E1: A First-Order Energy Metric for AI Model Architectures

Anonymous authors
Paper under double-blind review

Abstract

Energy is a primary constraint on AI inference at scale, yet models are designed and compared on accuracy alone. A trustworthy energy number requires an optimized implementation, which arrives long after the architecture is fixed, and only for the ideas that well-resourced teams choose to engineer. We introduce *E1*, an infrastructure-agnostic energy metric computed from unoptimized PyTorch code. *E1* lowers the model to a graph of einsums, searches the tiling and fusion schedules of that graph on an abstract accelerator architecture with a finite on-chip buffer, and sums the energy of the arithmetic operations and off-chip data movement. The resulting *E1* score represents what an ideally optimized implementation of the model could achieve, without any system optimization work by the developer. We validate *E1* against energy measured on NVIDIA H100 GPUs for individual kernels, production LLM-serving kernels across implementations and across two years of their development, and for full model forward passes. *E1* tracks measured energy over three orders of magnitude; optimized kernels land 2–4$\times$ above it and eager PyTorch about an order of magnitude higher; and when one mixture-of-experts model runs under two implementations whose measured energy differs by 8$\times$, *E1* does not move. Our target is a reliable, infrastructure-agnostic metric that will enable fair accuracy–energy studies. We plan to open-source *E1*.

1 Introduction

AI models are judged primarily on accuracy, yet their sustainable and scalable deployment depends on cost. Reliable cost measurements (energy, latency, throughput, dollar expense) arrive much later: making a model efficient requires hardware-specific kernel engineering, compiler support, optimized libraries, and runtime integration, all of which take time and deep systems expertise. A fair energy comparison between two architectures therefore needs an optimized implementation of each, and only a few well-resourced labs can afford that engineering for every idea they want to test; a research group without it can show that its architecture is accurate, but not that it is efficient.

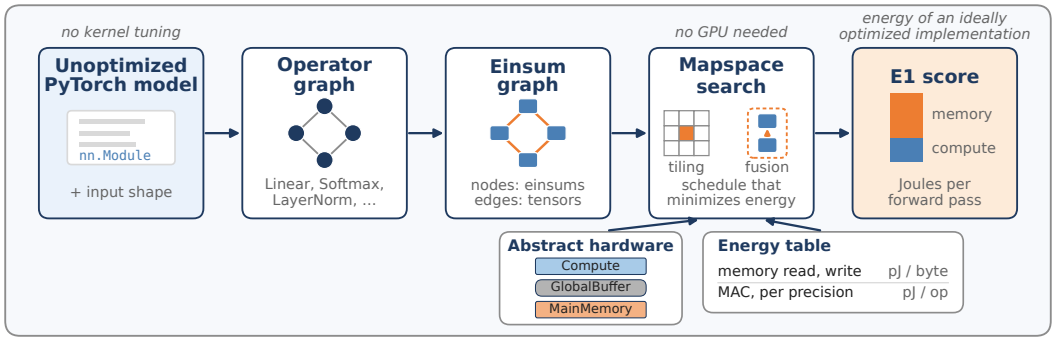


Figure 1: *E1* end-to-end pipeline. PyTorch code is lowered to an einsum graph; a search over the tiling and fusion mapspace of an abstract hardware model yields the energy of an optimized implementation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Estimating cost early is hard because cost depends on the implementation as much as on the model. Attention is a telling example: Transformers appeared in 2017, while FlashAttention and its successors, which cut the energy of the same computation by orders of magnitude through tiling, fusion, and memory placement, arrived in 2022–2024 (Vaswani et al., 2017; Dao et al., 2022; Dao, 2024; Shah et al., 2024); FlashInfer does the same for LLM-serving kernels (Ye et al., 2025). Architecture choices are made much earlier, and new ideas are judged on hardware and software tuned for old ones (Hooker, 2021; Geens et al., 2026). Hardware-independent proxies avoid this dependence but miss what matters most: FLOP counts ignore data movement, underestimating measured energy (Fernandez et al., 2025). For example, on the same GPU the energy per arithmetic operation differs by three orders of magnitude between a large GEMM and an element-wise kernel (§3.1). Measurements on real hardware capture data movement but tie the result to one accelerator and to the maturity of its software stack, which is why Intelligence per Watt evaluates model–accelerator pairs rather than models (Saad-Falcon et al., 2025); in §3.4 the same mixture-of-experts model, on the same GPU and prompt, uses $8\times$ more energy under one implementation of its expert layers than under another. Existing approaches to energy efficiency (Appendix A) therefore fall short on one or more of three axes: they measure cost empirically, favoring mature implementations over new ideas; they are infrastructure-dependent, blurring whether efficiency comes from the architecture or from a favorable mapping to specific hardware; and they disregard memory movement, a critical energy component for memory-bound operations.

