Energy-Aware Benchmarking of Generative Information Extraction

Andrei Politov* **and Iryna Vergunova and René Jäkel and Michael Färber**
Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI),
TU Dresden, Dresden/Leipzig, Germany
andrei.politov@tu-dresden.de

Abstract

Using Large Language Models (LLMs) for Information Extraction (IE) inherently involves transforming diverse textual inputs into strictly formatted outputs. Guaranteeing this structural validity introduces a hidden energy tax driven by constrained decoding mechanisms, serialisation formats, and underlying serving configurations. Consequently, standard token-level metrics fail to capture the true energy cost of generative IE, which also cannot be predicted by model size alone. To address this, we introduce the first dedicated benchmark evaluating the energy-accuracy trade-offs of generative IE. Our framework establishes task-aligned energy metrics, including Energy per True Positive (J/TP) and the F1-Energy Ratio (F_1/kJ). Through comprehensive ablation across architectures, decoding regimes, and serving-layer limits, we demonstrate that efficiency is governed by complex parameter interactions. We translate these findings into actionable configuration guidelines, equipping practitioners to maximise predictive accuracy, optimise energy yield, or achieve a sustainable Pareto balance. The source code and evaluation results are available on GitHub[1].

1 Introduction

Information Extraction (IE) is rapidly shifting from highly efficient, fine-tuned encoders to zero-shot Large Language Models (LLMs) (Ni et al., 2025). While LLMs offer immense flexibility, their widespread deployment raises serious environmental concerns (Schwartz et al., 2020). Crucially, generative IE represents a special case of LLM inference: it relies on mapping long, heterogeneous inputs to very short, structurally rigid, and parseable outputs (Zhang et al., 2025). This structural disparity exposes a critical lack of energy-aware benchmarks tailored to the operational realities of constrained generation.

Equating smaller model architectures with proportional energy savings is an oversimplification, as generative IE efficiency is a systems-level property. Since structured extraction tasks (NER, RE, EE) demand strict schema adherence (Politov et al., 2025), assessing their actual energy overhead requires explicitly integrating output formats, constrained decoding mechanisms, and serving-layer configurations into the evaluation framework. Enforcing these constraints imposes a "computational tax" that alters energy consumption independently of raw token volume. Consequently, standard token-level energy metrics, which treat all generated text equally, are fundamentally insufficient for capturing the true GPU inference cost of structured extraction.

To map these unique inefficiencies, we establish the first dedicated benchmark for sustainable IE. Specifically, we make three core contributions. First, we introduce a comprehensive energy-aware benchmark for NER, RE, and EE spanning multiple model scales, output formats, decoding regimes, serving configurations, and GPU generations. Second, we define a standardised methodology using task-aligned energy metrics, such as the F1-Energy Ratio (F_1/kJ) and Energy per True Positive (J/TP), thereby demonstrating why conventional token-level metrics fail for structured extraction. Finally, we provide an empirical deployment analysis that translates these interactions into actionable guidelines (See Table 1). Specifically, our evaluation investigates whether model size reliably predicts energy efficiency; whether supervised encoders remain preferable to zero-shot decoders; how output formatting and constrained decoding affect the energy-accuracy trade-off; and the extent to which serving-layer parameters alter the efficiency frontier.

* Corresponding author.

[1] https://github.com/Energy-aware-IE/Energy-Benchmarking-of-IE

2 Related Work

Encoder only models Historically, IE tasks relied on fully supervised, encoder-only architectures, with models like mBERT (Pires et al., 2019) and XLM-R (Conneau et al., 2020) establishing strong baselines. Complex structured prediction tasks required specialised architectures, such as DREEAM for document-level RE (Ma et al., 2023) and argument-linking networks for EE (Ebner et al., 2020). In low-resource scenarios, translation-plus-projection pipelines (García-Ferrero et al., 2022; Poncelas et al., 2023) achieve predictive accuracy but introduce immense computational overhead during the translation step (Politov et al., 2025).

LLMs for IE tasks Generative LLMs have shifted the IE paradigm toward zero- and few-shot prompting (Zhang et al., 2025), bypassing translation pipelines with pre-trained multilingual representations (Mo et al., 2025). However, enforcing strict schema adherence (e.g., JSON, DST) (Bourhis et al., 2020) (Shen et al., 2025) necessitates constrained decoding mechanisms (Ruan et al., 2025). Libraries like Outlines (Willard and Louf, 2023) and XGrammar (Dong et al., 2024) guarantee valid outputs but impose a hidden autoregressive computational overhead, making their deployment significantly more resource-intensive than that of single-pass encoders.

Benchmarks The NLP community lacks a unified benchmark targeting the intersection of IE and energy efficiency. Standard IE benchmarks prioritise predictive accuracy (F1) (Yao et al., 2019), (Hu et al., 2020), (Yu et al., 2026), while efficiency benchmarks evaluate raw throughput or memory, ignoring IE's structural complexities (Chitty-Venkata et al., 2025), (Ni et al., 2025). Consequently, no standardised approach simultaneously evaluates both classical encoders and zero-shot decoders to map the Pareto frontier between accuracy and energy.

Measurement of the energy consumption Empirical hardware profiling has largely replaced theoretical FLOP counting for measuring deep learning energy consumption (Zhou et al., 2024), (Niu et al., 2025). Existing energy studies often ignore the complexities of downstream tasks (Jurkschat et al., 2025). Accurate profiling requires robust telemetry; tools like *nvidia-smi* (Yang et al., 2024) and NVML (Huang et al., 2025) have limitations in polling overhead or scalability (Ansari and Ansari, 2025). Data Centre GPU Manager (DCGM) (Wassermann et al., 2026) offers high-fidelity, low-overhead measurements compared to other solutions. Alongside hardware, we standardise the software serving layer on vLLM, utilising its PagedAttention mechanism for high-concurrency throughput, which provides efficiency benefits compared to other popular inference engines, e.g., HuggingFace Text Generation Inference (TGI) (Kolluru, 2025), being the de facto standard. This controls the computational variance introduced by backend optimisations (Pape et al., 2026). Finally, prior LLM energy evaluations rely on generalised Energy-per-token metrics (Wu et al., 2025), (Solovyeva and Castor, 2026), which are insufficient for structured IE tasks and thus necessitate our task-aligned efficiency metrics.

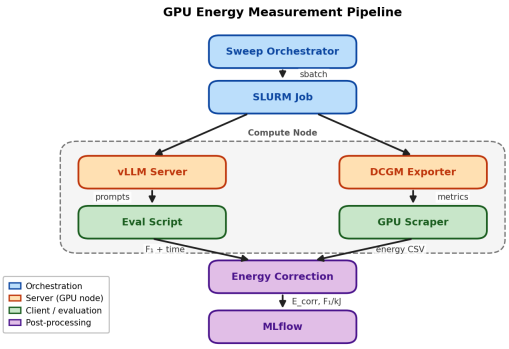


Figure 1: Scheme of the measurement

Measurement of CO2 and limitations of this approach Reporting Carbon Dioxide equivalent emissions is problematic for architectural benchmarking (Luccioni et al., 2023). Carbon footprints are tied to the local grid's carbon intensity at the time of computation (Li et al., 2024), causing identical models to yield vastly different footprints based on geographic location (e.g., Germany vs. Norway) (Wu et al., 2022). Because these variables obscure intrinsic algorithmic efficiency, reporting direct energy consumption provides a far more stable, reproducible metric for architectural comparisons.

3 Benchmark Design

3.1 System Overview

Because the energy efficiency of LLM-based IE is multifaceted, our modular benchmark provides a unified framework for navigating trade-offs between predictive quality (F1) and inference energy across core IE tasks (NER, EE, RE). We evaluate both zero-shot decoders and traditional fine-tuned encoders using shared, task-aligned ef-

ficiency metrics. By systematically isolating variables across tasks, decoding formats, and serving configurations, we map the energy-accuracy frontier. This equips the research community with actionable guidance to optimally configure IE pipelines, whether targeting peak accuracy, maximum efficiency, or a sustainable balance.

3.2 Energy Telemetry and Measurement Protocol

Accurate energy profiling requires isolating the *active inference window* from general server overhead. We omit full-system power to focus exclusively on GPU power draw, utilising an NVIDIA DCGM telemetry pipeline that polls the hardware energy counter (`DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION`) at 100 ms intervals. To eliminate initialisation variance, the active window is strictly defined as the continuous period where GPU utilisation exceeds zero. Crucially, we isolate pure computational cost by measuring only direct GPU power draw, explicitly excluding the energy consumption of the host CPU, system RAM, and VRAM. This calibrated estimator successfully excludes model loading and initial cache allocation, but it inherently captures operational realities such as prompt parsing, constrained decoding overhead, and baseline idle power.

Variability and significance We ran all configurations three times ($N = 3$). With three runs per configuration, a standard confidence interval would have to assume normally distributed errors, which we cannot verify at that sample size. We therefore report Chebyshev intervals, which make no distributional assumption but require an upper bound on the run-to-run variation, obtained from a separate calibration study of 300 repeated runs: GPU energy varies by 1.2% on average (1.6% at worst) between identical repetitions, and F1 far less. A mean from three runs is thus known to within $\pm 5\%$, and two configurations are separable only if they differ by more than 7% or about 2% for quantities averaged over ten XTREAM NER languages. The main effects we report exceed these thresholds several times over, whereas differences of a few percent should be read as directional rather than established. Full derivations and per-configuration intervals are given in Appendix A.4.

3.3 Task-Aware Energy Metrics

The conventional efficiency baseline, Energy per Generated Token ($E_{\text{Gen}} = \Delta E / N_{\text{gen}}$, for N_{gen} output tokens), is fundamentally flawed for Information Extraction (IE). It conflates formatting verbosity with utility, rewarding fast hallucinations while penalising structured prompts. Efficiency must instead reflect task-level utility.

We isolate the active inference energy (ΔE, in Joules) by subtracting the cumulative GPU energy recorded at the onset $E(t_{\text{start}}$ of active generation from the total energy recorded at its completion $E(t_{\text{end}}$:

$$\Delta E = E(t_{\text{end}}) - E(t_{\text{start}}) \tag{1}$$

We introduce the **F1-Energy Ratio** (F_1/kJ) ($\hat{F}_1/(\Delta E/1000)$) and its inverse, the Energy-to-F1 Cost ($E_{F1} = \Delta E / \hat{F}_1$), to quantify the energy yield of accuracy.

Concurrently, **Energy per Extraction** ($E_{\text{extraction}} = \Delta E / \hat{N}_{\text{pred}}$, for $\hat{N}_{\text{pred}}$ predicted spans) measures the raw cost per predicted span. Analysing F1-based efficiency alongside extraction cost reveals whether energy is wasted on over-generation or misclassification.

Finally, to more rigorously approximate the cost of successful extractions, we introduce **Energy per True Positive** (J/TP) (E_{TP}). This metric estimates the average energy expended to yield a single correct extraction:

$$E_{\text{TP}} = \frac{\Delta E}{\hat{P} \cdot \hat{N}_{\text{pred}}} \tag{2}$$

where $\hat{P}$ is extraction precision. By severely penalising high-volume, low-precision hedging, E_{TP} defines the true operational cost of useful outputs.

3.4 Taxonomy of Experimental Parameters

To systematically isolate the drivers of inference efficiency and computational overhead, we partition the experimental space into four operational axes. **Structural Decoding Regimes** encompass constrained decoding backends (`xgrammar`, `Outlines`, EBNF CFGs) alongside various output formats (JSON, YAML, XML, DST). Furthermore, **Serving-Layer Constraints** icluding memory fragmentation (`block_size`), decode budgeting (`max_new_tokens`), and prefill limits (`max_num_batched_tokens`, `max_model_len`). Finally, **Throughput Dynamics (Concurrency)**

Batch size (`batch_size`), vLLM scheduler capacity (`max_num_seqs`). It is important to note that batch size is a parameter of the evaluation client rather than of vLLM, which performs continuous batching and exposes no request-batch setting; we report it for reproducibility, but it is a property of how requests are issued, not a serving-layer tuning lever (Appendix A.9 characterises the mechanism and quantifies its reach). Moreover, **Task Specification** covers prompt taxonomy and few-shot scaling; as it is secondary to our systems-level focus, more information can be found in A.5. For the full list of the evaluated parameters, see Table 3.

4 Methodology

The detailed benchmark overview table is presented in the Appendix A.2

4.1 Experimental Setup

We conduct our evaluations across NVIDIA A100 (40GB) and H100 (96GB) GPUs to observe energy scaling across compute generations. For generative models, we propose standardising IE energy benchmarking on the vLLM engine (v0.8.4), thereby enabling broader compatibility across GPU generations (Kwon et al., 2023), while vLLM's PagedAttention mechanism delivers state-of-the-art memory efficiency and high throughput. Crucially, vLLM demonstrates exceptional energy efficiency during the decoding phase (Niu et al., 2025), which is crucial for IE tasks where the generation stage is dominant. The full description of the experimental setup is presented in A.8.

