = 3.8 \text{ B}$ and $E_{\text{ref}} = 10$ epochs.

The energy of a training run is

$$E_{\text{Wh}} = H_{\text{target}} \cdot \text{TDP}_{\text{gpu}} \cdot \text{PUE}, \quad (13)$$

where TDP_{gpu} is the thermal design power of the GPU in watts, that is, the maximum sustained power the GPU is designed to draw, and PUE is the power usage effectiveness of the data center, which accounts for cooling and other infrastructure. The training time on the target configuration is obtained by scaling the reference time:

$$H_{target} = \frac{P \cdot E_p \cdot M_{method} \cdot M_{arch}}{R_{perf} \cdot P_{ref} \cdot E_{ref}}. \quad (14)$$

where P is the number of model parameters, E_p is the number of epochs, M_{method} is the training method multiplier, M_{arch} is the architecture multiplier and R_{perf} is the throughput of the target GPU relative to the A100. The ratios P/P_{ref} and E_p/E_{ref} scale the computational load linearly with model size and training length. Substituting Eq. (14) into Eq. (13) gives

$$E_{Wh} = \frac{P \cdot E_p \cdot M_{method} \cdot M_{arch} \cdot TDP_{gpu} \cdot PUE}{R_{perf} \cdot P_{ref} \cdot E_{ref}}. \quad (15)$$

The coefficients follow common practice in LLM energy auditing [8]. We set $M_{method} = 1.6$ for full fine-tuning and $M_{method} = 1.0$ for LoRA. We set $M_{arch} = 1.0$ for dense models and $M_{arch} = 0.25$ for MatFormer (MF) models. The resulting training times and energy estimates for all models, alongside their specific parameters (P and M_{arch}), are summarized in Table 2.

Table 2 Estimated energy consumption of gradient-based fine-tuning on one NVIDIA A100 40 GB GPU (Eq. 15) including model-specific parameters. The Phi-3-mini LoRA row is the measured reference

Model	P (B)	M_{arch}	Method	Epochs	Time (h)	Energy (Wh)
Gemma 3n	4.0	0.25	Full	3	1.52	666.95
Gemma 3n	4.0	0.25	LoRA	10	3.16	1,389.47
Qwen3 4B	4.0	1.00	Full	3	8.08	3,557.05
Qwen3 4B	4.0	1.00	LoRA	10	16.84	7,410.53
Qwen3 30B	30.5	0.25	Full	3	15.16	6,669.47
Qwen3 30B	30.5	0.25	LoRA	10	31.58	13,894.74
Phi-3-mini	3.8	1.00	Full	3	7.68	3,379.20
Phi-3-mini (reference)	3.8	1.00	LoRA	10	16.00	7,040.00
Llama 2 13B	13.0	1.00	Full	3	26.27	11,560.42
Llama 2 13B	13.0	1.00	LoRA	10	54.74	24,084.21
Llama 2 70B	70.0	1.00	Full	3	141.47	62,248.42
Llama 2 70B	70.0	1.00	LoRA	10	294.74	129,684.21

Such models, particularly the “n” (nano/edge) variants, are fundamentally built on the Matryoshka Transformer (MatFormer) architecture and Per-Layer Embeddings (PLE) technology. Although their internal mechanisms include Mixture-of-Experts (MoE) inspired tiny expert networks (nanoMoE) for memory-efficient routing, their core structural innovation and unique composition are driven by MatFormer and PLE. Specifically, the nested, granular structure of Matryoshka Transformers allows the model to operate with as little as a quarter of the Feed-Forward Network capacity during elastic inference, significantly reducing active parameters and energy costs [50]. For the power usage effectiveness we take $PUE = 1.1$, a value reached by current energy-efficient data centers. For the A100 we use $TDP = 400$ W and $R_{perf} = 1.0$. For other hardware, the same formula applies with its own values; for example, an H200

GPU would be described with $TDP = 700 \text{ W}$ and $R_{\text{perf}} \approx 3.1$. We use 3 epochs for full fine-tuning and 10 epochs for LoRA, a choice justified by the restricted parameter space of Low-Rank Adaptation, which necessitates more iterations to achieve optimal convergence, whereas the full capacity of standard fine-tuning allows for rapid adaptation [51].

The values for Qwen3 4B and Gemma 3n are counterfactual estimates. These models were used only in our prompt optimization framework and were never fine-tuned. The estimates answer the question: what would it have cost to adapt the same model with gradient-based training? This keeps the base model fixed and isolates the adaptation paradigm as the only variable.

2.9 Direct energy measurement of prompt optimization

The energy of prompt optimization was measured directly at the workstation’s wall outlet that runs the pipeline. Figure 4 shows the measurement setup. The workstation has one NVIDIA GeForce RTX 3090 GPU (24 GB memory, 350 W power limit, driver 575.57), an AMD Ryzen 5 5600X 6-core CPU, 64 GB RAM, and runs Ubuntu 20.04.6 LTS. It is supplied by a metered outlet of an Eaton EMAB22 ePDU G3 managed rack power distribution unit (0U, C20 16 A single-phase inlet, $20 \times \text{C13}$ and $4 \times \text{C19}$ outlets). Its metering of voltage, current, active power, and energy has IEC Class 1 ($\pm 1\%$) billing-grade accuracy. The outlet readings are collected through SNMP by a Prometheus time-series database and queried with a resolution of 2 s. The power meter provides two signals: an integer counter that accumulates energy in Wh and a gauge that reports the outlet’s instantaneous power. The wall-outlet values, therefore, include the entire workstation (GPU, CPU, memory, storage, cooling fans, and power-supply losses), not just the GPU. During a run, no other workload was executed on the workstation, and the model was loaded into GPU memory before the first measured step.

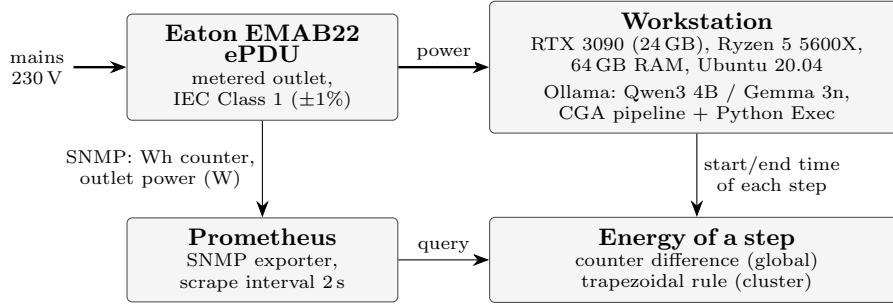


Fig. 4 Energy measurement setup. The workstation is supplied by a metered outlet of the ePDU. The outlet energy counter and power are collected by Prometheus through SNMP. The pipeline logs the start and end time of each global and cluster step, and the energy of the step is computed from the stored series for that interval (Eq. (16))

For the global steps, which last about 12 to 40 minutes, we use the difference of the counter values at the start and the end of the step. The counter has integer resolution,

but over such long intervals, the rounding error is small compared to the total. The cluster steps last only a few minutes or less, so the counter resolution is too coarse. For these steps, we integrate the sampled power signal with the trapezoidal rule. For each sub-interval $[t_j, t_{j+1}]$ the energy increment is

$$E_j = \int_{t_j}^{t_{j+1}} P(t) dt \approx (t_{j+1} - t_j) \frac{P(t_j) + P(t_{j+1})}{2}, \quad (16)$$

and the energy of the step is the sum of all increments. In effect, the power is taken to change linearly between two consecutive samples. The trapezoidal rule is robust for non-periodic signals sampled at a high rate and is cheaper to compute than higher-order spline methods. For Qwen3 4B, all steps are short enough to be computed with Eq. (16).