The key question in this paper is: *can we assess the energy-efficiency potential of AI algorithms before systems optimization?* To this end we introduce *E1*, an infrastructure-agnostic energy metric that estimates the *optimized energy efficiency* potential of *unoptimized code* for AI models (Figure 1). The idea rests on a simple decomposition. The energy of a model has an arithmetic part and a data-movement part. The arithmetic part is fixed by the model: every multiply-accumulate operation (MAC) must be executed, so *E1* counts them and attributes to each a per-operation cost. The data-movement part is not fixed: it depends on how the implementation is scheduled, because tiling and fusion decide which tensors stay on-chip and which travel to main memory. *E1* therefore converts the model into a graph of einsums and searches the space of tiling and fusion schedules on an abstract accelerator, using an input table of per-operation and per-byte energy costs. The result is a first-order estimate of the energy an optimized implementation could achieve, available before a custom kernel for specific hardware has to be written, and computed without a GPU or any kernel tuning.

We focus on energy for three reasons. First, energy is a primary limitation for AI deployments: as inference demand grows, serving capacity becomes the product of datacenter power capacity and hardware energy efficiency, which has pushed the field toward gigawatt-scale datacenters (Epoch AI, 2025) and chip manufacturers toward tokens per watt instead of peak FLOP/s (NVIDIA, 2025a). Second, energy avoids choosing between latency and throughput: the same energy can be spent quickly or amortized over many requests (Pope et al., 2023), so *E1* estimates energy only and makes no claim about latency. Third, monetary expense depends on cloud pricing, workload mix, utilization, and service-level policy (Griggs et al., 2024), which makes it difficult to estimate during algorithmic research.

The scope of this paper is the metric itself and its validation; the accuracy–energy studies and architecture exploration it is meant to enable are beyond it. Our contributions are:

- We introduce *E1*, a first-order energy metric that estimates the optimized energy potential of unoptimized model code.
- We build an end-to-end pipeline that lowers PyTorch models to einsum graphs, searches fusion and tiling schedules, and reports compute and memory energy breakdowns.
- We validate *E1* against H100 measurements on microbenchmarks, KernelBench, production LLM-serving kernels across implementations and across two years of their development, and full model forward passes.
- We set the target of a reliable, infrastructure-agnostic metric for accuracy–energy studies of model architectures; this paper is about the tool, not about those studies, and we plan to open-source it for them.

2 *E1* ENERGY METRIC

E1 takes an unoptimized PyTorch `nn.Module` and one set of representative inputs (batch size, sequence length, and so on), and returns the energy, in Joules, that an optimized implementation of that model could achieve on an abstract accelerator, together with a breakdown into compute and memory energy by precision. It runs on a CPU in minutes, needs no GPU and no kernel tuning, and takes a single hardware-specific input, an energy table. It is built for the stage before optimized code exists. We first walk through the pipeline end to end (§2.1), then describe the abstract hardware and energy table it runs against (§2.2), the energy model and mapspace search at its core (§2.3), and finally the scope decisions that keep *E1* simple and hardware-agnostic (§2.4).

2.1 END-TO-END PIPELINE

Figure 1 shows the four stages of *E1*.

Stage 1: PyTorch → operator graph. The user runs the model's `forward` once on representative inputs. *E1* uses Solar (NVlabs, 2025), an open-source PyTorch model analysis toolkit, to capture the `torchview` ComputationGraph (Kurttutan, 2024): a directed graph whose nodes are operators (`Linear`, `Softmax`, `LayerNorm`, ...) and whose edges are operands annotated with tensor shapes and data types.