4.2 Downstream Tasks and Datasets

To map the energy-accuracy frontier across varying structural demands, our benchmark evaluates three Information Extraction tasks of escalating complexity: NER, EE, RE.

NER serves as a foundational token-level baseline, isolating the fundamental energy costs of structural formatting with minimal reasoning overhead. This evaluation primarily centres on the XTREME PAN-X dataset for broad multilingual analysis, with secondary experiments on MasakhaNER to assess low-resource generalisation. Increasing in complexity, **EE** (RAMS 1.0c) shifts the workload to multi-sentence semantic reasoning, requiring models to identify event triggers and map entity spans to 139 distinct roles. Finally, **RE** (DocRED) exhibits the highest structural complexity, requiring document-level, multi-hop reasoning to track entities and classify relationships into 96 Wikidata types across long contexts. By tracking this progression from simple tagging to complex document reasoning, the benchmark systematically isolates how task difficulty dynamically alters the energy costs of both classical encoders and generative decoders.

4.3 Output formats and decoding strategies.

We evaluate three serialisation formats and four decoding strategies:

Serialisation formats: We consider four formats: **JSON** (key–value pairs with strict syntax, serving as the de facto standard for structured outputs); **YAML** (an indentation-based, human-readable format relying on whitespace rather than brackets); **XML** (a tag-based markup format providing unambiguous nesting at the cost of higher verbosity); and **DST** (a compact, line-oriented domain-specific template designed for information extraction with minimal punctuation overhead).

Decoding strategies: We compare four generation approaches: **Unconstrained** (baseline, free generation without structural guarantees); **xgrammar** (compiles a JSON Schema into a dynamic token mask for single-pass structural validity; **EBNF** (uses hand-written Extended Backus - Naur Form grammars for fine-grained control over allowed sequences); and **Outlines** (enforces regex or grammar using a token-level finite-state machine, offering high generality at the cost of FSM construction overhead).

4.4 Choice of Models

We evaluate several general-purpose instruction-tuned LLMs spanning two families and a different range of parameters: the Gemma 3 family (4B, 12B, and 27B) (Team et al., 2025) to isolate intra-family scaling effects on performance and energy, and Llama 3.3 70B (Grattafiori et al., 2024) as a more advanced, well-known reference point in the community. Moreover, to test our scalability hypothesis with a high-capacity model, we used the Mistral Large (123B) model (Mistral AI, 2024). For more details, see Appendix A.3. For encoder approaches across the three tasks, we compare against fully supervised architectures: DREEAM (Ma et al., 2023) for document-level RE (guided self-attention), the Multi-Sentence Argument Linking framework (Ebner et al., 2020) for EE (cross-sentence dependencies), and a projection-based

pipeline (Politov et al., 2025) for NER.

4.5 Baseline

To establish a realistic point of comparison, we define a naïve production baseline representing a standard vLLM deployment for high-throughput IE. Because input and output lengths differ sharply across tasks, the baseline is specified per task (Table 7): context and decode limits are set only so that they accommodate the respective dataset, and concurrency is scaled to fit the resulting sequences in memory. All other settings are held at general engine defaults and are identical across tasks. These values are chosen for feasibility rather than efficiency, providing a well-defined, unoptimised anchor against which all parameter ablations are measured. Furthermore, we enable standard throughput optimisations (prefix caching and chunked prefill) while leaving the Key-Value cache unquantised (`auto`) on `bfloat16` weights to ensure a clean energy reference. This setup provides a well-defined, unoptimised anchor against which all parameter ablations are measured. We highlight a few primary serving constraints throughout our analysis; the exhaustive baseline configuration is detailed in Appendix A.7.

5 Empirical Evaluation

5.1 Deployment Regimes: Task-Specific Supervision and Zero-Shot Flexibility

Direct comparisons between fully supervised encoder systems and zero-shot LLMs are inherently asymmetric. The evaluation situates encoder models and LLMs within different deployment regimes rather than architectures. This offers a practical perspective on when practitioners should still prefer supervised encoders over zero-shot generative decoders. Supervised encoders represent the efficiency upper bound when task-specific supervision and dedicated pipeline engineering are available. In contrast, generative LLMs become operationally attractive when supervision, language coverage, or engineering resources are limited.

To map these regimes, we establish supervised baselines across all three tasks. For RE, we evaluate DREEAM (Ma et al., 2023); for EE, Multi-Sentence Argument Linking (Ebner et al., 2020). Although Arglinking on gold spans is not directly comparable to the end-to-end span detection and role assignment required of LLMs, we include it to define the architectural upper bound for efficiency (F_1/kJ). All other metrics use identical evaluation protocols. For NER, we evaluate a standard translation-plus-projection pipeline (Politov et al., 2025) utilising NLLB-200 and XLM-R-Large. This allows us to contrast zero-shot flexibility against a classical multi-stage pipeline. We test this on German (XTREME), a high-resource control where supervised encoders establish the upper bound, and on Luganda (MasakhaNER), a low-resource African language. Luganda explicitly exposes the computational bottleneck in classical pipelines that include a translation step.

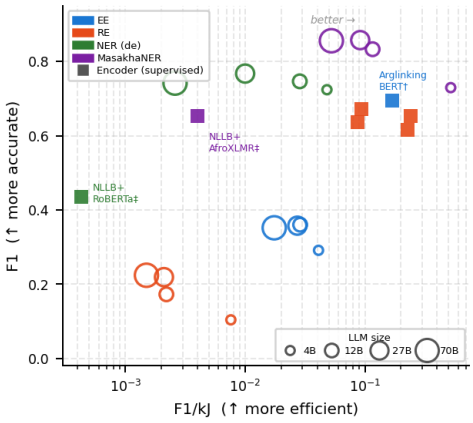


Figure 2: Performance of Encoder vs. Decoder Architectures Across Diverse Tasks. Open circles denote decoder LLMs at their best configuration per task and model size; filled squares denote supervised encoder baselines. (H100 config) Top-right is optimal.

As visualised in Figure 2, classical encoders establish the efficiency upper bound for monolingual inference efficiency when task-specific supervision is available. For document-level RE, DREEAM-BERT achieves 0.242 F1/kJ, substantially outperforming the most accurate tuned LLM (LLaMA-3.3-70B at 0.003 F1/kJ). Although the DREEAM-RoBERTa backbone attains a marginally higher F1 score (0.671 vs 0.653), BERT consumes less than half the energy (2.69 kJ vs 7.20 kJ), making it the Pareto-optimal choice. Similarly, for Event Extraction (EE), Arglinking-BERT reaches 0.170 F1/kJ, whereas the most efficient LLM configuration (Gemma-3-12B) achieves 0.091 F1/kJ at substantially lower accuracy. These results indicate that, for conventional structured prediction tasks, the flexibility of pipeline-free LLM inference incurs a significant energy cost.

In cross-lingual and low-resource settings, the efficiency trade-off shifts as classical approaches rely on energy-intensive translation pipelines. Gen-

erative decoders instead operate without explicit intermediate translation stages. On MasakhaNER (Luganda), Gemma-3-4B bypasses the translation bottleneck, achieving 0.704 F1 at only 1.68 kJ, corresponding to 0.418 F1/kJ, a very high efficiency observed across all tasks in this study.

This dynamic holds for higher-resource Named Entity Recognition (NER) as well. On the German XTREME benchmark, Gemma-3-27B achieves peak accuracy ($F_1 = 0.767$), but the smaller Gemma-3-4B provides a 14× energy reduction (5.15 kJ, 0.127 F1/kJ) while retaining an F1 of 0.655. Ultimately, while supervised encoders are the optimal deployment choice for high-resource, monolithic extraction, compact instruction-tuned decoders represent the most sustainable strategy for flexible, cross-lingual environments.

Motivations for Zero-Shot Generative Extraction When is LLM deployment justified? As Figure 2 illustrates, single-pass encoders define the efficiency upper bound when task-specific supervision is abundant. However, encoders lack zero-shot flexibility. In cross-lingual or low-resource settings, generative LLMs bypass energy-intensive translation pipelines, executing extraction in a single pass. Consequently, these architectures occupy distinct deployment regimes: LLMs provide a necessary, pipeline-free alternative when task-specific supervision or engineering resources are unavailable.

5.2 Decoding Regimes and Formats

To address how output format and constrained decoding affect the energy-accuracy trade-off, our evaluations show that inference costs and predictive quality are dictated by the complex interaction among task rigidity, model scale, and decoding format; see Figure 3.

Constrained JSON (*xgrammar*) incurs negligible overhead on NER. Across ten languages and nine prompts, median run energy remains stable at every scale (e.g., unconstrained vs. *xgrammar*: 23.8 vs. 25.5,kJ at 4B, and 228.6 vs. 230.0,kJ at 70B). Energy per correct entity actually decreases for models 12B–70B, and accuracy is fully preserved aside from a marginal 0.011 F_1 drop at 4B. This efficiency stems from computing the constraint mask in-kernel (GPU) rather than on the host (CPU). Different formatting influences energy consumption differently: JSON+`xgrammar` incurs minimal overhead (≈7% for median-of-medians), whereas FSM-heavy methods like DST+`Outlines`

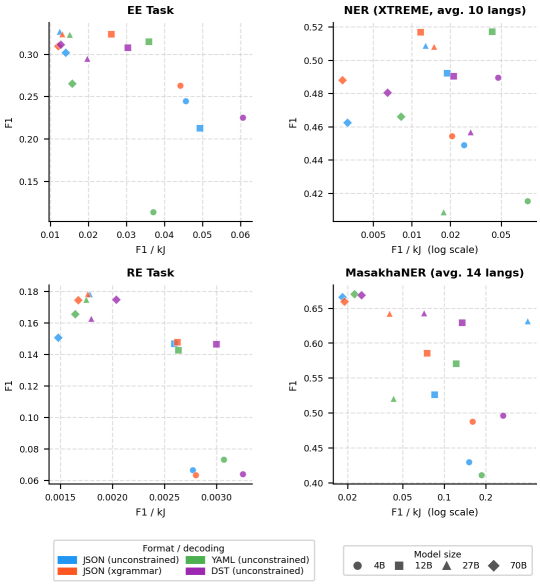


Figure 3: F1 vs. F1/kJ across four IE tasks, model scales, and output formats. NER: XTREME averages 10 languages; MasakhaNER averages 14 languages. Each point is one model-format configuration averaged across prompts (H100 config).

or XML+`xgrammar` are energy-prohibitive, consuming up to 7, 6× the same model's unconstrained JSON cost. Furthermore, accuracy is highly format-sensitive.

Decomposing the constrained-decoding tax. Separating GPU compute from idle time reveals the true source of this energy overhead. DST+EBNF uses 6.0× the energy of unconstrained DST (up to 10.4× at 70B), but only 1.7× once idle draw is excluded. XML+*xgrammar* shows similar spikes (up to 20.9×). Consequently, 62-71% of the added energy cost comes from the GPU idling while token masks are computed on the host CPU instead of in parallel. While a baseline compute penalty is inherent to any grammar enforcement, the massive idle penalty (roughly a factor of three) is simply an implementation bottleneck. JSON+*xgrammar* avoids this penalty entirely by computing its mask in-kernel(GPU side). Appendix A.10 reports the full decomposition.

Format selection for RE: All formats yield similar accuracy (within 0.03 F_1), but energy diverges sharply. DST uses 10-17% less energy per run than JSON and vastly less energy per correct triple (24.3 vs. 33.4 J/TP at 4B; 35.1 vs. 76.8 at 70B). Thus, we recommend DST for RE.

To balance accuracy, energy, and deterministic parsing, Gemma-3-12B with JSON+`xgrammar` is the ideal baseline, guaranteeing schema adherence

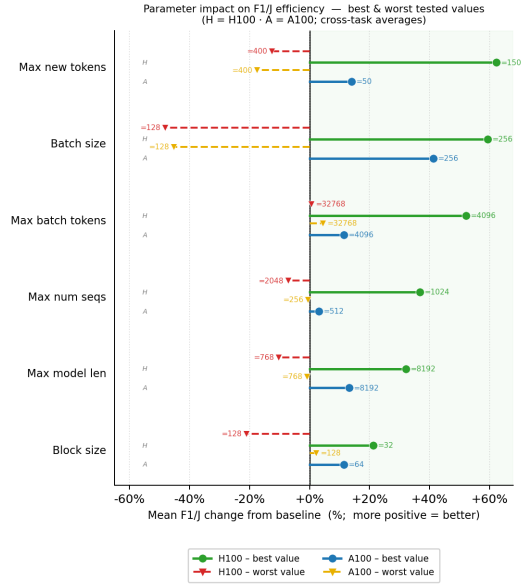


Figure 4: Cross-architectural analysis of server-side parameter variations

across NER, EE, and RE (with energy impacts ranging from -8% to $+115\%$). For cost-agnostic peak accuracy, use 12B JSON+`xgrammar` for NER, 27B unconstrained JSON for EE, and for RE, unconstrained DST (70B) or JSON (12B/27B). Strictly energy-constrained setups should default to 4B unconstrained DST.