Although Prometheus stores a sample every 2 s, the ePDU updates its outlet power reading only about every 5 to 6 s (median interval between value changes: 6 s over all runs). The cluster steps of Qwen3 4B last 6 to 15 s and are therefore covered by only 4 to 8 samples. The integrated energy is nevertheless repeatable: for example, the 10 runs of cluster step C2 gave 1.399 ± 0.015 Wh (Fig. 8). Moreover, the cluster steps account for only a few percent of the total energy (Table 7), so a sampling error in them has little effect on the totals.

To approximate the GPU share, we subtract a constant baseline of 54 W, multiplied by the run duration, from the wall-outlet energy. The baseline is the workstation’s average power measured over one day at 0% GPU load. This assumes the other components’ power does not depend on the load, so the GPU-only values are approximate.

We measured every step 10 times. Since decoding is deterministic, the predictions and the optimization path were the same in all runs, so the spread of the repeated measurements reflects variation in the power draw of the system rather than in the algorithm. To obtain stable repeated measurements, the recorded optimization trajectory was replayed: each repetition executed the same sequence of global and cluster steps, using the same prompts and clusters as the original optimization run. Tables 3 and 4 list the measured values for each step, and Table 7 summarizes them.

Table 3 Measured energy of each prompt optimization step of Gemma 3n, averaged over 10 runs. Global steps from the energy counter, cluster steps from trapezoidal integration

Step	Prompt	Energy (Wh)	Δt (min)	N	Avg. power (W)	s/item	Wh/item	EM (%)	Gate
G1	V18	244.800	38.227	497	384.9	4.615	0.493	41.65	
C1	V19a	10.852	1.723	19	377.8	5.442	0.571	89.47	✓
G2	V19a	252.700	39.442	497	385.4	4.762	0.508	45.88	
C2a	V20	10.092	1.592	18	380.4	5.306	0.561	5.56	×
C2b	V21	7.552	1.196	14	378.9	5.126	0.539	64.29	✓
G3	V21	256.500	40.013	497	385.6	4.831	0.516	50.10	
C3a	V22b	10.964	1.707	19	385.3	5.391	0.577	0.00	×
C3b	V23	8.127	1.288	14	378.7	5.518	0.581	7.14	×

Table 4 Measured energy of each prompt optimization step of Qwen3 4B, averaged over 10 runs (all values from trapezoidal integration)

Step	Prompt	Energy (Wh)	Δt (min)	N	Avg. power (W)	s/item	Wh/item	EM (%)	Gate
G1	V18	82.823	11.787	497	421.6	1.423	0.167	59.96	
C1	V23	6.129	0.886	34	415.3	1.563	0.180	97.06	✓
G2	V23	82.026	11.645	497	422.6	1.406	0.165	64.59	
C2	V41	1.399	0.205	8	409.8	1.536	0.175	62.50	✓
G3	V41	83.173	11.804	497	422.8	1.425	0.167	67.61	
C3a	V43c	1.637	0.257	10	381.5	1.545	0.164	20.00	×
C3b	V29	0.924	0.157	6	353.5	1.568	0.154	83.33	✓
G4	V29	89.709	12.729	497	422.9	1.537	0.181	70.82	
C4a	V31	1.156	0.196	7	354.4	1.678	0.165	57.14	×
C4b	V44b	1.630	0.241	7	405.3	2.069	0.233	28.57	×

C1: the rule was evaluated on cluster 24 of the base-prompt error clustering, which contains 34 items; 33 of them were corrected in each of the 10 runs. Our previous study reported 20 corrected items out of 22 evaluated items for the same rule [34]. The energy of C1 was measured in a separate session with the same workstation, model settings and measurement chain.

In the tables, Gk denotes the global step of iteration k , and Ck (or Cka and Ckb when two candidates were examined) the cluster steps of the same iteration. The *Prompt* column gives the internal prompt version identifier of our repository [52]. The columns show the step energy, elapsed time, number of processed items N , average power, average time per item, average energy per item, and the EM value. For G steps, EM is the global development accuracy $y(p_{k-1})$. For C steps, EM is the local accuracy y_k^C on the examined cluster, and the last column shows whether the candidate rule passed the gate of Eq. (9). Global steps are listed only for the base prompt and for accepted prompts, because a rejected rule never enters the prompt (Algorithm 1). Each value is the mean of 10 runs. Table 5 summarizes the variability of the total energy across runs.

Table 5 Descriptive statistics of the total energy of the prompt optimization over 10 repeated runs (Wh)

Statistic	Gemma 3n	Qwen3 4B
n	10	10
Mean	801.587	350.606
Standard deviation	1.901	0.836
Median	801.590	350.823
IQR	0.876	0.325
Coefficient of variation	0.237%	0.238%
95% CI of the mean	[800.227, 802.947]	[350.009, 351.204]

2.10 Sensitivity and break-even analysis

The fine-tuning estimates depend on the multipliers M_{method} and M_{arch} , which are taken from auditing conventions and were not measured independently. To test how

much the conclusions depend on them, we vary both multipliers by $\pm 25\%$ around their nominal values, that is, $M_{\text{method}} \in [0.75, 1.25] \cdot M_{\text{method}}^{\text{nom}}$ and, for dense models, $M_{\text{arch}} \in [0.75, 1.25]$. For the MoE multiplier, we use the interval $M_{\text{arch}} \in [0.15, 0.35]$, which represents 15% to 35% active computational routing.

Prompt optimization makes the prompt longer, so each later inference costs slightly more energy than with a fine-tuned model that uses a short prompt. We therefore compute the number of queries after which the extra inference energy equals the energy saved during adaptation:

$$Q^* = \frac{E_{\text{FT}} - E_{\text{PO}}}{\Delta e_{\text{inf}}}, \quad (17)$$

where E_{FT} is the estimated fine-tuning energy, E_{PO} is the measured prompt optimization energy and Δe_{inf} is the additional inference energy per query caused by the longer prompt.

2.11 Reproducibility and software environment

To support the provability of the results, the entire life cycle of code generation, execution, evaluation, and energy logging runs in software containers. Containerization fixes the operating system, libraries, and their versions, so that the same pipeline gives the same predictions on another machine [53, 54]. The services are defined with Docker and orchestrated with Docker Compose. The generated Python programs are executed in a separate, isolated container, so that faulty or unsafe code cannot affect the rest of the system. Because decoding is deterministic, repeated runs give identical predictions, as described in Section 2.9. The source code of the pipeline and of the energy calculations is publicly available [52].

3 Results

In the result, we first describe the behavior of the optimization loop: accuracy 3.1, error clusters 3.2, and power 3.3. It then compares prompt optimization with fine-tuning 3.4, tests the robustness of this comparison 3.5, and translates the results into operational indicators 3.6.

3.1 Accuracy trajectory of the optimization loop

Fig. 5 shows how the global accuracy changes over the iterations. For Qwen3 4B, the loop accepted three rules and examined three more that were rejected. The development EM increased from 59.96% with the base prompt (G1) to 64.59% (G2), 67.61% (G3) and 70.82% (G4), a total gain of 10.86 pp. The first rule, which forces a percent scale for percentage-change questions, gave the largest gain (+4.63 pp). In the fourth iteration, both candidates failed the gate: one corrected 57.14% of its cluster but only four items ($\pi_4 = 0.125$), the other corrected 28.57%. The loop therefore stopped after four global steps and returned the prompt of G4. For Gemma 3n, the loop accepted two rules and rejected three. The development EM increased from 41.65% (G1) to 45.88% (G2) and 50.10% (G3), a gain of 8.45 pp, and the loop stopped after three global steps when both candidates of the third iteration corrected at most one item of their clusters.