Stage 2: Operator graph → einsum graph. Solar then lowers each operator to one or more einsums: tensor expressions that name the dimensions of their inputs and output and sum over the dimensions that appear only in the inputs (matrix multiplication is `ik,kj→ij`). Dense operators become contractions, element-wise activations become unary einsums, and composite operators such as scaled dot-product attention or SwiGLU are expanded into their constituent einsum chains. Shape-only operators (`view`, `transpose`, ...) are recorded but excluded from the energy total because they move no data on a typical GPU. Index-heavy operators such as embedding lookups are accounted for by their data movement (§2.2). The result is a graph whose nodes are einsums and whose edges are the tensors flowing between them. This representation exposes exactly what *E1* needs: arithmetic work, tensor sizes, and which einsum produces the tensor that another consumes.

Stage 3: Einsum graph → schedule. This is the step that separates *E1* from FLOP counting. The einsum graph fixes how much arithmetic must be done, but not how much data must move; that depends on how each einsum is tiled and whether neighboring einsums are fused. *E1* searches this mapspace on an abstract accelerator with an unbounded off-chip memory and a finite on-chip buffer (§2.2) for the schedule that minimizes the energy model of §2.3, standing in for the hardware-specific engineering an optimized implementation would need.

Stage 4: Schedule → energy. The chosen schedule determines how many bytes each tensor reads from and writes to off-chip memory, and the einsum extents determine the arithmetic counts. *E1* multiplies both by the energy table and sums the compute and memory terms into the scalar *E1* score. It also reports the breakdown of operation counts and energy by precision and by component (reads, writes, and MACs), which shows where a model spends its energy.

2.2 HARDWARE ABSTRACTION AND ENERGY TABLE

E1 targets "Snowcat", an abstract accelerator with three components (Figure 2): an unbounded off-chip *MainMemory* (HBM/DRAM), a limited on-chip *GlobalBuffer*, and a compute unit. The GlobalBuffer stands in for all of an accelerator's on-chip storage, treated as a single level with no internal cost; Snowcat assumes nothing else about the microarchitecture, only the boundary between on-chip and off-chip storage that every accelerator with external memory has. This makes *E1* infrastructure-agnostic: a broad family of accelerators (e.g., GPUs, TPUs (Jouppi et al., 2017), RDUs (Emani et al., 2021)) fit the abstraction. SRAM-only designs such as the Groq TSP (Abts et al., 2020) and the Cerebras WSE (Lie, 2023) have no off-chip boundary and are outside its scope.

Snowcat's costs are given by a small input energy table: the energy of one MAC at each operand bit-width, and the energy per byte moved across the MainMemory ↔ GlobalBuffer boundary. *E1*

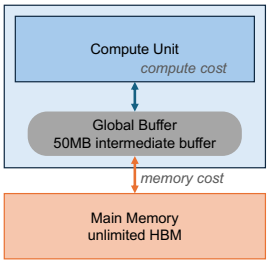


Figure 2: The hardware architecture assumed by *E1*: unbounded off-chip memory, a finite on-chip buffer, and a compute unit.

Table 1: Energy table used by *E1*, calibrated on an NVIDIA H100. Memory entries are pJ per byte moved between MainMemory and GlobalBuffer; compute entries are pJ per MAC.

Action	Energy
MainMemory read	70.6 pJ/byte
MainMemory write	63.9 pJ/byte
8-bit MAC	0.57 pJ/op
16-bit MAC	0.89 pJ/op
32-bit MAC	1.96 pJ/op
64-bit MAC	11.59 pJ/op

multiplies the operation and byte counts of a schedule by these entries and sums. The table is an interchangeable input: the user provides it to follow the profile of a particular target accelerator or an arbitrary design point, such as a chosen ratio of memory to compute cost, and *E1* scores the model against it (§4).