For multilingual deployments below 27B, unconstrained DST is the robust default: it achieves the best mean rank and uses 2-5$\times$ less energy than DST+EBNF. Grammar enforcement yields no accuracy gains at these scales. Notably, on MasakhaNER, unconstrained DST improves F_1 across all scales while reducing energy.

In summary, output formats and constrained decoding are primary drivers of the energy-accuracy trade-off, not mere implementation details. Sustainable generative IE requires deliberately aligning these mechanisms with the target task's structural rigidity.

5.3 Influence of Serving Parameters on Energy Efficiency

To determine how much serving-layer parameters affect the efficiency frontier, we conducted a comprehensive ablation study across NVIDIA A100 and H100 GPUs (details in A.2). Our findings in Figure 4 confirm that default inference engine settings are rarely calibrated for structured IE. Ultimately, achieving optimal efficiency requires precisely aligning a strict hierarchy of serving constraints with the target task's specific complexity.

We identified a three-way structure of parameters: critical (`max_new_tokens`, `batch_size`), task-specific (`max_num_batched_tokens`, `block_size`), and low-impact (`max_num_seqs`, `max_model_len`).

Critical Parameters: Decode Limits and Batch Sizing Driving efficiency swings of $\pm 50\%$ F_1/kJ, these parameters demand aggressive tuning. For NER, unconstrained generation wastes energy: verbose chain-of-thought prompts inflate costs by 50% (9.6 vs 6.4 J/TP), making tight decode caps (50–100 `max_new_tokens`) critical to prevent costly hallucinations. Conversely, Relation Extraction (RE) requires a large budget (150-200) tokens. For the client batch size (`batch_size`), moderate HTTP batches (256) reduce memory-bandwidth contention for EE and RE (yielding up to $+141.6\%$ F_1/kJ). Crucially, dropping the batch size too far (64-128) breaks the chunked-prefill scheduler, destroying energy efficiency across all tasks by up to 61.1%.

Task-Specific Parameters: Memory and Prefill Allocation Neither limit admits a single setting that is optimal across tasks. The prefill window (`max_num_batched_tokens`) is consequential only for prompt-heavy workloads: increasing it to 16,384 changes F_1/kJ by between -25% and $+78\%$ on EE and by -5% to $+22\%$ on RE, but by less than $\pm 6\%$ on NER. The direction of this effect varies across model scales, so the parameter requires per-model tuning rather than a global default. Enlarging `block_size` beyond the default of 16 is seldom beneficial: NER efficiency declines monotonically (-4.5%, -9.5% and -18.7% at 32, 64 and 128 respectively), EE improves only at 32 ($+18.0\%$), and a block size of 128 reduces efficiency in 14 of the 16 task–model combinations examined.

Low-Impact Parameters: Scheduler Capacity and Context Truncation Though these parameters have the smallest median effect, they are not uniformly safe. Reducing scheduler capacity (`max_num_seqs`) from 1024 to 512 improves F_1/kJ in 10 of 16 configurations (median $+0.8\%$), but increases energy cost in the rest—up to $+147\%$ for EE at 12B. Truncating context windows (`max_model_len`) behaves similarly: despite a median effect under 1%, outcomes range from -58% to $+213\%$. Neither should be adopted as a

Core Finding	Empirical Evidence	Practical Implication
Encoders set efficiency bounds.	DREEAM-BERT ($0.242F_1$/kJ) on RE outperforms tuned LLMs ($0.003F_1$/kJ).	Prefer classical encoders for supervised tasks; use LLMs for zero-shot or cross-lingual needs.
Output format determines accuracy.	DST is the efficient default for RE; simple NER optimises best under DST.	Larger models scale best with YAML or unconstrained JSON.
Constrained decoding imposes a tax.	FSMs (`Outlines`) inflate energy up to 7,6×. `xgrammar` boosts rigid tasks but minimally hurts small NER models.	Benchmark schema enforcement. Default to JSON+`xgrammar`, avoiding heavy FSMs if energy-bound.
Serving limits shift the frontier.	(`block_size`,`batch_size` `max_new_tokens`, `max_num_batched`) alters efficiency by $\pm 50\%$.	Aggressively tune decode budgets to prevent hallucinations; mandate reporting of serving parameters.
Scale does not guarantee efficiency.	On German NER, Gemma-3-4B offers a $5.1\times$ energy reduction over 27B with competitive utility.	Exhaust pipeline tuning (formats, limits) on compact 4B–12B models before scaling capacity.

Table 1: Main findings and practical implications for deploying energy-efficient generative IE pipelines.

global default without per-deployment validation.

Influence of parameters: Our ablation study identifies `max_new_tokens`, `batch_size`, and `block_size` as the primary drivers of LLM energy efficiency in IE. This triad triggers the largest performance swings, altering efficiency by $\pm 50\%$ (median) and up to $+159\%$. `max_new_tokens` bounds decoding budgets to prevent costly hallucinations in short-span tasks like NER. Tuning `batch_size` prevents memory-bandwidth contention and maintains scheduler stability, while `block_size` manages KV-cache paging, where exceeding the default of 16 is seldom beneficial. Therefore, aligning these three parameters with a task's structural demands is a foundational step for optimising generative IE pipelines.

5.4 Scaling Beyond the 70B Reference Point

Gemma-3-27B matches or exceeds our 70B reference point on several tasks; thus, we test whether this reflects the model or the scale by adding a more modern Mistral-Large-Instruct-2411 (123B dense) under the same serving configuration (Table 7). We compare both models at TP = 4 and report Llama-3.3-70B additionally at TP = 2, the allocation used throughout this paper. In F_1/kJ the ordering is the inverse of model size: Gemma-3-27B is the most efficient of the three models and Mistral-Large the least on all four tasks, using 20 - 84% more energy per unit of F_1 than Llama-3.3-70B on matched hardware, while Llama-3.3-70B is itself 36% more efficient at TP = 2 than at TP = 4 on German NER. Accuracy does not follow this ordering: Mistral-Large improves F_1 only on RE (0.208 vs. 0.146), matches Llama-3.3-70B on German NER, and less efficient on EE (0.287 vs. 0.306) and MasakhaNER (0.618 vs. 0.663). Extended results are in Table 10. We conclude that the largest model is thus the least efficient on every task and more accurate on only one.

5.5 Main Findings

Our comprehensive evaluation demonstrates that the energy efficiency of LLM-based IE cannot be predicted by model scale alone. Instead, achieving optimal F1 and energy trade-offs requires precise alignment of prompt design, decoding structures, and serving parameters. We synthesise our core findings in Table 1.

5.6 Scenario-Driven Configuration Guidelines

To operationalise these trade-offs and provide actionable setup guidance, we categorise the optimal parameter configurations and their empirical gains relative to the unoptimised baseline across four distinct deployment scenarios:

Scenario A: Peak Accuracy. *Settings:* The most accurate output format per task. *Outcome vs. Baseline:* Peak configurations consistently outperform the naïve 4B unconstrained-JSON baseline in average F_1: NER 0.518 vs. 0.449 (12B, YAML); MasakhaNER 0.693 vs. 0.430 (70B, unconstrained XML); EE 0.327 vs. 0.245 (27B, unconstrained JSON); and RE 0.178 vs. 0.067 (27B, unconstrained JSON). Only NER peaks under a constrained format. Strict FSM enforcers, tested solely on the NER tasks, fall short of these peaks (0.491 NER, 0.664 MasakhaNER) while inflating energy up to $38\times$, limiting their viability to offline, high-stakes extraction.

Scenario B: Maximum Tuning Gain. *Settings:* Reducing HTTP `batch_size` to 256 and capping decode budgets at `max_new_tokens=200`.

Outcome vs. Baseline: These are our most reliable efficiency levers. Reducing `batch_size` eases memory-bandwidth contention, improving all EE and RE tested runs. It boosts F_1/J by up to $+141.6\%$ and cuts energy per correct extraction by a median 27%, for a minor median cost of 3% F_1. Capping `max_new_tokens` at 200 applies broadly (NER, MasakhaNER, and RE), cutting energy (median -14%, best -35%) while maintaining accuracy (median $+0.5\%$ F_1).

Scenario C: Baseline Efficiency. *Settings:* Default vLLM deployment (`batch_size`=1024, `max_num_seqs`=1024) paired with compact models (4B) and unconstrained JSON. *Outcome vs. Baseline:* Ideal for strict energy limits with zero engineering overhead. Thus 4B model uses 2.7-4.2$\times$ less energy compared to 27B and 7.5–10$\times$ compared to 70B.Unconstrained generation reduces output validity in smaller models, causing a modest accuracy drop in short-span extraction (87% of peak F_1 on NER) but a severe degradation in document-level relation extraction (36% on RE).

Scenario D: Pareto-Optimal Default. *Settings:* Mid-scale models (Gemma-3-12B) paired with lightweight JSON+*xgrammar* constraints and task-specific decode budgets (`max_new_tokens`=100 for NER and MasakhaNER, 150 for EE and RE). *Outcome vs. Baseline:* This robust setup prevents hallucinations and preserves accuracy. It recovers at least 80% of the peak F_1 (100% on NER, 97% on EE, 85% on MasakhaNER and 81% on RE) and delivers transformative efficiency gains, including a $+52.2\%$ F_1 surge in EE and avoidance of Scenario A energy spikes. Crucially, decode budgets must match the task: the NER budget cuts energy per correct extraction by 45%, and 5% on MasakhaNER, while RE needs 150 tokens to avoid truncating triples

5.7 Discussion

Our findings indicate that energy-aware evaluation for generative Information Extraction must move beyond reporting model size, GPU type, or raw Energy-per-Token statistics. Because structured outputs and constrained decoding fundamentally alter inference behaviour, energy efficiency in generative IE is a systems-level property rather than a direct consequence of parameter count. An energy benchmark of this kind measures a pipeline, not a model in isolation, meaning the serving stack contributes to every reported metric. We distinguish between generalizable IE properties and system-specific artefacts. Task-aligned metrics, encoder-decoder separation, format–accuracy interactions, and scale-efficiency decoupling should transfer across deployments. Conversely, metrics involving block size, chunked prefill, and KV-cache precision reflect vLLM on NVIDIA hardware rather than the IE task itself. Though stack-dependent, these hardware insights remain valuable for optimising real-world deployments. (Properties arising from the measurement apparatus bounding uncaptured energy are detailed in Appendix A.9). As a result, two deployments of the same model can exhibit substantially different energy profiles depending on these configurations. For reproducible and meaningful benchmarking, we recommend reporting the structural properties of the extraction pipeline alongside predictive metrics, including the output format, constrained decoding strategy, parser enforcement mechanism, decode budget, and serving configuration. These operational details dictate inference behaviour and energy consumption; without them, cross-model comparisons cannot be reliably interpreted.

6 Conclusion

As Information Extraction (IE) rapidly transitions toward generative architectures, understanding the true operational costs of inference is critical. In this work, we introduced the first dedicated benchmark for energy-efficient IE, moving beyond standard token-throughput metrics to use task-aligned energy indicators (such as J/TP and F_1/kJ).

Crucially, our evaluations demonstrate that the energy efficiency of LLM-based IE cannot be predicted by model size or any single parameter alone. Instead, efficiency is largely determined by the complex interplay among architectural paradigms, structural decoding formats, and serving-layer dynamics. By systematically mapping how these parameters affect the energy-accuracy trade-off, this benchmark provides actionable, scenario-driven deployment guidelines. We explicitly outline which configuration settings to use and when, enabling practitioners to set up their pipelines to achieve peak predictive accuracy, maximise energy efficiency, or establish a targeted Pareto-optimal balance. Ultimately, this work equips the NLP community with the empirical foundation necessary to deploy highly capable, yet energy-sustainable, Information Extraction systems.

Ethics Statement

The rapid adoption of Large Language Models (LLMs) for Information Extraction (IE) necessitates balancing extraction quality with environmental cost. Because generative IE imposes substantial energy overheads, we align with the Green AI principle (Schwartz et al., 2020) that efficiency must be a first-class evaluation criterion. However, minimising energy is not an absolute rule: in high-stakes domains or low-resource cross-lingual settings, the energy cost of larger models may be ethically justified to prevent harmful errors or expand language access. We therefore advocate *proportional deployment*, where practitioners select the lowest-energy configuration that meets their reliability threshold. Task-aware metrics like F1-Energy Ratio (F_1/kJ) make these trade-offs explicit, guiding optimisations that can drastically reduce total energy use (Fernandez et al., 2025).