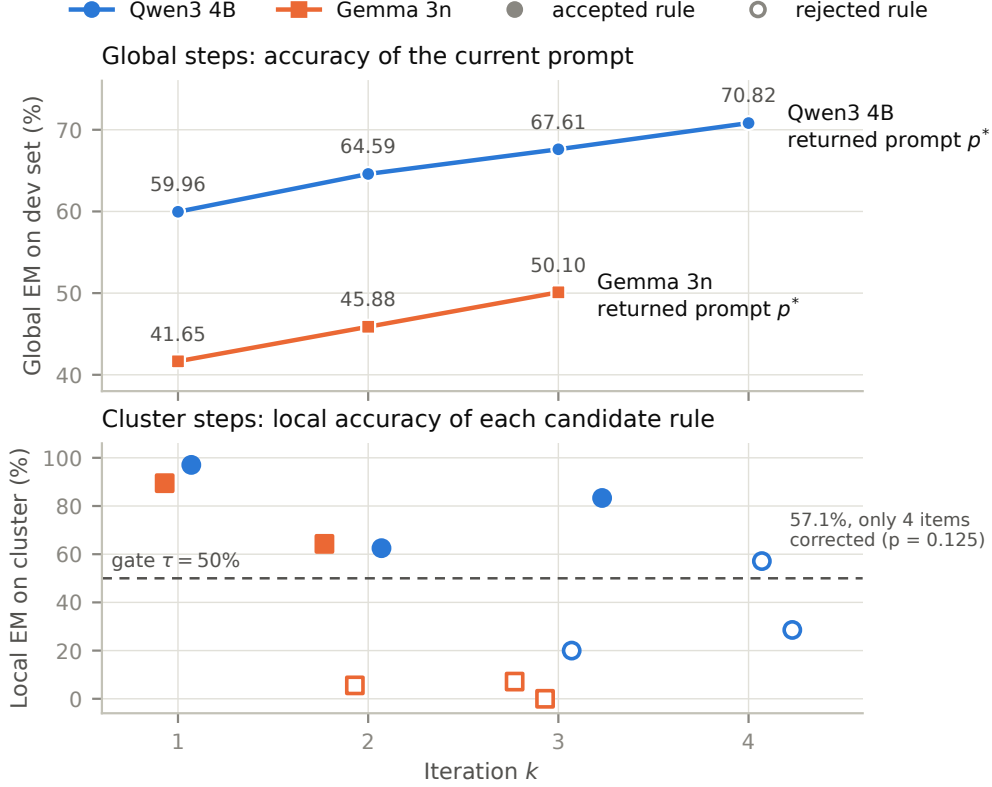


Fig. 5 Convergence of the optimization loop on the development set. Top: global exact match accuracy $y(p_{k-1})$ of the current prompt in each global step; the last point is the returned prompt p^* . Bottom: local accuracy y_k^C of each candidate rule on its error cluster; filled markers are accepted rules, hollow markers are rejected rules. The dashed line is the threshold τ of the gate in Eq. (9); a rule must also correct at least five items ($\pi_k \leq 0.0625$)

Because only accepted rules enter the prompt, the development accuracy increased monotonically in both runs. The gate was effective in filtering harmful rules: in our previous study, the rejected candidates were also evaluated on the whole development set, and each of them would have lowered the accuracy, by 0.21 to 1.61 pp for Qwen3 4B and by 0.61 to 2.01 pp for Gemma 3n [34].

The final prompts were evaluated once on the held-out test set. The test EM of Qwen3 4B increased from 56.69% with the base prompt to 66.45% with the optimized prompt, and that of Gemma 3n from 40.13% to 49.89%; both gains (9.76 pp) are significant according to the McNemar test ($p < 10^{-5}$) [34]. The test gains are smaller than the development gains for Qwen3 4B (10.86 pp), which indicates a slight overfitting of the rules to the development set. In the rest of the article, development values are used for the behavior of the loop and test values for the comparison with other systems.

The local accuracy of the candidate rules varied widely, from 0% to 89.47% for Gemma 3n and from 20.00% to 97.06% for Qwen3 4B. The gate requires both a high local accuracy and a minimum number of corrected items, so a rule that corrects most items of a very small cluster is not accepted on the basis of a few lucky corrections.

3.2 Evolution of the error clusters

Fig. 6 shows how the mispredictions are distributed over the error types in each global step, together with the number of HDBSCAN clusters and noise points.

Qwen3 4B					Gemma 3n		
	G1	G2	G3	G4	G1	G2	G3
	59.96%	64.59%	67.61%	70.82%	41.65%	45.88%	50.10%
Scale	79 ^{✓34}	31	33	30	75 ^{✓19}	43 ^{✓14}	29
Calculation	28	52 ^{✓8}	32	33	30	24	29
Selection	52	50	54 ^{✓6}	47 ^{×7}	136	130 ^{×18}	138 ^{×19 ×14}
Sign	19	18	18 ^{×10}	16	11	13	11
Runtime	21	23	24	23 ^{×7}	15	18	13
Syntax	0	2	0	0	23	41	28
errors	199	176	161	149	290	269	248
clusters	45	45	44	38	65	57	56
noise	31	27	16	27	35	43	33

✓n / ×n: examined cluster of n items whose rule was accepted / rejected (solid / dashed outline)

Fig. 6 Evolution of the failure modes over the global steps of the optimization loop. Cells give the number of mispredicted development items per error type (same color scale in both panels); column headers give the global development EM. Outlined cells contain the examined clusters: ✓n marks a cluster of n items whose rule was accepted (solid outline), ×n one whose rule was rejected (dashed outline). Every examined cluster contained a single error type. The rows below give the number of errors, HDBSCAN clusters and noise points of each step

In the first global step, scale errors were the largest group for both models (79 of 199 errors for Qwen3 4B and 75 of 290 for Gemma 3n), and the first examined cluster of both models consisted only of percentage-change questions answered with a wrong scale. The corresponding rule removed most of these errors: 44 scale errors of Qwen3 4B disappeared, and the number of scale errors fell from 79 to 31. The rule also had a side effect. For Qwen3 4B, 20 previously correct questions that ask for a plain difference (e.g., “What was the change in . . .”) became calculation errors, so the number of calculation errors rose from 28 to 52. The next accepted rule (“change in percentage is a subtraction”) repaired 17 of these 20 items and reduced the calculation

errors to 32. For Gemma 3n, the first rule was followed by an increase of syntax errors from 23 to 41, which fell back to 28 after the second accepted rule. Selection errors, where the generated program does not find the right values in the table, were hardly affected by the loop (47–54 for Qwen3 4B and 130–138 for Gemma 3n) and remain the largest error group at the end of both runs; the candidate rules that targeted selection, sign and runtime clusters were mostly rejected. The number of clusters decreased from 45 to 38 for Qwen3 4B and from 65 to 56 for Gemma 3n, while 16 to 43 errors per step remained unclustered noise.

3.3 Power profile and energy per step

Fig. 7 and Fig. 8 show how the power draw changes during the optimization. The global steps dominate the energy budget. They account for 96.3% of the total energy of Qwen3 4B (337.73 of 350.61 Wh) and for 94.1% of the total energy of Gemma 3n (754.00 of 801.59 Wh). The cluster steps are short and cost between 0.9 and 11.0 Wh each.