Calibration. For *E1* to report Joules that can be checked against a real device, we populated the table by measuring an NVIDIA H100, the GPU used for every measurement in this paper, and set the GlobalBuffer capacity to its 50 MB L2 cache (Appendix B); Table 1 lists the resulting entries, used throughout this paper. Because the calibration kernels keep the hardware fully busy, the entries describe an efficient implementation rather than idle or leakage energy.

Operations that are not MACs. Element-wise and reduction operations, such as the exponentials in softmax, are counted as MACs at the einsum's operand precision; in most models they are a small, memory-bound fraction of the arithmetic. Operations that move data without arithmetic, such as embedding lookups or the routing of token rows to experts and the combination of their outputs in a mixture-of-experts layer, are accounted for by the bytes they move at the table's memory costs, with data-dependent indices taken from one execution of the model (§3.4, Appendix D); control such as top-k selection moves no tensor and is not counted. Recurrent state is an explicit input and output tensor of the graph, so a decode step maps like any other kernel (Appendix G).

2.3 Energy Model and Mapspace Search

The total energy of an einsum graph, whether a single kernel or a whole model forward pass, is the sum of a compute term and a memory term:

$$E_{total} = E_{compute} + E_{memory}. \tag{1}$$

The compute term sums the arithmetic operation count of each einsum, weighted by the per-operation cost at the einsum's input bit-width:

$$E_{compute} = \sum_{einsum\ e} N_e^{\text{op}} \cdot \epsilon_{\text{bits}(e)}^{\text{op}}, \tag{2}$$

where N_e^{op} is the number of operations in einsum e and $\epsilon_{\text{bits}(e)}^{\text{op}}$ is the per-operation cost at that bit-width from Table 1. The memory term sums the bytes that cross the on/off-chip boundary, separately for reads and writes:

$$E_{memory} = B^R \cdot \epsilon^R + B^W \cdot \epsilon^W, \tag{3}$$

where B^R and B^W are the number of bytes that cross the MainMemory $\leftrightarrow$ GlobalBuffer boundary in each direction over the entire invocation, and ϵ^R and ϵ^W are the per-byte read and write costs.

E1's central job is therefore to find the schedule that minimizes (3) subject to the GlobalBuffer capacity, and to report the resulting total (1).

E1 uses AccelForge (Andrulis et al., 2026a;b), which searches the tiling and fusion mapspace exactly, provably pruning suboptimal mappings. Adjacent einsums that share rank labels may be *joined*: their shared loops are fused so that the intermediate tensor flowing from producer to consumer stays in the

GlobalBuffer rather than being written back to MainMemory. AccelForge propagates shared ranks through the dataflow graph and searches a single mapspace over tiling, loop order, and legal fusion for the whole graph, so chains of joins form fused regions that span many operators. Appendix F shows what the search buys on nine kernels: counting every operator's traffic once costs 1.1–$10.5\times$ the energy of the schedule the mapper finds.

E1 maps the einsum graph of a kernel as a whole, so that an intermediate is only counted as travelling to MainMemory when the schedule requires it; a full model is cut into its repeated blocks at tensors the framework materializes anyway, which adds about 2% to the total (Appendix D).

2.4 Scope and Design Choices

Three decisions keep *E1* simple, fast to compute, and independent of any particular accelerator.

Inference only. *E1* scores a single forward pass at a user-specified input shape: inference is expected to dominate long-term AI energy consumption (Jegham et al., 2025), and forward-pass cost is the common unit across deployed workloads.

Single device. *E1* scores a single device with one main memory system; distributed inference adds the communication energy of a parallelism strategy on top (§4).

First-order, model-level energy. *E1* scores the model, not a deployment: it counts the dynamic energy of arithmetic and off-chip data movement, the two costs the model itself determines. Static and leakage power, thermal throttling and frequency scaling, on-chip cache traffic, and software overheads such as kernel launches belong to a deployment, since launch overhead disappears with fusion or CUDA Graphs and throttling depends on cooling and run length, and are not part of the score.

3 *E1* Validation and Results

We validate *E1* against energy measured on an NVIDIA H100 GPU, the accelerator the energy table was calibrated on. For each workload we run the implementation repeatedly and read the GPU's energy counter over the run, and report the mean energy per invocation (Appendix B). Measurements are total board energy and therefore sit above *E1*; where it helps, we also report them with idle power subtracted.