Crucially, optimising individual extractions carries the ethical risk of the rebound effect: making IE cheaper may encourage processing at a massive scale, paradoxically increasing total computational demand. Furthermore, runtime GPU energy underrepresents the true ecological footprint, omitting life-cycle equipment emissions (Luccioni et al., 2023), water consumption, and infrastructure expansion (Carmeno et al., 2026). Consequently, energy-efficient benchmarks must not only identify optimal configurations but also encourage restrained deployment, ensuring efficiency gains are used to reduce total environmental impact rather than merely subsidising further automation.

Limitations

While our benchmark captures critical efficiency dynamics, its scope is limited to specific extraction tasks (NER, RE, EE) and models (Gemma-3, Llama-3.3, used per their licenses), meaning these trends may not generalise to alternative architectures or retrieval-augmented pipelines. Furthermore, our evaluation framework introduces systemic boundaries that influence the reported metrics, which we detail further in Appendix A.9. To ensure reproducible comparisons, our measurements strictly isolate the energy used for active GPU inference. By excluding model initialisation, system-level overhead (e.g., host CPU, network interface, cooling), and regional grid variables, we deliberately avoid making broader carbon-footprint claims. However, this boundary impacts certain configurations unevenly: because FSM-based constrained decoding forces the GPU to idle while waiting for CPU-bound parsing, its measured energy overhead is highly sensitive to host system performance rather than being intrinsic to the extraction task itself.

Additionally, our batching and scheduling findings are inherently tied to the vLLM stack and NVIDIA hardware. To avoid confounding variables, we standardised on this single inference backend; consequently, while our broad task-level findings generalise, specific system behaviours characterise this software stack and may not transfer to other frameworks. Notably, the H100 served as our primary evaluation platform, as establishing a direct architectural comparison with the A100 falls outside the core objectives of this study. We also note that our Hyperparameter Optimisation (HPO) trial was exploratory rather than exhaustive. Finally, the measurement client itself can distort results: high request concurrency can trigger transport-layer failures that imitate model errors, and telemetry misalignment can corrupt energy readings. We therefore filter out invalid measurement windows, report request success rates alongside accuracy, and emphasise that robust benchmarking must carefully distinguish infrastructure bottlenecks from actual model failures.

Acknowledgments

The authors acknowledge the financial support by the Federal Ministry of Research, Technology and Space of Germany and by Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus in the programme Center of Excellence for AI-research „Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig“, project identification number: ScaDS.AI.

The authors gratefully acknowledge the computing time made available to them on the high-performance computer at the NHR Center of TU Dresden. This center is jointly supported by the Federal Ministry of Research, Technology and Space of Germany and the state governments participating in the NHR.

Andrei Politov extends his deepest gratitude to his late mother for her unwavering belief in him, and to his wife, whose invaluable support and inspiration were instrumental throughout the development of this research.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Mufakir Qamar Ansari and Mudabir Qamar Ansari. 2025. Racing to idle: Energy efficiency of matrix multiplication on heterogeneous cpu and gpu architectures. *Preprint*, arXiv:2507.20063.

Pierre Bourhis, Juan L. Reutter, and Domagoj Vrgoč. 2020. Json: Data model and query languages. *Information Systems*, 89:101478.

Noemi Luna Carmeno, Tiago Domingos, and Daniel W. O'Neill. 2026. The impacts of artificial intelligence on environmental sustainability and human well-being. *Preprint*, arXiv:2602.24091.

Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. 2025. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators. In *Proceedings of the SC '24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '24, page 1362–1379. IEEE Press.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2024. Xgrammar: Flexible and efficient structured generation engine for large language models. *ArXiv*, abs/2411.15100.

Seth Ebner, Patrick Xia, Ryan Culkin, Kyle Rawlins, and Benjamin Van Durme. 2020. Multi-sentence argument linking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8057–8077, Online. Association for Computational Linguistics.

Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. 2025. Energy considerations of large language model inference and efficiency optimizations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32556–32569, Vienna, Austria. Association for Computational Linguistics.

Iker García-Ferrero, Rodrigo Agerri, and German Rigau. 2022. Model and data transfer for cross-lingual sequence labelling in zero-resource settings. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6403–6416, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. Xtreme: a massively multilingual multi-task benchmark for evaluating cross-lingual generalization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Hongzhen Huang, Kunming Zhang, Hanlong Liao, Kui Wu, and Guoming Tang. 2025. Aimeter: Measuring, analyzing, and visualizing energy and carbon footprint of ai workloads. *Preprint*, arXiv:2506.20535.

Lena Jurkschat, Preetam Gattogi, Sahar Vahdati, and Jens Lehmann. 2025. Bali - a benchmark for accelerated language model inference. *IEEE Access*, 13:98976–98989.

Saicharan Kolluru. 2025. Comparative analysis of large language model inference serving systems: A performance study of vllm and huggingface tgi. *Preprint*, arXiv:2511.17593.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024. Sprout: Green generative AI with carbon-efficient LLM inference. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21799–21813, Miami, Florida, USA. Association for Computational Linguistics.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. 2023. Estimating the carbon footprint of bloom, a 176b parameter language model. *J. Mach. Learn. Res.*, 24(1).

Youmi Ma, An Wang, and Naoaki Okazaki. 2023. DREEAM: Guiding attention with evidence for improving document-level relation extraction. In *Proceedings of the 17th Conference of the European*

Chapter of the Association for Computational Linguistics, pages 1971–1983, Dubrovnik, Croatia. Association for Computational Linguistics.

Mistral AI. 2024. Large Enough | Mistral AI. https://mistral.ai/news/mistral-large-2407/. [Online; accessed 2026-08-28].

Haosi Mo, Xinyu Ma, Xuebo Liu, Derek F. Wong, Yu Li, Jie Liu, and Min Zhang. 2025. CDT: A comprehensive capability framework for large language models across cognition, domain, and task. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 3715–3734, Suzhou, China. Association for Computational Linguistics.

Shiwen Ni, Guhong Chen, Shuaimin Li, Xuanang Chen, Siyi Li, Bingli Wang, Qiyao Wang, Xingjian Wang, Yifan Zhang, Liyang Fan, Chengming Li, Ruifeng Xu, Le Sun, and Min Yang. 2025. A survey on large language model benchmarks. *Preprint*, arXiv:2508.15361.

Chenxu Niu, Wei Zhang, Yongjian Zhao, and Yong Chen. 2025. Energy efficient or exhaustive? benchmarking power consumption of llm inference engines. *SIGENERGY Energy Inform. Rev.*, 5(2):56–62.

David Pape, Jonathan Evertz, and Lea Schönherr. 2026. The silent hyperparameter: Quantifying the impact of inference backends on llm reproducibility. *Preprint*, arXiv:2605.19537.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Andrei Politov, Oleh Shkalikov, Rene Jäkel, and Michael Färber. 2025. Revisiting projection-based data transfer for cross-lingual named entity recognition in low-resource languages. In *Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, pages 499–507, Tallinn, Estonia. University of Tartu Library.

Alberto Poncelas, Maksim Tkachenko, and Ohnmar Htun. 2023. Sakura at SemEval-2023 task 2: Data augmentation via translation. In *Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023)*, pages 1718–1722, Toronto, Canada. Association for Computational Linguistics.

Zhiwen Ruan, Yixia Li, Yefeng Liu, Yun Chen, Weihua Luo, Peng Li, Yang Liu, and Guanhua Chen. 2025. G2: Guided generation for enhanced output diversity in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14116–14134, Suzhou, China. Association for Computational Linguistics.

Oscar Sainz, Iker García-Ferrero, Rodrigo Agerri, Oier Lopez de Lacalle, German Rigau, and Eneko Agirre. 2024. GoLLIE: Annotation guidelines improve zero-shot information-extraction. In *The Twelfth International Conference on Learning Representations*.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green ai. *Commun. ACM*, 63(12):54–63.

Zhengyuan Shen, Darren Yow-Bang Wang, Soumya Smruti Mishra, Zhichao Xu, Yifei Teng, and Haibo Ding. 2025. SLOT: Structuring the output of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 472–491, Suzhou (China). Association for Computational Linguistics.

Lola Solovyeva and Fernando Castor. 2026. Towards green ai: Decoding the energy of llm inference in software development. *Preprint*, arXiv:2602.05712.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Christian Wassermann, Tobias Dollenbacher, Christian Terboven, and Matthias S. Müller. 2026. Leveraging nvml gpm for nvidia gpu monitoring. In *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region Workshops*, SCA/HPCAsiaWS '26, page 59–68, New York, NY, USA. Association for Computing Machinery.

Brandon T. Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *Preprint*, arXiv:2307.09702.

Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin Lee, and 6 others. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. *Proceedings of Machine Learning and Systems*, 4:795–813.

Hui Wu, Xiaoyang Wang, and Zhong Fan. 2025. Addressing the sustainable ai trilemma: a case study on llm agents and rag. *ArXiv*, abs/2501.08262.

Zeyu Yang, Karel Adamek, and Wesley Armour. 2024. Accurate and convenient energy measurements for gpus: A detailed study of nvidia gpu's built-in power sensor. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 1–17. IEEE.

Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou, and Maosong Sun. 2019. DocRED: A large-scale document-level relation extraction dataset. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 764–777, Florence, Italy. Association for Computational Linguistics.

Aaron Yu, Jochen Weile, and Mélanie Courtot. 2026. Benchmarking llm-based information extraction tools for medical documents. *medRxiv*.

Zikang Zhang, Wangjie You, Tianci Wu, Xinrui Wang, Juntao Li, and Min Zhang. 2025. A survey of generative information extraction. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4840–4870, Abu Dhabi, UAE. Association for Computational Linguistics.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. *Preprint*, arXiv:2404.14294.

A Appendix

A.1 Software Environment

All experiments were executed under the following software stack.

- **OS & Workload manager:** SLURM 23.11.8; Operating system: Rocky Linux 9.6
- **Container runtime:** Apptainer 1.4.5 (formerly Singularity); The vLLM serving stack was packaged as a single `.sif` image built from a custom Dockerfile (`nvidia/cuda:12.8` base)
- **CUDA:** 12.8 (host driver); CUDA 11.7 inside the inference container
- **Inference server:** vLLM 0.8.4, served via HTTP with OpenAI-compatible API
- **Python:** 3.10 (container) / 3.11 (evaluation environment); key packages: `transformers`, `seqeval`, `aiohttp`, `mlflow`, `prometheus_client`
- **Energy monitoring:** NVIDIA DCGM Exporter via Prometheus (scrape interval 100 ms); custom NVML poller at 10 Hz for per-job GPU power traces
- **Module system:** Lmod 9.2

A.2 Benchmark Overview

To establish a clear foundation for our empirical analysis, Table 2 provides a comprehensive architectural overview of the energy-aware generative Information Extraction benchmark. It explicitly outlines the core dimensions of our evaluation framework, systematically categorising the targeted downstream tasks, model families, and the specific constrained decoding regimes applied.

A.3 Model deployment

All generative models used in this study (Gemma-3, Llama-3.3) were accessed in accordance with their respective community licenses and terms of use. We strictly adhered to the usage policies set forth by the model providers, ensuring that all outputs were used solely for non-commercial research purposes and that no prohibited content generation was performed. Hardware allocation was scaled according to model parameters and architectural requirements. On the NVIDIA H100 infrastructure, the Gemma model family was deployed on a single GPU, whereas the Llama models were deployed on a 2-GPU configuration. Mistral Large for Section 5.4 was hosted on 4 GPUs. Conversely, deployments on the NVIDIA A100 hardware scaled incrementally: the Gemma 4B and 12B variants were hosted on a single GPU, the Gemma 27B model required a 2-GPU setup, and the 70B model was distributed across eight A100 GPUs. Moreover, we compared Gollie (Sainz et al., 2024), fine-tuned specifically for information extraction, with other LLMs on a configuration with A100 GPUs (1 GPU).

A.4 Statistical Analysis of Run-to-Run Variability

Calibration study Reporting confidence intervals for the sweep results requires knowing how much a measurement moves when nothing changes. For ten task $\times$ model combinations, we therefore repeated a single fixed prompt under the task baseline 30 times, giving 300 runs in which the only source of variation is hardware and scheduler noise. Table 5 reports the outcome: active-window GPU energy has a coefficient of variation between 0.72% and 1.60% (mean 1.17%), while F_1 varies by at most 0.59% and usually far less.