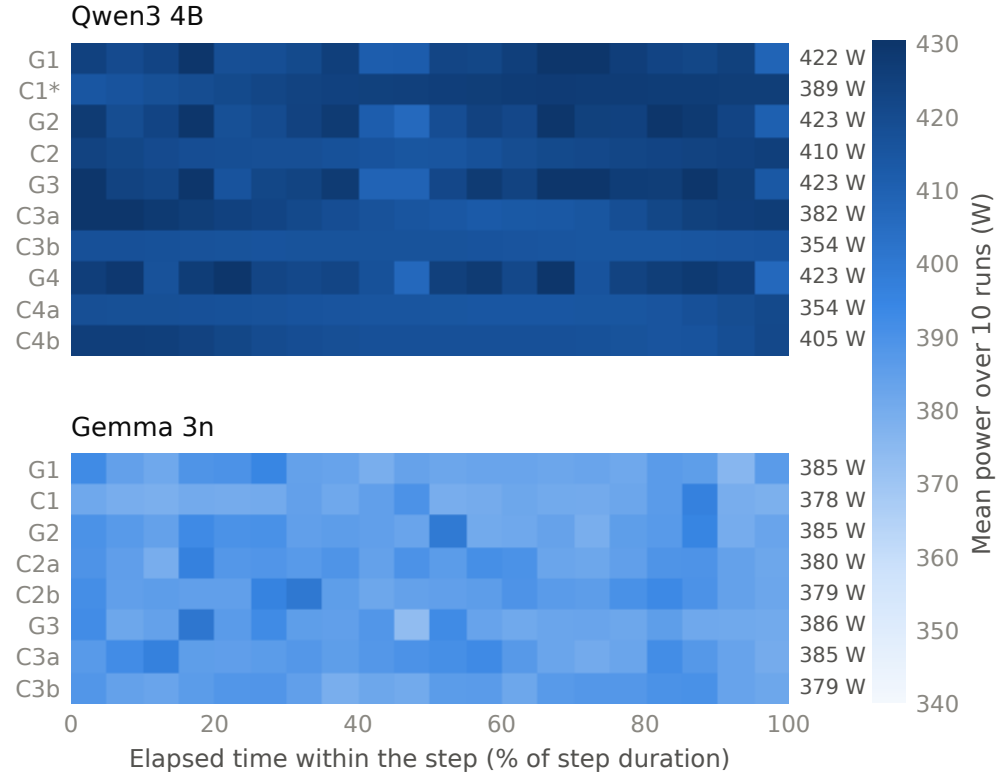


Fig. 7 Mean power of the workstation during the optimization steps (accepted path, Tables 3 and 4). Each row is one step; its duration is divided into 20 equal intervals, and each cell gives the mean outlet power in the interval, averaged over the 10 runs. Both panels use the same color scale; the mean power of each step is given on the right.

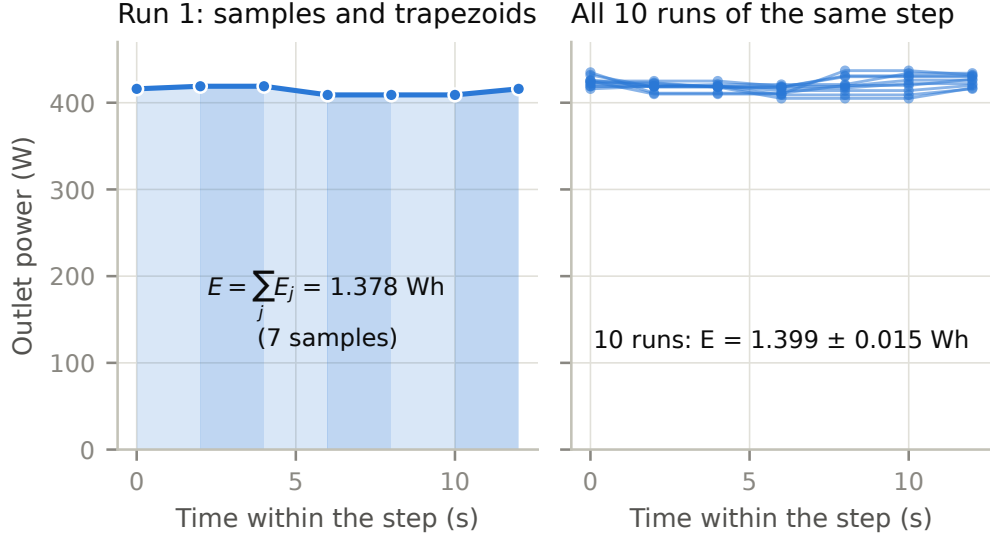


Fig. 8 Trapezoidal energy integration (Eq. (16)) for a short cluster step (Qwen3 4B, C2, 8 items). Left: outlet power samples of the first run and the trapezoids whose areas are summed. Right: the power curves of all 10 runs of the same step

The average power was very stable within each step type. During global steps, it was between 421.6 and 422.9 W for Qwen3 4B and between 384.9 and 385.6 W for Gemma 3n. The cluster steps showed larger variation (353.5–415.3 W for Qwen3 4B), probably because short steps include a larger share of start-up and idle phases. Although Qwen3 4B drew about 10% more power, it answered each question about three times faster (1.41–1.54 s versus 4.62–4.83 s per item in the global steps). As a result, its energy per item was about three times lower (0.165–0.181 Wh versus 0.493–0.516 Wh). Although the Gemma 3n run required one fewer global step, its total optimization energy was 2.3 times higher; the Qwen3 4B run used 56.3% less energy. In other words, the energy of prompt optimization depends mainly on the inference speed of the base model, while the energy of fine-tuning depends mainly on its size.

3.4 Prompt optimization versus gradient-based fine-tuning

Table 6 compares the optimized CGA with published fine-tuned models, and Table 7 summarizes the energy of the optimization runs. Figure 9 contrasts the energy of the two adaptation paradigms, and Figure 10 relates model size, energy, and accuracy.

For Qwen3 4B, the complete prompt optimization consumed 350.61 Wh. The estimated LoRA fine-tuning of the same model requires 7,410.53 Wh, which is 21.1 times more. The estimated full fine-tuning with 3 epochs requires 3,557.05 Wh, which is 10.1 times more. A single global evaluation of the development set costs on average 84.4 Wh (0.17 Wh per question), which is about 88 times less than the LoRA estimate. The fine-tuning estimates include a PUE factor, while the wall-outlet measurement of the workstation does not. If we apply the same PUE of 1.1 to the GPU-only energy

Table 6 Accuracy and adaptation energy of fine-tuned reference models and of the prompt-optimized CGA on the TAT-QA test set. FT: fine-tuning, PO: prompt optimization, MR: manually written prompt rules

Model	Base model	Method	EM	Base EM	Δ EM	Time (h)	E (Wh)	η_{acc}	η_{imp}
MATATA	Phi-3-mini 3.8B	FT	74.20	51.29	22.91	16.00	7,040	94.88	307
TAT-LLM	Llama 2 70B	FT	81.42	45.94	35.48	141.47	62,248	764.53	1754
CGA	GPT-3.5 Turbo	—	52.62	—	—	—	—	—	—
CGA	GPT-3.5 Turbo	MR	66.27	—	—	—	—	—	—
CGA	Gemma 3n	PO	49.89	40.13	9.76	2.09	802	16.07	82
CGA	Qwen3 4B	PO	66.45	56.69	9.76	0.83	351	5.28	36

η_{acc} and η_{imp} in Wh per EM point (Eqs. 1–2). The training time of TAT-LLM is derived from the training setup published by its authors [47]. CGA values are measured on our filtered test subset ($N = 471$) [32, 34]; for PO, Base EM is the test EM of the base prompt, and E and Time are those of the optimization loop on the development set.