3.1 Microbenchmarks

E1 receives only a minimal, unoptimized PyTorch expression of each operation and shape; since it estimates the energy of optimized code, the measured side is a high-performance implementation. We use cuBLAS and cuDNN for the GEMM-like kernels and convolutions and hand-written CUDA kernels for the rest (Appendix B), each over a grid of representative shapes.

Figure 3 shows that *E1* tracks the measured energy over three orders of magnitude. For the large GEMM, linear, and convolution shapes the measurement approaches *E1*, because the optimized cuBLAS and cuDNN kernels saturate the relevant hardware paths. For smaller kernels the gap grows, since fixed costs that *E1* leaves out, kernel launch, imperfect utilization and the board's idle power, become visible at this scale. Read per arithmetic operation, the same measurements show why FLOP counts fail as an energy proxy: the $16\text{K}\times16\text{K}$ GEMM costs $0.75\,\text{pJ}$ per multiply-accumulate, while the element-wise addition over 1G elements costs $1.3\,\text{nJ}$ per element, $1700\times$ more, because every element travels to and from memory for a single operation.

Appendix C compares, for the normalization kernels, PyTorch eager against the best available implementation and against *E1*: eager code costs 1.7–$8\times$ more than the best implementation, which itself sits 3–$6\times$ above *E1* at 128 rows. That distance between eager and optimized code is the headroom *E1* is meant to expose before the optimized kernel exists.

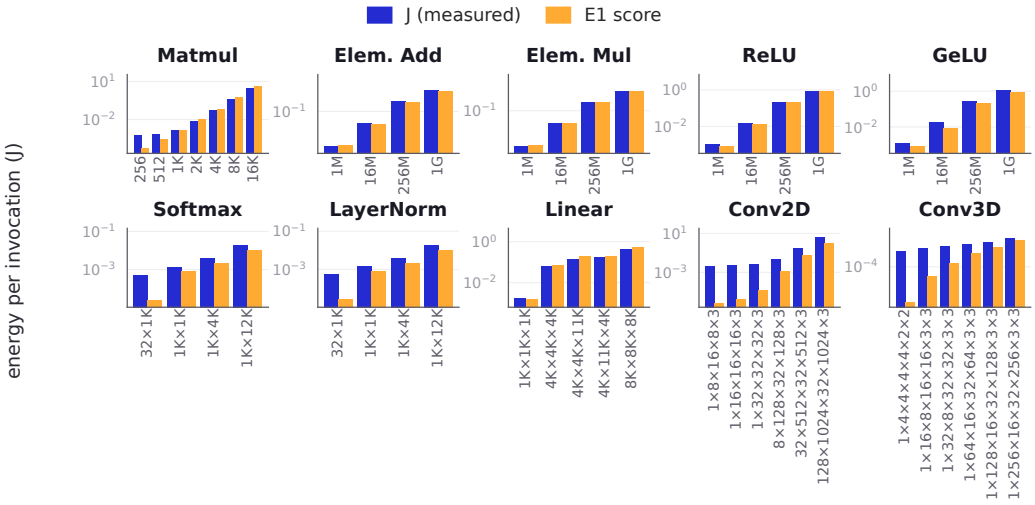


Figure 3: Energy per invocation of ten kernel families on an NVIDIA H100 in Joules: an optimized implementation (blue, idle power subtracted) next to the *E1* score (orange). Shape labels: K for a $K \times K \times K$ matmul; $M \times N \times K$ for linear layers; rows×columns for softmax and LayerNorm; batch×channels×spatial size×filters×kernel for convolutions, with depth and a temporal kernel added for Conv3D.