Why distribution-free intervals The sweeps use $N = 3$, at which a Student-t interval is valid only under normality. Energy is close to normal

Dimension	Benchmark Components
Tasks	Named Entity Recognition (NER), Event Extraction (EE), Relation Extraction (RE)
Datasets	XTREME PAN-X (10 langs), MasakhaNER 2.0 (14 langs), RAMS 1.0c (En), DocRED (En)
Languages	**XTREME** (91 000 test sentences): Arabic, German, English, Spanish, French, Japanese, Korean, Russian, Chinese (10 000 each), Hindi (1 000) **MasakhaNER** (22 210 test sentences): Ewe (1 001), Hausa (1 633), Igbo (2 181), Kinyarwanda (2 235), Luganda (1 412), Chichewa (1 785), Nigerian Pidgin (1 294), Kiswahili (1 883), Setswana (996), Akan/Twi (1 211), Wolof (1 312), isiXhosa (1 633), Yorùbá (1 964), isiZulu (1 670) **RAMS 1.0c:** 871 test events (English) **DocRED:** 1 000 test documents (English)
Volume	3,488 planned configurations (model $\times$ task $\times$ language $\times$ format $\times$ prompt); each configuration repeated $N{=}3$ times $\rightarrow$ 9,291 total GPU inference runs
Model Families	**Decoders:** Gemma-3 (4B, 12B, 27B), Llama-3.3 (70B), Mistral-Large, GoLLIE **Encoders:** DREEAM, Argument Linking, NLLB+XLM-R
Output Formats	JSON, YAML, XML, DST (Domain-Specific Text)
Decoding	Unconstrained, `xgrammar`, EBNF, `Outlines`
Serving Limits	`batch_size`, `max_new_tokens`, `max_num_batched_tokens`, `block_size`, `max_num_seqs`, `max_model_len`
Hardware	**Node type A:** 8$\times$ NVIDIA A100-SXM4 (40 GiB HBM2), 2$\times$ AMD EPYC 7352 (24 cores, 2.3 GHz), 1 TiB RAM **Node type B:** 4$\times$ NVIDIA H100-SXM5 (94 GiB HBM2e), 2$\times$ AMD EPYC 9334 (32 cores, 2.7 GHz), 768 GiB RAM SLURM cluster; vLLM v0.8.4
Protocol	$N = 3$ runs; active GPU inference window isolated via NVIDIA DCGM
Metrics	**Predictive:** Micro-averaged F1, Precision, Recall **Energy:** F1-Energy Ratio (F_1/kJ), Energy per True Positive (J/TP), Energy per Extraction ($E_{\text{extraction}}$)

Table 2: Overview of the IE benchmark, presented in the paper

Parameter	Evaluated Variations
Task Specification	
Prompt Styles	P1–P9 (Across all NER models)
Output Formats	JSON DST XML YAML
Decoding Strategies	Unconstrained JSON + `xgrammar` XML + `xgrammar` DST + `ebnf` DST + `Outlines`
Concurrency Limits	
`max_concurrency`	256, 512, 1024
`batch_size`	128, 512, 1024
`max_num_seqs`	512, 1024, 2048
Context & Budgets	
`max_new_tokens`	50, 100, 150, 200, 250, 300
`max_model_len`	768, 1024, 2048
`max_num_batched_tokens`	8192, 16384, 32768
vLLM Engine Flags	
`block_size`	16, 32, 64, 128
`kv_cache_dtype`	`auto`, `fp8`
`dtype`	`bfloat16`
`enable_prefix_caching`	`True`, `False`
`enable_chunked_prefill`	`True`, `False`
Hardware Topology	
`tensor_parallel_size`	Scaled to physical GPUs

Table 3: Comprehensive taxonomy of all architectural, structural, and serving-layer parameters evaluated across the benchmark sweeps.

Parameter	Description
Output & Decoding	
Prompt style	Instruction phrasing variant (P1–P9)
Output format	JSON, DST, YAML, XML
Decoding	Unconstrained or grammar constrained (`xgrammar`, `ebnf`, Outlines)
Concurrency	
`max_concurrency`	Max parallel HTTP requests
`batch_size`	Client-side request batch size
`max_num_seqs`	vLLM in-flight sequence cap
Context & Token Budgets	
`max_new_tokens`	Per-request generation budget
`max_model_len`	Maximum prompt + response length
`max_num_batched_tokens`	Prefill window size
Engine Flags	
`block_size`	KV cache page size (tokens)
`kv_cache_dtype`	KV cache precision (`fp8`/`auto`)
`enable_prefix_caching`	Reuse KV cache across shared prefixes
`enable_chunked_prefill`	Split long prefills across steps
`calculate_kv_scales`	Dynamic FP8 KV scale computation
`tensor_parallel_size`	GPU parallelism degree

Table 4: Serving and inference parameters varied in the benchmark sweep.

(skewness within $[-0.72, 0.28]$), but F_1 is not: the EE 27B cell has skewness -2.81 and excess kurtosis 9.01. Resampling triples from each 30-run pool, a nominal 95% t-interval covers the pool mean only 0.88–0.94 of the time for energy and 0.83 for the heavy-tailed F_1 cells. We therefore use the Chebyshev interval, which assumes only i.i.d. observations with standard deviation bounded by $\sigma_{\max}$ and holds exactly at any sample size; its empirical coverage under the same protocol is 1.00 in every cell.

Thresholds Chebyshev requires $\sigma_{\max}$ as an external input, which is what the calibration study provides. Since the sample standard deviation is itself

		Energy (J)			F_1		
Task	**Model**	Mean	SD	CV %	Mean	SD	CV %
NER	4B	14,686	217	1.48	0.5831	0.00012	0.021
NER	12B	36,023	281	0.78	0.6136	0.00002	0.002
Masakha	4B	3,192	42	1.30	0.5245	0.00007	0.013
Masakha	12B	8,109	58	0.72	0.6370	0.00009	0.014
EE	4B	3,209	50	1.55	0.2592	0.00154	0.593
EE	12B	8,490	111	1.30	0.3243	0.00116	0.357
EE	27B	17,397	159	0.91	0.3621	0.00027	0.074
RE	4B	18,393	213	1.16	0.0799	0.00015	0.190
RE	12B	58,597	558	0.95	0.1674	0.00072	0.432
RE	27B	128,925	2,063	1.60	0.2244	0.00083	0.370

Table 5: Calibration study: $N = 30$ repetitions of a fixed configuration per cell (300 runs). SD and CV describe pure run-to-run noise.

uncertain, we inflate it by the one-sided χ^2 bound $\sigma \leq s\sqrt{(n-1)/\chi^2_{\alpha,n-1}} = 1.28\,s$ for $n = 30$, $\alpha = 0.05$. Applied to the worst observed energy CV of 1.60%, this gives 2.05%, and we adopt $\sigma_{\max} = 2\%$ of the mean; for F_1 the same rule yields 0.0002 (NER, MasakhaNER), 0.002 (EE) and 0.0015 (RE).

From $P(|\bar{X} - \mu| \geq \varepsilon) \leq \sigma^2/(N\varepsilon^2)$, the $(1-\alpha)$ half-width of a mean over N runs is $h = \sigma_{\max}/\sqrt{N\alpha}$. For two configurations, the variance of the difference is bounded by $2\sigma^2_{\max}/N$, so their intervals separate only if they differ by $\Delta_{\min} = \sigma_{\max}\sqrt{2/(N\alpha)}$; requiring in addition that a true effect be missed with probability at most β gives $\sigma_{\max}\sqrt{2/N}\,(\alpha^{-1/2} + \beta^{-1/2})$. At $N = 3$ and $\alpha = 0.05$ the coefficients are 2.58, 3.65 and 5.48 respectively, so with $\sigma_{\max} = 2\%$ the energy mean is known to $\pm 5.2\%$, two configurations are separable at 7.3%, and the threshold rises to 11.0% at 80% power. Monte-Carlo simulation on a two-point distribution, the worst case for which the bound is tight, confirms that the realised type-I error stays below α at $N = 3$. When a quantity is averaged over L independent languages or prompts, N is replaced by NL: $\Delta_{\min}$ tightens to 2.3% for the ten XTREME languages and 2.0% for the fourteen of MasakhaNER.

Application to the reported findings Measured against these thresholds, the effects highlighted in Section 5 are comfortably resolved: the $\pm 50\%$ swings of the `batch_size` / `max_new_tokens` / `max_num_batched_tokens` triad, the $+74.0\%$ F_1 gain of JSON+xgrammar on EE, the up-to-$7,6\times$ energy cost of FSM decoding and the -69.7% collapse of non-JSON RE all exceed $\Delta_{\min}$ by at least a factor of five. The smaller gains in Table 15 are resolved only because they are language-averaged: $+6.2\%$ for `max_num_seqs`$= 512$ and $\sim +4\%$ for `max_num_batched_tokens`$= 32768$ clear the aggregated threshold of 2.3% but not the 7.3% of a single-configuration comparison. Differences below 2% should be read as directional only.

A.5 Prompt Configurations

To systematically isolate the impact of prompt design on both performance and energy consumption, we evaluated nine prompt variants (P1–P9) for the NER task and eight variants (P1–P8) for EE and RE. The configurations span a spectrum from minimal zero-shot instructions (P1) to structured bullet-point prompts (P3–P6), rule-based verbose templates inspired by PromptNER (P7), and chain-of-thought reasoning prompts (P8). Variants P2–P6 additionally inject language-specific few-shot examples (one to three input–output pairs) into the context. For NER only, variant P9 uses instructions produced by the DSPy GEPA (Gradient-free Ensemble Prompt Adaptation) optimiser (Agrawal et al., 2026): a GEPA run is executed per language on the XTREME-PAN-X benchmark, so the resulting optimised instructions are both task-specific and language-specific, and give results on the level of the simple P1 prompt. Table 6 provides a short overview of the prompts. Full prompts are presented in the GitHub repository. Unless stated otherwise, energy figures reported throughout this work are measured on a fixed baseline configuration, enabling accurate isolation of generation cost from prompt-induced variability.

A.6 Prompt Design

Prompt design details can be found in Table 6

A.7 Baseline Experimental Configuration

Table 7 details the complete unoptimised vLLM serving configuration used to establish the baseline energy and throughput metrics prior to the ablation study.

A.8 Energy Measurement Pipeline

To ensure reproducible, high-fidelity measurements across isolated hardware environments, all energy telemetry was managed through a dedicated hardware-level monitoring stack. Because the foundational metric definitions and active-window boundaries, this section details the operational mechanics of the pipeline's execution on High-Performance Computing (HPC) infrastructure.

HPC Orchestration and Strict Isolation. To prevent thermodynamic or scheduler interference

ID	Type	Design description
P1	Basic zero-shot	Minimal instruction: task + output format only; no rules or examples.
P2	Basic one-shot	P1 + single annotated demonstration example.
P3	Rules zero-shot	Structured bullet-point rules (verbatim, no hallucination, JSON); no example.
P4	Rules one-shot	P3 rules + one annotated demonstration example.
P5	Rules two-shot	P3 rules + two annotated demonstration examples.
P6	Extended three-shot	Expanded rules (entity boundary constraints, cross-lingual note) + three examples.
P7	PromptNER rules	PromptNER-style explicit rule list (zero-shot).
P8	Chain-of-thought	Two-phase CoT: internal reasoning then final JSON; one CoT demonstration.
P9	Alt-format	Non-JSON output format (`### per / org / loc` markers).

Table 6: Prompt design summary

Parameter	NER XTREME	Masakha NER 2.0	EE RAMS	RE DocRED
Serving-Layer Constraints				
`max_model_len`	1024	1024	2048	4096
`max_new_tokens`	150	150	200	300
`max_num_batched_tokens`	8192	8192	8192	8192
`block_size`	16	16	16	16
Throughput & Concurrency				
`batch_size`	1024	1024	128	128
`max_concurrency`	256	256	256	256
`max_num_seqs`	1024	1024	512	512
Engine Optimisations				
`dtype`	`bf16`	`bf16`	`bf16`	`bf16`
`kv_cache_dtype`	`fp8`	`fp8`	`fp8`	`fp8`
`enable_prefix_caching`	`True`	`True`	`True`	`True`
`enable_chunked_prefill`	`True`	`True`	`True`	`True`
`calculate_kv_scales`	`False`	`False`	`False`	`False`
Hardware Topology				
`tensor_parallel_size`	#GPUs	#GPUs	#GPUs	#GPUs

Table 7: Naïve Production Baseline: default vLLM serving configuration per task. Serving-layer and concurrency limits are aligned to each task's input and output length; engine flags are identical across tasks. `tensor_parallel_size` is set equal to the physical GPU count of the allocation.

from parallel workloads, experiments were submitted via a SLURM workload manager. The orchestrator script enumerated the Cartesian product of all evaluated axes (model sizes, languages, constrained-decoding backends, and vLLM serving limits) and dispatched a single exclusive job per configuration to dedicated, fully isolated A100 or H100 compute nodes (e.g., the Capella and Alpha partitions).