Table 7 Energy, mean power and duration of the complete prompt optimization runs, with the share of the global steps and the development EM gain (mean of 10 runs)

Model	Energy (Wh)	Avg. power (W)	Time (h)	Global-step share	EM gain (pp)
Gemma 3n (total)	801.587	384.2	2.086	94.1%	8.45
Qwen3 4B (total)	350.606	421.5	0.832	96.3%	10.86
Gemma 3n (GPU only)	688.918	330.2	2.086	—	—
Qwen3 4B (GPU only)	305.691	367.5	0.832	—	—

Global-step share: energy of all G steps divided by the total energy. EM gain: development of EM of the final prompt minus the EM of the base prompt. GPU only: total energy minus a constant non-GPU baseline of 54 W over the run time.

of prompt optimization (305.69 Wh), we obtain 336.26 Wh, and the ratios become 22.0 (LoRA) and 10.6 (full fine-tuning). The comparison is therefore not sensitive to this choice.

For Gemma 3n, the advantage is smaller. Its optimization consumed 801.59 Wh, which is 1.7 times less than the estimated LoRA energy (1,389.47 Wh) but 1.2 times more than the estimated full fine-tuning energy (666.95 Wh, ratio 0.83). The main reason is the MatFormer multiplier $M_{arch} = 0.25$ in the estimate, which reduces the estimated training time to one quarter. If Gemma 3n is treated as a dense model ($M_{arch} = 1.0$), the estimates become 5,557.88 Wh for LoRA and 2,667.80 Wh for full fine-tuning, and the ratios become 6.9 and 3.3. The energy advantage of prompt optimization is therefore clear for fast models such as Qwen3 4B, but less so for slower models.

In terms of accuracy, the optimized Qwen3 4B agent (66.45% test EM) clearly outperforms the GPT-3.5 Turbo-based CGA without prompt rules (52.62%) [32], and it is on par with the GPT-3.5 Turbo-based CGA that uses manually written rules (66.27%) [34]. A 4-billion-parameter model running on-premises thus matches a much larger proprietary model. The fine-tuned MATATA (74.2% EM) [49] and TAT-LLM

with Llama 2 70B (81.42% EM) [47] are more accurate. However, their energy-to-accuracy ratios are 94.88 and 764.53 Wh per EM point, while the optimized CGA requires 5.28 Wh per EM point.

The energy-to-improvement ratio shows the same pattern: 36 Wh per EM point gained for prompt optimization, versus 307 and 1,754 Wh for the two fine-tuned systems. Prompt optimization thus gives a smaller absolute improvement (9.76 pp) but at a much lower energy cost per point.

3.5 Robustness of the comparison

The energy measurements were highly stable. The coefficient of variation of the total energy was 0.237% for Gemma 3n and 0.238% for Qwen3 4B, and the 95% confidence intervals of the means are narrower than 3 Wh (Table 5). The uncertainty of the comparison, therefore, comes almost entirely from the fine-tuning estimates.

With a $\pm 25\%$ variation of both multipliers, the estimated LoRA energy of Qwen3 4B ranges from 4,168.42 to 11,578.95 Wh. Even at the lower bound, prompt optimization uses 11.9 times less energy, and at the upper bound, 33.0 times less. The same variation yields a range of 2,000.84 to 5,557.89 Wh for full fine-tuning, corresponding to ratios between 5.7 and 15.9. For Gemma 3n, varying the MatFormer multiplier between 0.15 and 0.35 gives a LoRA estimate between 833.68 and 1,945.26 Wh and a full fine-tuning estimate between 400.17 and 933.73 Wh. The corresponding ratios range from 1.04 to 2.43 for LoRA and from 0.50 to 1.16 for full fine-tuning. Prompt optimization of Gemma 3n therefore needs less energy than LoRA over the whole range, but the comparison with full fine-tuning is not robust to the choice of M_{arch} . For Qwen3 4B, the conclusion that prompt optimization needs substantially less energy than both fine-tuning methods holds for the entire tested range.

3.6 Break-even point and operational indicators

For Qwen3 4B, prompt optimization saves 7,059.92 Wh compared with LoRA and 3,206.44 Wh compared with full fine-tuning. If we assume, as in the conference version, that the longer prompt adds $\Delta e_{\text{inf}} = 0.05$ Wh per query, Eq. (17) gives a break-even point of about 141,200 queries for LoRA. Our measurements suggest that this assumption is conservative. The energy per item of the global step with the final prompt (G4, 0.181 Wh) is 0.014 Wh higher than that of the base prompt (G1, 0.167 Wh). With this value, the break-even point would be about 510,000 queries. Because the energy per item also varies between global steps for reasons other than the prompt length (0.165 to 0.181 Wh), this second value should be treated as indicative only.

Table 8 translates the energy figures into indicators relevant to AIBO decision-makers: electricity costs, greenhouse gas emissions, and human effort.

For a single adaptation, the absolute numbers are small. Prompt optimization of Qwen3 4B costs about 0.06 EUR of electricity and causes about 0.07 kg CO₂e, and the saving compared with LoRA is about 1.30 EUR and 1.5 kg CO₂e. At this scale, the 2 to 3 person-hours of rule synthesis are by far the largest cost item, exceeding the electricity costs of any of the three adaptation methods; fine-tuning also requires expert effort for data preparation and training setup, which we did not quantify. The