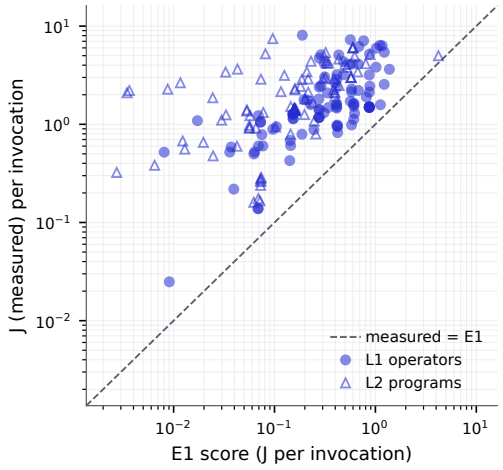


Figure 4: KernelBench L1 operators (circles) and L2 programs (triangles) on an NVIDIA H100: the *E1* score (horizontal axis) against the measured energy per invocation of the PyTorch reference (vertical axis), both in Joules on logarithmic axes; the dashed line marks equality. Every task lies above the line.

3.2 KernelBench

KernelBench (Ouyang et al., 2025) provides PyTorch references for individual operators (L1) and short fused programs (L2). We score the operators and programs of both levels and measure each reference as written, in eager PyTorch (Figure 4).

All references lie above the line, at a median of $5.4\times$ *E1* for the single operators and $9.3\times$ for the fused programs. The element-wise and pointwise operators come closest (1.4–$1.7\times$); matrix multiplications sit 3–$10\times$ above *E1* because the references run their arithmetic on the CUDA cores while *E1* counts MACs at the tensor-core rate an optimized implementation would use; and the L2

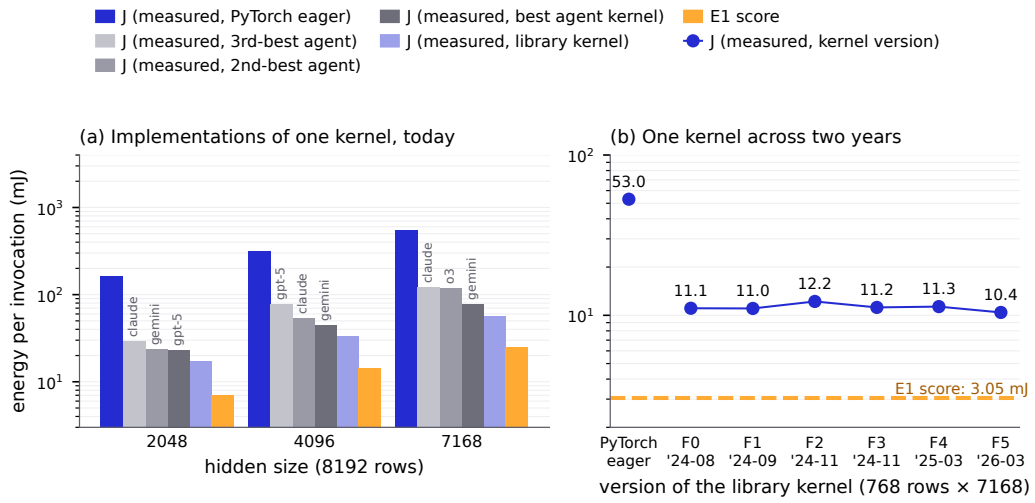


Figure 5: FlashInfer's fused add+RMSNorm kernel (BF16) against the *E1* score on an NVIDIA H100, energy per invocation in mJ on logarithmic axes. (a) At 8192 rows and three hidden sizes: PyTorch eager (dark blue), the FlashInfer 0.6.10 library kernel (light blue), the three lowest-energy kernels written by automated agents for FlashInfer's public benchmark (greys, labelled with the agent's model), and the *E1* score (orange). (b) At 768×7168: the measured energy of the library kernel at six points of its history, from its introduction (F0, August 2024) to the current CuTe-DSL kernel (F5, March 2026), with PyTorch eager for reference, against the one *E1* score for the operation (dashed orange), which does not change because none of the versions changes the math.

programs, which eager PyTorch executes as one kernel per operator with every intermediate written to memory, reach the largest gaps.

3.3 *E1* AS AN OPTIMIZATION TARGET

Because *E1* depends on the math and not on the implementation, it gives kernel engineering a target that does not move. We look at this from two directions: several implementations of the same kernel at one point in time, and one implementation across time.