Containerised Serving and Telemetry. To avoid host-level dependency conflicts and ensure identical execution environments across GPU architectures, the pipeline utilised a dual-container architecture via Apptainer/Singularity. The target LLM was served via vLLM within the primary container, which exposed an OpenAI-compatible REST API. Concurrently, NVIDIA's DCGM exporter (`nvidia/dcgm-exporter`) ran in a secondary container. This secondary container exposed a Prometheus-compatible HTTP endpoint and directly accessed the GPU hardware management interface, ensuring the telemetry stream remained entirely independent of the inference server's software stack.

Asynchronous Dispatch and Polling. Instead of sequential processing, the inference evaluation script dispatched prompts to the vLLM server via asynchronous HTTP requests. The concurrency of these requests was strictly bounded by a configurable semaphore to match the target throughput dynamics of the given trial. Simultaneously, a lightweight background Python process polled the DCGM exporter endpoint. By appending hardware state data (including board power, VRAM utilisation, and die temperature) to a per-job registry at 100 ms intervals, the pipeline effectively captured the transient hardware spikes induced by chunked prefilling and constrained FSM compilation.

Dynamic Alignment and Artefact Logging. Following gold-label parsing, telemetry was dynamically aligned to the exact window of $> 0\,\%$ GPU utilisation, isolating active inference energy (ΔE) from loading overhead. All resulting artefacts (latency, efficiency metrics, parsing success) were then logged to MLflow to map the Pareto frontier.

A.9 Measurement Boundaries

In any empirical evaluation of systems-level energy consumption, the boundaries of the measurement apparatus dictate the limits of the conclusions that can be drawn. This section details the operational constraints of our evaluation framework, explicitly separating phenomena that are intrinsic to generative Information Extraction (IE) from artefacts introduced by the serving stack, the hardware configuration, and the network client. By transparently defining these boundaries, we contextualise the empirical claims presented in the main text, bound the

magnitude of uncaptured systemic overheads, and identify specific telemetry bottlenecks.

Pipeline architecture and the request path. Requests reach the inference engine through three stages of queueing, only the last of which belongs to vLLM. The evaluation client partitions the dataset into groups of `batch_size` and waits for each group to complete before issuing the next. Within a group, a semaphore limits in-flight requests to at most `max_concurrency`. Each in-flight request occupies a pooled TCP connection, which is reused across requests under HTTP keep-alive. On the server side, arriving requests enter vLLM's waiting queue, from which the scheduler admits up to `max_num_seqs` sequences for concurrent decoding. Consequently, `batch_size` and `max_concurrency` shape the traffic distribution the engine observes rather than how the engine internally batches work, with the connection pool acting as an intermediary buffer.

Scope of energy telemetry. All reported energy represents GPU board power sampled from DCGM at 100 ms intervals and integrated strictly over the active inference window; system-level overheads such as host CPU, RAM, NIC, storage, conversion loss, and facility cooling are excluded. We bound the largest omitted term using utilisation traces: the supervised-encoder baseline over 10,000 samples runs 47 minutes at an average of 0.6 active cores on an AMD EPYC 9334 (main H100 architecture), yielding an estimated 6–17 kJ of CPU energy - two orders of magnitude below the GPU energy recorded over the same interval. The generative path (Gemma-3-12B) completes the identical workload in 3 minutes, reducing the host CPU term to roughly 0.8–2 kJ.

Constrained decoding represents a critical exception to this dynamic, as the host energy term is no longer negligible. Per-token mask computation is bound to the host CPU and is not vectorised across the batch, forcing the GPU to remain powered while idling. The share of active-window energy drawn while the GPU is idle measures 1.4% for unconstrained JSON and 4.0% for JSON+xgrammar, but surges to 60.7% for XML+xgrammar and 71.5% for DST+EBNF. While this cost is a genuine deployment overhead, it largely reflects GPU idle power during a host-side stall rather than active computation. Broadly, constrained decoding imposes a per-token masking cost that scales with vocabulary size and grammar complexity; however, the absolute magnitude of this penalty is an artefact of this specific Outlines/vLLM/EPYC hardware combination.

C	loss	F_1	P	R	E (kJ)
256	6.9%	0.449	0.501	0.429	22.6
512	21.1%	0.430	0.501	0.391	17.3
1024	27.9%	0.406	0.501	0.356	21.0

Table 8: Gemma-3-4B performance on XTREME, averaged across ten languages and nine prompts. Precision remains invariant while recall strictly tracks the surviving-request fraction.

Transport-layer request attrition. The client connection pool and the server fundamentally disagree on how long an idle connection remains valid: 15 seconds on the client (the `aiohttp` default) versus 5 seconds on the server (the `VLLM_HTTP_TIMEOUT_KEEP_ALIVE` default). Because the client awaits the resolution of each request group before issuing the next, every group boundary introduces a latency pause—involving response parsing, telemetry reads, and cooldown—that exceeds the server timeout. During this pause, the server closes pooled connections while the client still registers them as live. The subsequent group then attempts to issue requests into closed sockets, resulting in failure bursts strictly aligned to group boundaries.

Across the 2,972 runs supporting our results, 5,945,386 requests fail via this mechanism, overwhelmingly throwing a `ServerDisconnectedError`. By task, this comprises 5,892,489 failures for XTREME NER, 31,633 for GoLLIE, 11,712 for EE, 9,552 for RE, and none for MasakhaNER. While exhaustion of the ephemeral port pool is a natural alternative hypothesis, empirical evidence contradicts it: errors occur exclusively on *reused* connections rather than during new handshakes, and port exhaustion would require approximately 28,000 sockets in `TIME_WAIT`—a threshold never approached under our connector limit of 4,096.

Isolating the impact of request attrition on performance. If dropped requests are the sole source of error, precision—which is computed exclusively over successfully returned responses—should remain invariant, while recall should degrade proportionally to the fraction of surviving requests. Both conditions hold true in our evaluations (Table 8).

Modelling the recall ratio between $C = 256$ and $C = 1024$ as the surviving-request ratio

$(1 - \ell_{\text{hi}})/(1 - \ell_{\text{lo}})$ predicts the empirically observed ratio with $r = 0.9917$ and a mean absolute error of 1.10 pp across nine languages. Two distinct data subsets serve as controls. Hindi, representing the only 1,000-sentence split, experienced zero losses at any concurrency level: the predicted ratio was 1.000, and the observed ratio was 1.001. Conversely, Korean at $C = 1024$ shows a reversed trend: losses drop from 7.1% to 1.6%; here, the model predicts recall should *rise* by a factor of 1.059, matching the observed factor of 1.057. Correspondingly, measured energy rises $3.27\times$ because the computational work is actually performed. Because a dropped request consumes zero decoding energy, request loss simultaneously depresses measured energy alongside recall. Consequently, apparent gains in F_1/kJ at high concurrency are largely a mirage of work not performed; we therefore make no specific recommendations regarding optimal client concurrency.

Scope of affected empirical results. Request attrition is heavily concentrated in the multilingual NER sweeps, where the sheer request volume per run is highest. In contrast, MasakhaNER exhibits zero transport failures across 1,448 runs, and both EE and RE losses remain below 0.3% of total requests. Critically, the encoder/decoder comparison, the cross-family scale results, the format and decoding-regime analysis, and the task-aligned metrics are entirely independent of these affected measurements.

Two specific hyperparameters warrant clarification. `batch_size` alters measured energy by draining the scheduler and introducing idle intervals inside the measurement window, rather than by altering the engine's internal batching mechanics; it should therefore be interpreted strictly as an artefact of the testing harness. Similarly, `max_num_seqs` is swept across 512, 1024, and 2048 while client concurrency remains capped at 256; thus, the upper limit is never the binding constraint, meaning these runs do not isolate its true effect. We report both parameters for methodological reproducibility but decline to draw broader tuning recommendations from either.

Methodological recommendations for energy benchmarking. Transport-layer failures remain largely invisible within conventional reporting paradigms, as standard accuracy or energy metrics cannot distinguish a network drop from a model that validly produced an empty output. To mitigate this, we recommend three general practices for benchmarking pipelines: (1) Record requests attempted, succeeded, and failed as first-class experimental artefacts, quarantining any runs with non-zero failures; (2) Explicitly configure both client and server keep-alive timeouts, ensuring the client timeout remains strictly lower than the server's, rather than relying on framework defaults; and (3) Dispatch requests continuously under a fixed concurrency bound rather than in client-side batches, preventing the scheduler from draining and pooled connections from idling. When analysing results, a decline in recall alongside constant precision serves as a highly reliable diagnostic signature of hidden request loss.

A.10 Decomposing the Constrained-Decoding Energy Tax

Section 5.2 reports that FSM-heavy decoding modes inflate energy by up to $7,6\times$ relative to the unconstrained baseline. That figure is correct as a deployment cost but is not a single phenomenon: it mixes the cost of the output format with the cost of enforcing a grammar, and it mixes computation with waiting. This appendix separates the three.

Method. We make two changes to the comparison. First, each constrained mode is compared against the *same* serialisation format decoded without constraints (DST+EBNF against unconstrained DST, and so on), rather than against unconstrained JSON, so that format verbosity cancels and only the decoding regime varies. Second, we use the 100 ms DCGM traces to split the active window into samples with non-zero GPU utilisation and samples at zero, following the idle-energy accounting introduced in Appendix A.9. Comparisons are formed per task × model × language cell and we report medians over the matched cells.

	DST +EBNF	**XML +xgr**	**JSON +xgr**
Energy relative to same format, unconstrained			
total	6.0×	20.9×	0.4×
compute-only	1.7×	7.2×	0.4×
Time relative to same format, unconstrained			
wall-clock	8.4×	31.5×	0.3×
compute-only	2.3×	10.2×	0.3×
Idle structure inside the active window			
idle energy share	71.5%	60.7%	4.0%
zero-utilisation samples	964	2,162	5

Table 9: Median decomposition of the constrained-decoding overhead.

Most of the tax is waiting, not computing. Table 9 shows that the overhead splits roughly three-to-one. DST+EBNF costs $6.0\times$ the energy of unconstrained DST but only $1.7\times$ once zero-utilisation samples are removed; XML+xgrammar costs $20.9\times$ and $7.2\times$ respectively. The time decomposition agrees: wall-clock inflates $8.4\times$ and $31.5\times$, but compute time only $2.3\times$ and $10.2\times$. The two decompositions agreeing is expected, since GPU board power varies far less than duration, and it confirms that the effect is not an artefact of the energy counter.

The stalls are per-token, not one-off. A plausible alternative explanation is that grammar compilation imposes a single large start-up cost. The idle structure rules this out. Constrained runs accumulate their idle time across hundreds to thousands of separate 100 ms sampling windows in which GPU utilisation reads zero — a median of 964 for DST+EBNF and 2,162 for XML+xgrammar — whereas every unconstrained mode registers between three and ten. Idle time is therefore distributed throughout decoding rather than concentrated at the start, which is the signature of a per-token operation. This matches the implementation: the validity mask must be recomputed over the full vocabulary at every decode step, Outlines applies it on the host after the softmax, and vLLM does not vectorise the lookup across the sequences that its continuous batching decodes concurrently, so mask evaluation serialises while the accelerator stays powered.

Which part transfers. The compute-bound component, 1.7–$7.2\times$, follows from evaluating a grammar over a large vocabulary at every step and should appear in any engine that enforces structure this way; its magnitude scales with grammar complexity, which is why XML, with its paired opening and closing tags, costs far more than the line-oriented DST grammar. The remaining factor of roughly three is specific to this integration, and a vectorised or in-kernel mask would remove it. JSON+xgrammar demonstrates exactly that: its mask path is executed in kernels using index remapping, its idle share stays at 4.0% against 1.4% for unconstrained JSON, and it shows no idle-driven penalty. We therefore recommend JSON+xgrammar wherever schema adherence is required and energy is a constraint, and we caution that the $7,6\times$ figure should be read as a property of the current Outlines/vLLM combination rather than of grammar-constrained generation as such.

Caveats. The matched cells are dominated by Gemma-3-4B (9 of 10 for DST+EBNF, 11 of 13 for XML+xgrammar); the 12B cells give a broadly similar picture, with DST+EBNF at $7.9\times$ total and $2.6\times$ compute-only, and XML+xgrammar at $14.8\times$ and $4.8\times$. Only three matched cells exist for JSON+xgrammar, so we treat its total-energy ratio as indicative and rest the JSON conclusion on the idle-share measurement, which is available for all 112 runs of that mode. Finally, the split between computing and waiting is resolved only at the 100 ms telemetry granularity, so it bounds rather than exactly partitions the two components.