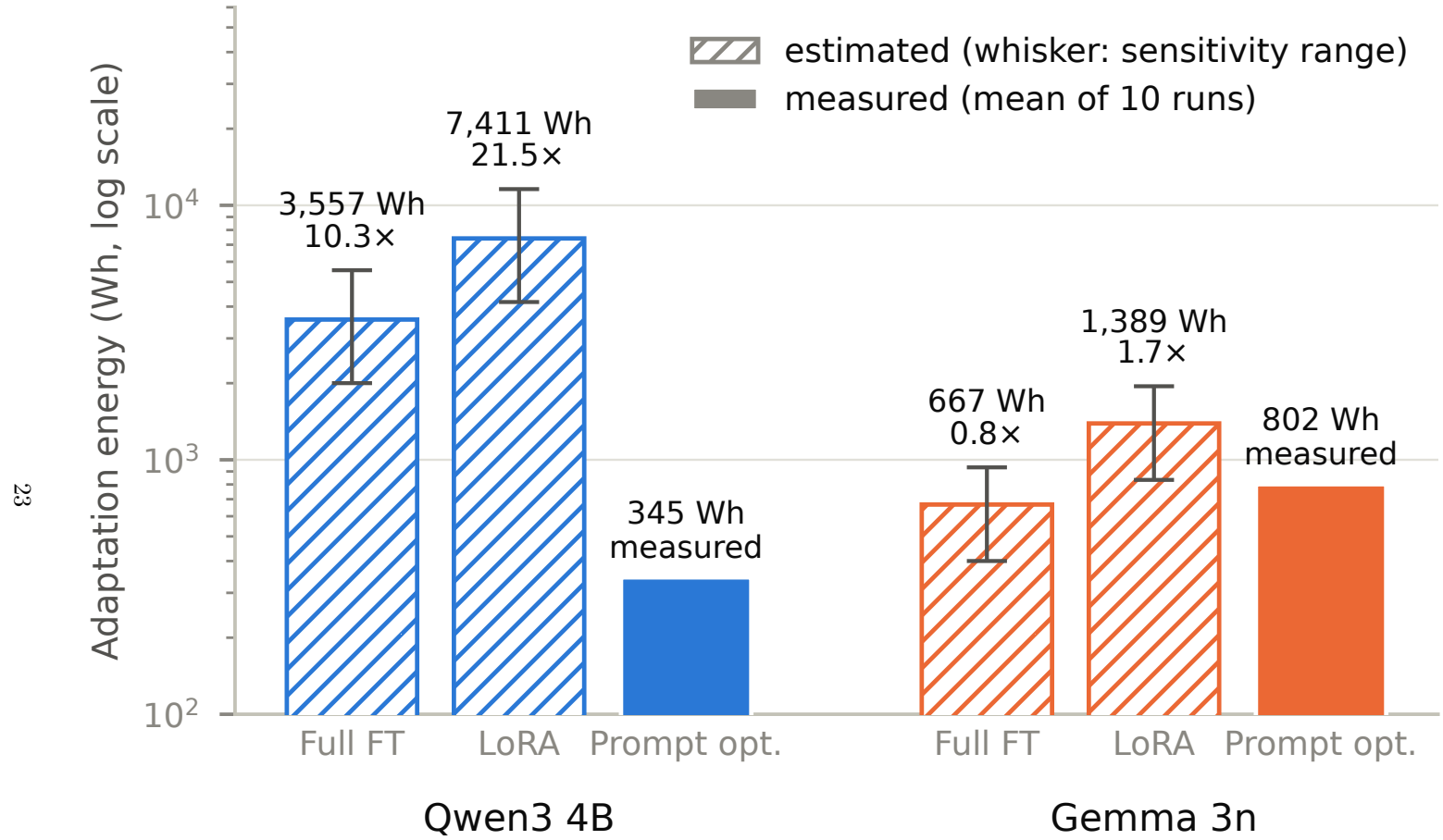


Fig. 9 Adaptation energy of full fine-tuning and LoRA (both estimated, Eq. (15)) and of prompt optimization (measured at the wall outlet, mean of 10 runs) for the two SLMs, on a logarithmic scale. Whiskers show the sensitivity range of the estimates (Section 3.5); the standard deviation of the measured values (below 2 Wh) is too small to be visible. The factors above the estimated bars give the ratio to the measured prompt optimization energy of the same model

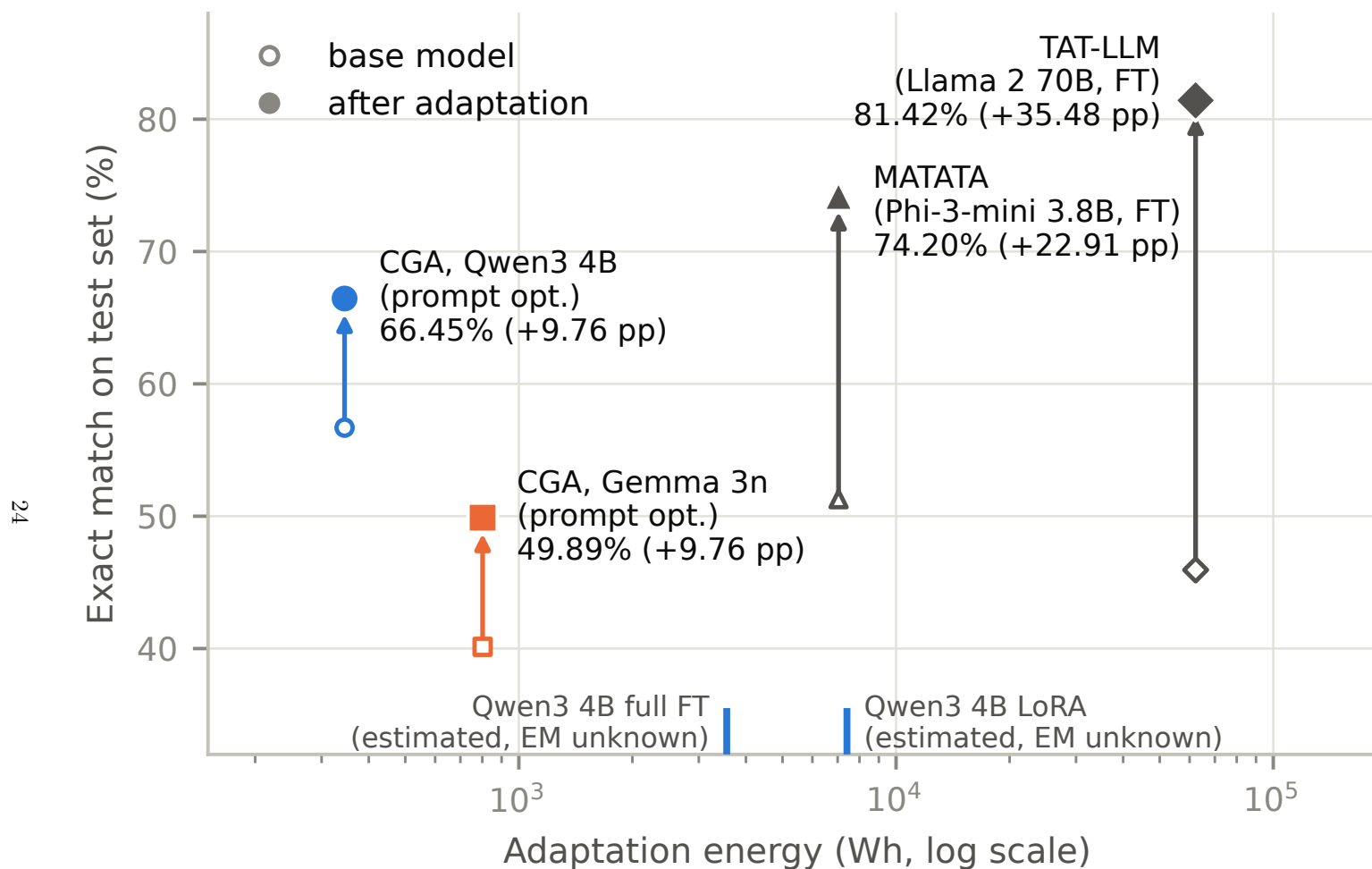


Fig. 10 Adaptation energy versus exact match accuracy on the test set for fine-tuned systems and for the prompt-optimized CGA. Hollow markers show the base model before adaptation, filled markers the adapted system; the arrows show the accuracy gain. Model sizes are given in the labels. The ticks on the energy axis are the estimated LoRA and full fine-tuning energies of Qwen3 4B, whose accuracy after fine-tuning is unknown. Energies of the fine-tuned systems are estimates (Table 6)

Table 8 Operational (OPEX) and sustainability indicators of the adaptation of Qwen3 4B

Adaptation	Source	Energy (kWh)	Cost (EUR)	CO ₂ e (kg)	Expert effort (h)
Full fine-tuning	estimated	3.557	0.65	0.747	n/m
LoRA fine-tuning	estimated	7.411	1.36	1.555	n/m
Prompt optimization	measured	0.351	0.06	0.074	2–3
Saving vs. LoRA		7.060	1.30	1.482	–
Saving vs. full FT		3.206	0.59	0.673	–
<i>Scenario: 20 adaptations per year (5 business units, quarterly re-adaptation)</i>					
Saving vs. LoRA per year		141.198	25.94	29.638	–
Saving vs. full FT per year		64.129	11.78	13.461	–

Cost = $E \times c_{el}$ with $c_{el} = 0.1837$ EUR/kWh, the EU-27 average electricity price for non-household consumers (annual consumption 500–1,999 MWh, excluding VAT, second half of 2025; Eurostat dataset `nrg-pc.205`) [55]. CO₂e = $E \times CI$ with $CI = 209.9$ g CO₂e/kWh, the carbon intensity of EU-27 electricity generation in 2025 [56]. n/m: not measured; fine-tuning additionally requires data preparation and training set-up, which we did not quantify. The fine-tuning energies are estimates (Table 2).

energy advantage becomes relevant at scale. With 20 adaptations per year, for example, quarterly re-adaptation in five business units, prompt optimization saves about 141 kWh, 26 EUR, and 30 kg CO₂e per model compared with LoRA, and organizations that maintain many models, adapt larger models, or re-adapt more often save proportionally more. Equally important for regulated companies, the energy of every adaptation is measured and documented, which supports emissions accounting and the reporting obligations discussed in Section 1. With the Hungarian non-household electricity price (0.2132 EUR/kWh) [55] and carbon intensity (163.0 g CO₂e/kWh in 2025) [56], the cost figures are 16% higher and the emission figures 22% lower.

4 Discussion and conclusion

4.1 Technological limitations

Two limitations are central. First, the rule synthesis step is still manual or semi-automated. It requires a domain expert for about 2 to 3 person-hours per run, which limits out-of-the-box use and adds a cost not captured by energy measurements. Second, the energy comparison is asymmetric. The energy of prompt optimization was measured directly, while the energy of fine-tuning was estimated with a scaling model anchored to a single empirical baseline. The multipliers of this model come from auditing conventions, and the assumed TDP of 400 W is a hardware ceiling; real training usually draws less power. Our sensitivity analysis shows that the conclusion is robust for Qwen3 4B; for Gemma 3n, it is robust against LoRA but not against full fine-tuning. The reported ratios should therefore be read as order-of-magnitude estimates. Further limitations are that we studied a subset of one benchmark, two SLMs of similar size, and a single hardware platform for inference.

4.2 Practical applicability

Because it is energy-efficient, keeps data on-premises and can be audited, the approach is well suited to regulated industries such as finance and the energy sector. The model

runs on-premises, so confidential data stays within the internal network. Every adaptation step is a readable rule that can be reviewed, approved, or removed, which supports internal control and regulatory audits. The energy of the adaptation is measured and documented, which supports the reporting duties mentioned in Section 1. For AIBO, this means that model adaptation can be managed as a transparent, controlled business process with a known energy and cost budget, rather than as an opaque training job. Our results also show when this is most beneficial: the base model should be fast at inference, because the cost of prompt optimization is dominated by repeated evaluations of the development set. The same properties make the framework a practical teaching tool for courses on the business applications of AI.

4.3 Future research directions

We see three directions for future work. First, the rule synthesis step can be automated with meta-prompting, in which a language model proposes rules from the cluster summaries and the expert only approves them. This would reduce human effort while keeping the method explainable. Second, both adaptation paradigms should be run on identical hardware and workloads, so that the comparison is fully measured rather than partly estimated. Third, since 94 to 98% of the energy is spent in global evaluation steps, the loop can be made cheaper by re-evaluating only items that a new rule is likely to affect, or by using a stratified sample of the development set. From a control perspective, a formal analysis of the convergence of the loop under such partial measurements is also an open question.

4.4 Concluding remarks

Accurate tabular reasoning, as this study shows, can be obtained without energy-intensive fine-tuning. Formulated as a feedback control loop, error-driven prompt optimization enabled a 4-billion-parameter model to match a much larger proprietary model, with an adaptation energy an order of magnitude below that of gradient-based methods for the same model. The method is precise, explainable and provable, which makes it a practical building block for sustainable, responsible AI-based business optimization.

Declarations

- Funding: This research received no specific funding.
- Competing interests: The authors declare no competing interests.
- Ethics approval and consent to participate: Not applicable.
- Consent for publication: Not applicable.
- Availability of data and materials: The TAT-QA benchmark dataset [41] used in this study is publicly available.
- Materials availability: Not applicable.
- Code availability: The full implementation, including the energy calculations, is available in our public repository at <https://github.com/apandyud/slm.prompt.opt.energy> [52].

- Authors’ contributions: R.L.: conceptualization, methodology, software, measurements, writing original draft, and formal analysis. Á.P.: methodology, software, data curation, measurements, writing original draft, and formal analysis. A.H.: conceptualization, supervision, writing review, and editing. All authors read and approved the final manuscript.
- Use of AI tools: The authors used generative artificial intelligence tools, specifically Gemini (Google), ChatGPT (OpenAI), Claude (Anthropic), and Grammarly, in preparing this manuscript and the associated research. The core concepts, research methodology, and scientific contributions of this paper constitute original work by the authors. All AI-generated outputs, including text refinements, source code fragments, and visual elements, were produced under human supervision, hand-validated, and critically reviewed and edited by the authors. The authors remain fully responsible for the accuracy, integrity, and originality of the final content.
- Prior publication: This article extends the conference paper [35] (R. Lakatos, Á. Pándy, and A. Hajdu, ”Error-driven prompt optimization for sustainable tabular reasoning”). In the conference version of the present study, we compared the energy of this framework with fine-tuning. The present article extends that comparison in four directions:
 - a control-theoretic formulation of the method.
 - a detailed description of the measurement protocol and software environment.
 - a broader analysis of the results.
 - a translation of the energy figures into operational indicators for business decision makers.

Acknowledgements. This research was supported by the Doctoral School of Informatics at the University of Debrecen.

References

- [1] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Nee-lakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.*: Language models are few-shot learners. In: Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901 (2020)
- [2] Davenport, T.H., Ronanki, R.: Artificial intelligence for the real world. *Harvard Business Review* **96**(1), 108–116 (2018)
- [3] Dwivedi, Y.K., Hughes, L., Ismagilova, E., Aarts, G., Coombs, C., Crick, T., Duan, Y., Dwivedi, R., Edwards, J., Eirug, A., *et al.*: Artificial intelligence (AI): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management* **57**, 101994 (2021)
- [4] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B.: Rewon child, scott gray, alec radford, jeffrey wu, and dario amodei. 2020. scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* **2** (2020)

- [5] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Las Casas, D., Hendricks, L.A., Welbl, J., Clark, A., *et al.*: Training compute-optimal large language models. In: *Advances in Neural Information Processing Systems*, vol. 35, pp. 30016–30030 (2022)
- [6] Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650 (2019)
- [7] Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., Dean, J.: Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021)
- [8] Luccioni, A.S., Viguier, S., Ligozat, A.-L.: Estimating the carbon footprint of bloom, a 176b parameter language model. *Journal of machine learning research* **24**(253), 1–15 (2023)
- [9] Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C., *et al.*: Sustainable AI: Environmental implications, challenges and opportunities. In: *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813 (2022)
- [10] Luccioni, S., Jernite, Y., Strubell, E.: Power hungry processing: Watts driving the cost of AI deployment? In: *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, pp. 85–99 (2024)
- [11] Samsi, S., Zhao, D., McDonald, J., Li, B., Michaleas, A., Jones, M., Bergeron, W., Kepner, J., Tiwari, D., Gadepally, V.: From words to watts: Benchmarking the energy costs of large language model inference. In: *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9 (2023). IEEE
- [12] Masanet, E., Shehabi, A., Lei, N., Smith, S., Koomey, J.: Recalibrating global data center energy-use estimates. *Science* **367**(6481), 984–986 (2020)
- [13] International Energy Agency: Energy and AI. World energy outlook special report, IEA, Paris (2025)
- [14] Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green ai. *Communications of the ACM* **63**(12), 54–63 (2020)
- [15] Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., Pineau, J.: Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research* **21**(248), 1–43 (2020)
- [16] Lacoste, A., Luccioni, A., Schmidt, V., Dandres, T.: Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700* (2019)