A.11 Results of Scaling Architectures

Detailed results about Scaling Architecture findings can be found in Table 10

A.12 Concurrency and the Serving Layer

Concurrency and Serving-Layer Bottlenecks. We observe that increasing the client-side concurrency limit (C) paradoxically degrades measured extraction quality on large XTREME NER splits. A transport-layer ablation reveals this is a client-side networking artifact rather than an operating system limitation. Our evaluation harness processes XTREME requests in batches of 1,024. Between batches, the system idles to read energy counters and parse responses. This idle duration plausibly exceeds the server's five-second keep-alive window while remaining within the client's fifteen-second threshold. Consequently, the client attempts to reuse sockets that the server has already closed, causing requests to fail and be recorded as empty responses. Higher values of C leave more connections idle at these batch boundaries, exacerbating request loss.

Recall as a Systems Diagnostic. Because a failed request yields an empty output, this bottleneck manifests as a sharp drop in Recall while Precision remains unaffected. For example, on the XTREME benchmark using Gemma-3-4B, increasing C from 256 to 1024 causes macro Recall to degrade (from 0.436 to 0.352), whereas macro Precision remains perfectly stable (at approximately 0.501). This signature uniquely identifies lost, rather than degraded, outputs. Entity yield falls with it, to 91.9% of baseline at C=512 and 77.1% at C=1024, while three values of `max_num_seqs` leave it unchanged (100.0–

Task	Gemma-3-27B (TP=1)		Llama-3.3-70B (TP=2)		Llama-3.3-70B (TP=4)		Mistral-Large (TP=4)	
	F_1	F_1/kJ	F_1	F_1/kJ	F_1	F_1/kJ	F_1	F_1/kJ
NER XTREME (de)[†]	**0.703**	**8.26**	0.635	2.99	0.622	2.15	0.665	1.67
EE (RAMS)	**0.327**	**12.07**	0.302	6.54	0.306	3.42	0.287	2.21
RE (DocRED)	0.178	**1.79**	0.151	0.74	0.173	0.61	**0.208**	0.51
MasakhaNER (hau)	**0.688**	**35.40**	0.661	16.82	0.663	13.03	0.618	7.07

Table 10: Scaling past the 70B reference point, under the baselines. F_1/kJ in units of 10^{-3}, computed from ΔE (Eq. 1) over all GPUs the job held. Best per task in bold. Mistral-Large requires four H100s, so Llama-3.3-70B is reported both at its minimum allocation (TP = 2) and hardware-matched (TP = 4). Efficiency is ordered by deployed capacity on every task: Gemma-3-27B on a single GPU is 2.7–3.8× more efficient than Llama-3.3-70B at TP = 4 and 3.5–5.5× more than Mistral-Large, which is last throughout despite the highest F_1 on RE. [†]On German NER the nine-prompt means are 0.665 (Mistral-Large) and 0.622 (Llama-3.3-70B), but the entire gap comes from the per-language GEPA prompt (P9, Table 6), on which Llama-3.3-70B collapses to 0.203; over the eight remaining prompts the models are level (0.664 vs. 0.675).

(a) Entity Yield vs. Swept Parameter

Run (B, seq, C)	**Entities**	**% vs base**	**P**	**R**
(128, 1024, 256)	13,824	105.7%	0.654	0.650
(512, 1024, 256)	13,540	103.5%	0.654	0.637
Base (1024, 1024, 256)	13,085	100.0%	0.654	0.615
(1024, 512, 256)	13,127	100.3%	0.654	0.617
(1024, 2048, 256)	13,088	100.0%	0.654	0.616
(1024, 1024, 512)	12,028	91.9%	0.655	0.567
(1024, 1024, 1024)	10,091	77.1%	0.655	0.473

(b) Transport Ablation at C=1024

Transport	**R**	**P**	**TIME_WAIT**	**E (kJ)**
Pooled (default)	0.490	0.655	117 / 654	19.5
Fresh per request	0.651	0.655	3,849 / 6,539	23.5
Pool capped to C	0.478	0.655	130 / 637	19.3

Table 11: Concurrency diagnostics. (a) Gemma-3-4B on XTREME German, mean over nine prompt styles; B is the client batch size, seq is `max_num_seqs` and C the concurrency ceiling. Yield is flat across `max_num_seqs` (100.0–100.3%), which never binds because C caps the engine below every value tested, and falls only with C (100 → 91.9 → 77.1%). Smaller batches recover yield above the baseline, consistent with a shorter interval between batches. Precision is invariant at 0.654–0.655 throughout while recall tracks yield, the signature of discarded rather than degraded outputs. (b) Transport ablation at C=1024 with TIME_WAIT sockets polled directly (mean / max). Opening a fresh connection per request restores recall at unchanged precision while producing 33× more TIME_WAIT sockets, which rules out ephemeral-port exhaustion; capping the pool to C does not help, which rules out surplus idle connections. The failure tracks connection reuse, at a cost of 4 kJ per run in handshakes.

100.3%) because C caps the engine below every value tested (Table 11a). Furthermore, the severity of the drop depends on whether a run spans more than one batch, not on model scale. Large datasets requiring multiple batches suffer severe Recall losses (11–32%), whereas smaller datasets (e.g., Hindi, which fits into a single batch) experience zero loss.

Isolating the Connection-Reuse Mechanism. We confirmed this mechanism by comparing the default pooled transport against a configuration that forces a new connection for every request, holding C=1024 constant. Disabling connection reuse fully restores Recall (from 0.490 to 0.651) while holding Precision constant, despite generating roughly 33× more `TIME_WAIT` sockets (3,849 vs. 117 on average; Table 11b). This rules out ephemeral-port exhaustion as the bottleneck; capping the pool to exactly C leaves Recall at 0.478, ruling out surplus idle connections as well. Failures therefore stem strictly from stale connection reuse.

Latency, Scale, and Energy Implications. This client-side failure mode persists across all model sizes (4B to 70B), maintaining stable Precision while degrading Recall by 11–24%. It is circumvented when the batch size falls below the concurrency ceiling, as observed in EE and RE tasks where a batch size of 128 preempts the C=256 limit. Crucially, the fresh-connection transport is a diagnostic rather than a deployable fix, and its cost is smaller than it appears: energy rises from 19.5 to 23.5 kJ, but the recovered outputs offset almost all of it (F_1/kJ 27.7 vs. 28.5×10^{-3}). Concurrency (C) is therefore not merely a hyperparameter for maximising throughput; it is a critical system setting that must be carefully co-optimised with batch size, dataset scale, and network keep-alive configurations to ensure reliable and energy-efficient evaluation.

A.13 Comparison with GoLLIE: Fine-Tuned Specialist vs Zero-Shot Generalist

GoLLIE (Sainz et al., 2024) is a 7B-parameter model (CodeLLaMA-7B) fine-tuned Specifically for information extraction using annotated, code-style supervision. We ran GoLLIE under the same serving infrastructure and hardware (Configuration with A100: single GPU) on the German NER split of XTREAM, enabling a direct comparison.

Quality vs. energy efficiency. Despite requiring no task-specific fine-tuning, Gemma-3-4b-it outperforms GoLLIE on German NER by a substantial margin (F1 = 0.616 vs. 0.521, +18%) while consuming *less* GPU energy (37,807 J vs. 46,654 J, −19%). The combined effect is a 50% advantage in quality-per-joule (F1/J: 1.67×10^{-5} vs. 1.12×10^{-5}). GoLLIE does achieve notably higher throughput (59.8 Tok/s vs. 14.2 Tok/s for Gemma-3-4b-it), because its code-style output format is far more compact than the natural-language explanations produced by instruction-tuned models. However, this latency advantage does not translate into an energy advantage under our batch-serving regime, where total joules are dominated by the prefill cost of the input context rather than the generation length.

Scaling penalty for large generalist models. Scaling up the instruction-tuned model beyond 4B parameters erodes the energy advantage rapidly. Gemma-3-12b-it and Gemma-3-27b-it reach higher German F1 scores (0.712 and 0.706, respectively) but at 65% and 3.3× more energy than GoLLIE, pushing their F1/J (6.6×10^{-6}) well below GoLLIE's. LLaMA-3.3-70B presents the starkest trade-off: it uses 7.3× more energy than GoLLIE yet achieves *lower* F1 (0.471 vs. 0.521), making it the least energy-efficient option by a wide margin. These results suggest that for NER on A100 hardware, a compact 4B instruction-tuned model occupies a Pareto-optimal position that neither fine-tuned specialists nor Large-scale generalists can match without task-specific adaptation.

Implications for deployment. The comparison reinforces a practically important finding: the energy cost of fine-tuning a specialist model such as GoLLIE (which is a one-time, offline expense) can be amortised across many inference calls, but the *per-query* Energy at inference time is what dominates in production. Our results show that, at inference time, a zero-shot 4B model is already more energy-efficient than a 7B fine-tuned specialist on the same NER task. If the serving configuration is further optimised using the recommendations in specifically `max_num_seqs=512` and `max_new_tokens=100` for NER, the gap widens: Gemma-3-4b-it would consume an estimated additional 12–20% less energy, pushing its advantage over GoLLIE to roughly 2× in F1/J.

A.14 Hyperparameter Optimisation vs Single-Parameter Sweeps

Parameter	Value	Mean $J_{\text{TP}}^{\text{corr}}$	Min $J_{\text{TP}}^{\text{corr}}$	Δ vs BL (%)
kv_cache_dtype	auto	4.41	4.34	-15.5
	fp8 (BL)	5.22	4.54	0.0
block_size	16 (BL)	5.22	4.54	0.0
	32	34.78	5.60	+566.2
enable_prefix_caching	True (BL)	5.22	4.54	0.0
	False	4.62	4.52	-11.5
chunked_prefill	True (BL)	5.22	4.54	0.0
	False	5.32	4.88	+1.9
enforce_eager	False (BL)	5.22	4.54	0.0
	True	6.67	6.55	+27.7
max_new_tokens	50	5.83	5.64	+11.7
	100 (BL)	5.22	4.54	0.0
	150	28.45	5.17	+445.0
	200	27.87	4.36	+433.8

Table 12: One-Parameter-at-a-Time (OPAAT) sweep results. Baseline (BL) is indicated. Δ is the relative change of the mean $J_{\text{TP}}^{\text{corr}}$ against the baseline.

Hyperparameter	Optimal Value
block_size	32
chunked_prefill_enabled	False
enable_prefix_caching	False
enforce_eager	False
kv_cache_dtype	auto
max_new_tokens	150
Best Objective ($J_{\text{TP}}^{\text{corr}}$)	**4.25 J/TP (-18.6% vs BL)**

Table 13: The globally optimal configuration discovered by the Optuna HPO engine after 56 trials.

Hyperparameter	Importance Variance (%)
max_new_tokens	43.1
enforce_eager	21.3
chunked_prefill_enabled	18.0
enable_prefix_caching	10.6
block_size	5.5
kv_cache_dtype	1.5

Table 14: Global parameter importance calculated via fANOVA over the HPO trial history. Values indicate the percentage of variance in $J_{\text{TP}}^{\text{corr}}$ explained by each parameter.

To minimise the corrected Energy per True Positive (J/TP) during inference, we evaluated the serving configuration of the 4B model using two distinct search strategies: a One-Parameter-at-a-Time (OPAAT) sweep and Bayesian Hyperparameter Optimisation (HPO) via Optuna. The OPAAT approach isolates individual variables against a default baseline and successfully identifies immediate efficiency gains. For instance, switching the

KV cache data type to `auto` and disabling prefix caching yielded energy reductions of 15.5% and 11.5%, respectively. However, this isolated approach obscures critical inter-parameter dynamics and can lead to flawed conclusions about the utility of absolute parameters.

The Bayesian HPO strategy overcomes this limitation by exploring the joint configuration space, ultimately uncovering a global optimum that reduces the energy metric by 18.6% relative to the baseline. Crucially, HPO exposes hidden, non-linear interactions between variables. A prime example is the `block_size` parameter: in the isolated OPAAT sweep, setting `block_size=32` resulted in a catastrophic 566% energy penalty. Yet the HPO engine determined that, when combined with disabled chunked prefill and an `auto` KV cache, `block_size=32` is necessary for the most energy-efficient configuration.

Furthermore, variance analysis (fANOVA) of the HPO trials completely reordered the assumed hierarchy of parameter importance. While the OPAAT sweep suggested that the KV cache data type was the critical lever for energy savings, fANOVA revealed it accounts for the lowest global importance (1.5%) across the joint space. Instead, maximum token limits and enforcing eager execution drove the vast majority of performance variance. Ultimately, the HPO search yielded a marginal 3.7% efficiency gain over the best OPAAT combination. Because this nominal improvement falls well within the natural run-to-run variance of GPU telemetry and workload scheduling, the substantial computational overhead required for joint optimisation is essentially unjustified.

A.15 Optimal Serving Configurations

Table 15 distils the one-parameter-at-a-time ablation results into eight actionable serving-layer settings; each row states the recommended value, its target task scope, and the median energy and quality-per-energy shifts relative to the shared baseline configuration.

A.16 Influence of the Formats and Decoding

Table 16 reports, for each task and model, the baseline F1 and energy under unconstrained JSON decoding, along with the best-performing output format, quantifying the F1 gain and energy change achieved by switching to the optimal format.

A.17 Comparison of the Architectures and Distribution Methods

Baseline energy efficiency across hardware architectures. Under a shared baseline (`kv_cache_dtype=fp8`, `block_size=16`, `max_new_tokens=150`, `batch_size=1024`), the H100 is the more efficient architecture on every task and scale we measure. For 12B multilingual NER (XTREME) it expends 60,871 J at $F_1 = 0.492$ against 77,851 J at $F_1 = 0.507$ on the A100. The margin widens with the smaller models: the 4B model reaches 0.046 and 0.090 F_1/kJ for Event Extraction and African NER (MasakhaNER) on H100, against 0.039 and 0.071 on A100. At 27B both partitions fall by roughly the same factor, to 0.012 and 0.031 on H100 and 0.007 and 0.017 on A100, so the relative gap is stable across scale rather than concentrated at the small end. The two partitions differ in sample count and vLLM version, so these ratios are indicative of the generational difference rather than a controlled measurement of it. Across tasks, the 4B model on H100 consistently delivers the highest F_1/kJ, confirming that memory bandwidth and tensor-core density disproportionately benefit throughput-bound inference.

Divergent parameter sensitivity between GPU generations. H100 and A100 respond differently to serving-layer tuning. KV cache quantisation shows the clearest contrast: on H100, FP8 is the better default on all four tasks, saving 1.6–7.4% energy for +1.7% to +6.9% F_1/kJ, with the gain largest on the two English document-level tasks. On A100, FP8 similarly benefits EE and RE but harms MasakhaNER (+11% energy, −14.2% F_1/kJ), likely due to the precision-sensitive token distributions of morphologically complex languages. Block size also diverges: H100 penalises every size above 16, and the penalty grows with the block (−16% to −22% F_1/kJ at 128 on NER, MasakhaNER and EE), while A100 benefits from sizes 32 and 64 (+37.9% and +23.5% for EE), reflecting generational shifts in paged-attention overhead. The largest architecture-agnostic lever is `batch_size=256` on the document-level tasks, cutting energy 34.9% and raising F_1/kJ by 50.6% on H100 and 50.4% on A100. It does not generalise to the sentence-level NER tasks, where the baseline batch is already 1,024 and reducing it is strongly negative (−16.6% F_1/kJ at 512, −45.9% at 128).

Parameter Configuration	Thermodynamic & Predictive Impact
`enable_chunked_prefill` = false *(Event Extraction)*	Removes the prefill/decode interleave on RAMS' long, dense prompts. The largest single gain we measure: saves 42% in Energy(J) for $+68\%$ F_1/J, at a cost of 1.0% absolute F_1. The effect is EE-specific and does not transfer to NER, where it is within noise.
`batch_size` = 256 *(EE / RE)*	Raises the client batch above the default 128, alleviating memory bandwidth contention. Saves 35% in Energy(J) and improves F_1/J by 51%, for 2.2% absolute F_1. Positive in all eight task–model cells, the most reliable setting in the sweep.
`max_new_tokens` = 200 *(Relation Extraction)*	Tightens the decode budget from the default 300, bounding complex reasoning to minimise J/TP. Saves 36% in Energy(J) for $+50\%$ F_1/J at 0.4% absolute F_1. Tightening further to 150 reaches $+92\%$ but costs 5.4% F_1.
`block_size` = 32 *(Event Extraction)*	Halves the paged-attention block from the default 16. Reduces Energy(J) by 14% for $+21\%$ F_1/J with accuracy slightly improved ($+0.8\%$). Larger blocks are uniformly worse: 64 and 128 raise energy and lower F_1/J on every task.
`max_new_tokens` = 100 *(NER / MasakhaNER)*	Eliminates post-extraction over-generation. Lowers J/Gen and reduces Energy(J) by 4% (XTREME) to 7% (Masakha), for $+3.5\%$ and $+5.7\%$ F_1/J, with absolute F_1 unchanged. On XTREME a 50-token budget does better still (-7% Energy, $+8\%$ F_1/J), but costs 2.1% F_1 on Masakha.
`kv_cache_dtype` = auto *(A100 Deployments Only)*	Prevents the 20–40% throughput penalty observed with `fp8` on the A100. The H100 result is the reverse: `fp8` is the better default there, and forcing `auto` costs 2–7% F_1/J across all four tasks.

Table 15: Summary of optimal serving configurations and their median thermodynamic/predictive shifts against the per-task baseline of Table 7. Each parameter cell is paired with the baseline cell sharing its task, model, language and prompt style before the median is taken, so every comparison is like-for-like. Gains concentrate on EE and RE; on NER and MasakhaNER no single parameter exceeds $+8\%$ F_1/J.

Task	Model	BL F1	Best F1	ΔF1 (%)	Best Format	BL E (J)	ΔE (%)
NER	Gemma-3-4B	0.4492	**0.4898**	**+9.0**	$\mathrm{DST}_{\varnothing}$	31,383	−4.7
	Gemma-3-12B	0.4924	**0.5175**	**+5.1**	$\mathrm{YAML}_{\varnothing}$ ‡	60,871	−2.1
	Gemma-3-27B	0.5089	0.5089	+0.0	$\mathrm{JSON}_{\varnothing}$	89,476	+0.0
	Llama-3.3-70B	0.4625	**0.4880**	**+5.5**	$\mathrm{JSON}_{\mathrm{xgr}}$	235,019	+2.6
EE	Gemma-3-4B	0.2449	**0.2630**	**+7.4**	$\mathrm{JSON}_{\mathrm{xgr}}$	5,389	+12.9
	Gemma-3-12B	0.1861	**0.3238**	**+74.0** §	$\mathrm{JSON}_{\mathrm{xgr}}$	5,159	+144.1
	Gemma-3-27B	0.3271	0.3271	+0.0	$\mathrm{JSON}_{\varnothing}$	27,106	+0.0
	Llama-3.3-70B	0.3019	**0.3114**	**+3.2**	$\mathrm{DST}_{\varnothing}$	23,067	+25.8
RE	Gemma-3-4B	0.0666	**0.0731**	**+9.7**	$\mathrm{YAML}_{\varnothing}$	24,042	−0.1
	Gemma-3-12B	0.1468	0.1479†	+0.7†	$\mathrm{JSON}_{\mathrm{xgr}}$	56,635	−0.4
	Gemma-3-27B	0.1784	0.1784	+0.0	$\mathrm{JSON}_{\varnothing}$	99,909	+0.0
	Llama-3.3-70B	0.1507	**0.1749**	**+16.1**	$\mathrm{DST}_{\varnothing}$	102,021	−13.9

Table 16: Cross-task summary: best format and decoding regime per task and model. Baseline (BL) utilises unconstrained JSON. Best format denotes the highest mean F_1 across all evaluated modes. Energy is the whole-run ΔE in Joules, summed over every batch and every GPU the job held. Bold cells mark cases where an alternative format improves over baseline. Llama-3.3-70B figures come from the corrected TP = 2 reruns; RE figures reflect the corrected prompt templates, in which every serialisation format receives the same closed relation vocabulary. †Difference below the resolution of a three-run comparison (Appendix A.4); the two formats are effectively tied. ‡Margin over $\mathrm{JSON}_{\mathrm{xgr}}$ (0.5116) is 0.006; the two are effectively tied. §

GPU inference cost (J/TP) analysis and cross-task implications. The Joules per True Positive (J/TP) metric quantifies extraction cost independently of recall–precision trade-offs. On H100, NER ranges from 4.45 J/TP (4B, DST) to 9.09 J/TP (27B, DST), and the cost of scale is steep: the 70B model needs 29.3 J/TP under DST and 44.9 under unconstrained JSON, six to ten times the 4B figure. For EE, J/TP varies from 7.22 (4B, DST) to 23.74 (27B, DST), a 3.3-fold spread driven by generation length rather than extraction quality. On RE, once every serialisation receives the same closed relation vocabulary, the format spread closes almost entirely: at 12B, unconstrained DST costs 27.5 J/TP against 35.1 for unconstrained JSON, so the lighter serialisation is in fact the cheaper one per recovered triple. Ultimately, the H100's efficiency advantage concentrates in one regime: deploying

small models with concise formats for high-volume extraction, where its compute density absorbs the decoding overhead at a lower energy tax than the A100.

A.18 GPU Utilisation and Energy Efficiency

In throughput-bound LLM serving, GPU utilisation is proxied by the number of concurrent requests (C). Low C starves the decoding engine, burning energy without generating tokens. As C increases, the scheduler keeps tensor cores occupied, amortising per-request overhead. We exploit a concurrency sweep ($C \in \{256, 512, 1024\}$) across hardware configurations to evaluate this relationship via F_1/kJ and J/TP.

Table 17 reports efficiency across concurrency levels, and the result is not uniform: the benchmark decides the sign. On MasakhaNER, raising C behaves as the utilisation argument predicts. The 4B model improves from 3.76 to 2.99 J/TP (-20%) while gaining accuracy (F_1 0.430 $\rightarrow$ 0.458), and the 12B and 27B models improve by 11% and 11%. On XTREME the effect reverses. Every model gets worse: 4B from 5.37 to 7.21 J/TP ($+34\%$), 12B from 8.50 to 10.51 ($+24\%$), 27B from 13.14 to 14.35, and F_1 falls by 0.02 to 0.04 points in each case. F_1/kJ on XTREME peaks at C=256 or 512 and never at 1024.

XTREME runs 10,000 sentences per language at `batch_size`=1024, giving nine boundaries per run; MasakhaNER languages run 996 to 2,235 sentences, giving zero to two, 1.1 on average. MasakhaNER is therefore close to the clean case the utilisation model describes, while on XTREME the apparent energy saving at high C is largely the energy of requests that were never served. Two further patterns hold on both benchmarks: energy savings are largest in the first doubling, from 256 to 512, and the 70B model is the least responsive at either scale, gaining nothing on MasakhaNER and 12% on XTREME before regressing at C=1024, as its larger KV cache footprint already occupies High Bandwidth Memory consistently.

The practical rule is therefore conditional rather than global. On workloads whose runs fit in one or two client batches, raise C to the hardware memory ceiling: the energy savings are real and reach 24% at 4B on MasakhaNER for no loss of accuracy. On workloads that span many batches, C must be held at the baseline until the keep-alive mismatch is corrected, after which the sweep should be repeated; the gains currently reported at C=1024 on XTREME are not attributable to utilisation.

HW	Task	Mod.	C	F_1	E (kJ)	F_1/kJ ↑	J/TP↓
H100	XTREME	4B	256	0.449	313.8	1.43×10^{-3}	5.37
			512	0.430	291.8	1.47×10^{-3}	5.75
			1024	0.406	311.4	1.30×10^{-3}	7.21
		70B	256	0.463	2350.2	1.97×10^{-4}	44.87
			512	0.433	2055.5	2.11×10^{-4}	39.57
			1024	0.425	2033.4	2.09×10^{-4}	46.23
	MASAKHA	4B	256	0.430	66.6	6.47×10^{-3}	3.76
			512	0.445	51.2	8.70×10^{-3}	2.92
			1024	0.458	50.7	9.04×10^{-3}	2.99
		70B	256	0.666	666.7	9.99×10^{-4}	31.34
			512	0.637	581.2	1.10×10^{-3}	32.52
			1024	0.634	649.2	9.77×10^{-4}	34.85
A100	XTREME	4B	256	0.531	606.7	8.76×10^{-4}	21.0
			512	0.492	445.1	1.11×10^{-3}	15.2
			1024	0.508	402.0	1.26×10^{-3}	13.3
		70B	256	0.357	4116.7	8.67×10^{-5}	237.6
			512	0.352	2948.0	1.19×10^{-4}	206.4
			1024	0.347	2678.0	1.30×10^{-4}	188.1
	MASAKHA	4B	256	0.552	115.1	4.80×10^{-3}	6.7
			512	0.536	73.1	7.34×10^{-3}	4.9
			1024	0.525	61.6	8.52×10^{-3}	5.4
		70B	256	0.705	1702.0	4.14×10^{-4}	96.6
			512	0.697	1471.7	4.74×10^{-4}	86.3
			1024	0.690	1561.6	4.42×10^{-4}	90.1

Table 17: Efficiency metrics across concurrency (C), tasks, and models, with 4B and 70B shown for both benchmarks on both partitions. E is the total corrected energy summed over all languages; J/TP is the median across languages and prompts. ↑ / ↓ indicates the desired direction.