- [17] European Parliament and Council of the European Union: Regulation (EU) 2021/1119 of the European Parliament and of the Council of 30 June 2021 establishing the framework for achieving climate neutrality (European Climate Law). Official Journal of the European Union, L 243, 9 July 2021, pp. 1–17 (2021)
- [18] European Parliament and Council of the European Union: Directive (EU) 2023/1791 of the European Parliament and of the Council of 13 September 2023 on energy efficiency and amending Regulation (EU) 2023/955 (recast). Official Journal of the European Union, L 231, 20 September 2023 (2023)
- [19] European Parliament and Council of the European Union: Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L series, 12 July 2024 (2024)
- [20] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., *et al.*: Lora: Low-rank adaptation of large language models. *Iclr* **1**(2), 3 (2022)
- [21] Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: QLoRA: Efficient finetuning of quantized LLMs. In: *Advances in Neural Information Processing Systems*, vol. 36 (2023)
- [22] Lester, B., Al-Rfou, R., Constant, N.: The power of scale for parameter-efficient prompt tuning. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059 (2021)
- [23] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys* **55**(9), 1–35 (2023)
- [24] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. In: *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837 (2022)
- [25] Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., Neubig, G.: PAL: Program-aided language models. In: *Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research*, vol. 202, pp. 10764–10799 (2023)
- [26] Chen, W., Ma, X., Wang, X., Cohen, W.W.: Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research* (2023)
- [27] Zhou, Y., Muresanu, A.I., Han, Z., Paster, K., Pitis, S., Chan, H., Ba, J.: Large language models are human-level prompt engineers. In: *International Conference on Learning Representations (ICLR)* (2023)

- [28] Pryzant, R., Iter, D., Li, J., Lee, Y.T., Zhu, C., Zeng, M.: Automatic prompt optimization with “gradient descent” and beam search. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 7957–7968 (2023)
- [29] Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q.V., Zhou, D., Chen, X.: Large language models as optimizers. In: International Conference on Learning Representations (ICLR) (2024)
- [30] Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T.T., Moazam, H., Miller, H., Zaharia, M., Potts, C.: DSPy: Compiling declarative language model calls into self-improving pipelines. In: International Conference on Learning Representations (ICLR) (2024)
- [31] Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., Alon, U., Dziri, N., Prabhunoye, S., Yang, Y., *et al.*: Self-refine: Iterative refinement with self-feedback. In: Advances in Neural Information Processing Systems, vol. 36 (2023)
- [32] Pándy, Á., Lakatos, R., Hajdu, A.: Arithmetic-aware question-answering on tabular data using a large language model-based code generation agent. In: 2025 IEEE 29th International Conference on Intelligent Engineering Systems (INES), pp. 000083–000088 (2025). IEEE
- [33] Pandya, A., Lakatos, R., Hajdu, A.: Closing the gap: Small language model-based agent for accurate and privacy-compliant tabular arithmetic. In: 1st IEEE International Conference on Crisp and Soft Computing in AI, Modeling and Control (CSC-AIMC) (2025). in press
- [34] Pándy, Á., Lakatos, R., Hajdu, A.: Error-driven prompt optimization for arithmetic reasoning: A code generation approach using on-premises small language models on tabular data. IEEE Access **14**, 62570–62583 (2026) <https://doi.org/10.1109/ACCESS.2026.3685125>
- [35] Lakatos, R., Pándy, Á., Hajdu, A.: Error-driven prompt optimization for sustainable tabular reasoning. In: 2026 IEEE 4th Conference on Information Technology and Data Science (CITDS), pp. 251–256 (2026). <https://doi.org/10.1109/CITDS70571.2026.11706920>
- [36] Bristow, D.A., Tharayil, M., Alleyne, A.G.: A survey of iterative learning control. IEEE Control Systems Magazine **26**(3), 96–114 (2006)
- [37] Åström, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers, 2nd edn. Princeton University Press, Princeton, NJ (2021)

- [38] Wu, X., Xiao, L., Sun, Y., Zhang, J., Ma, T., He, L.: A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems* **135**, 364–381 (2022)
- [39] Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., *et al.*: ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* **103**, 102274 (2023)
- [40] McKinney, W.: Data structures for statistical computing in Python. In: *Proceedings of the 9th Python in Science Conference*, pp. 56–61 (2010)
- [41] Zhu, F., Lei, W., Huang, Y., Wang, C., Zhang, S., Lv, J., Feng, F., Chua, T.-S.: Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (volume 1: Long Papers)*, pp. 3277–3287 (2021)
- [42] Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., *et al.*: Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025)
- [43] Gemma Team: Gemma 3 technical report. *arXiv preprint arXiv:2503.19786* (2025). Gemma 3n is described in the Google model card; consider citing the model card as well
- [44] McNemar, Q.: Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* **12**(2), 153–157 (1947) <https://doi.org/10.1007/BF02295996>
- [45] Campello, R.J.G.B., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: *Advances in Knowledge Discovery and Data Mining (PAKDD 2013)*. *Lecture Notes in Computer Science*, vol. 7819, pp. 160–172. Springer, ??? (2013)
- [46] McInnes, L., Healy, J., Astels, S.: hdbscan: Hierarchical density based clustering. *Journal of Open Source Software* **2**(11), 205 (2017)
- [47] Zhu, F., Liu, Z., Feng, F., Wang, C., Li, M., Chua, T.S.: Tat-llm: A specialized language model for discrete reasoning over financial tabular and textual data. In: *Proceedings of the 5th ACM International Conference on AI in Finance*, pp. 310–318 (2024)
- [48] Abdin, M., Jacobs, S.A., Awan, A.A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., *et al.*: Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219* (2024) [arXiv:2404.14219](https://arxiv.org/abs/2404.14219) [cs.CL]

- [49] Vinayagame, V., Senay, G., Martí, L.: Matata: Weakly supervised end-to-end mathematical tool-augmented reasoning for tabular applications. In: International Conference on Document Analysis and Recognition, pp. 428–467 (2025). Springer
- [50] Kudugunta, S., Kusupati, A., Dettmers, T., Chen, K., Dhillon, I., Tsvetkov, Y., Hajishirzi, H., Kakade, S., Farhadi, A., Jain, P.: Matformer: Nested transformer for elastic inference. *Advances in Neural Information Processing Systems* **37**, 140535–140564 (2024)
- [51] Zhao, J., Gao, Y.: A survey on lora of large language models. *Frontiers of Computer Science* **19**(7), 197605 (2025)
- [52] Pándy, Á.: Codgen Agent. Accessed: 20260510 (2026). https://github.com/apandyud/slm_prompt_opt_energy
- [53] Boettiger, C.: An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* **49**(1), 71–79 (2015)
- [54] Merkel, D.: Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal* **2014**(239), 2 (2014)
- [55] Eurostat: Electricity prices for non-household consumers – bi-annual data (from 2007 onwards) (nrg_pc_205). Data for the second half of 2025, updated 9 October 2026 (2026). https://ec.europa.eu/eurostat/databrowser/view/nrg_pc_205/default/table
- [56] Ember: Yearly electricity data: carbon intensity of electricity generation. Processed by Our World in Data, accessed 12 October 2026 (2026). <https://ourworldindata.org/grapher/carbon-intensity-electricity>