Several implementations of the same kernel. FlashInfer (Ye et al., 2025) is a production library of attention and normalization kernels for LLM serving, and its public benchmark asks automated agents to write kernels for the same operations. Figure 5a compares, for fused add+RMSNorm at 8192 rows and three hidden sizes, PyTorch eager, the FlashInfer library kernel and the three lowest-energy agent-written kernels against *E1*. The ranking is the same at every size: eager PyTorch costs $22\times$ *E1*, the best agent kernels 3.1–$3.3\times$, and the library kernel 2.3–$2.4\times$. The three best agent kernels for one size differ by up to $1.8\times$ among themselves, and the best of them stays 1.3–$1.4\times$ above the library kernel it competes with. A leaderboard tells an engineer which implementation is currently best, but not how far any of them is from what the math requires; *E1* does: the library has already taken most of the $22\times$ between eager code and *E1*, and what the agents are competing over is the last factor of three.

Kernel history trajectory. We then follow a single production kernel through its own history across 2 years. FlashInfer's fused add+RMSNorm kernel was introduced in August 2024 and changed in five kernel-level pull requests before the current CuTe-DSL implementation (Appendix E). We build each version unmodified and measure all of them, together with PyTorch eager, on the same GPU at 768×7168 BF16, against the one *E1* value for the operation, 3.05 mJ.

Figure 5b shows that the first version already captured most of the gain over eager PyTorch ($4.8\times$); the five later changes moved the energy by between -6% and $+10\%$, and the one motivated by memory traffic, dropping a redundant reload of the input, was invisible (-0.2%) because the re-reads were already served from cache. Throughout, *E1* did not move, and every version remained 3.4–$4\times$

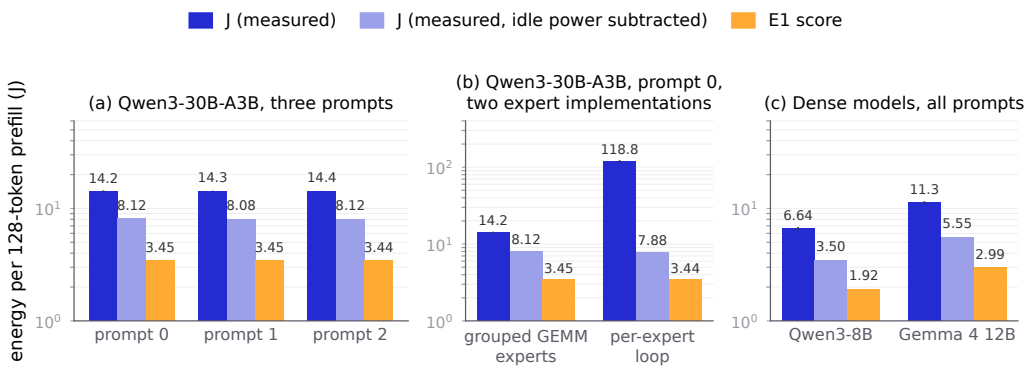


Figure 6: Energy of 128-token prefills of pretrained models on an NVIDIA H100, in Joules on a logarithmic axis: measured board energy (dark blue), the same with idle power subtracted (light blue), and the *E1* score (orange). (a) Qwen3-30B-A3B, three prompts under Transformers' default expert implementation. (b) Qwen3-30B-A3B, one prompt under the default grouped-GEMM experts and under Transformers' per-expert loop. (c) The dense Qwen3-8B and Gemma 4 12B, all prompts pooled.

above it: what these kernels still pay is not data movement, which *E1* already accounts for at its minimum, but fixed per-launch and utilization cost that shrinks only with more work per launch (2.3× at 8192 rows, Figure 5a). This is the second thing a fixed target tells an engineer: not just how far the implementation is from *E1*, but that the remaining distance is not where the last two years of changes were looking.

3.4 End-to-end Models

E1 also scores full model forward passes. We show a mixture-of-experts model and two dense models, all pretrained checkpoints run through the standard Transformers implementation at batch 1 with a 128-token prefill, so that the measured side is what a researcher would actually run.

Mixture-of-experts models. Qwen3-30B-A3B routes each token of each of its 48 layers to 8 of 128 experts. *E1* maps each expert once per number of tokens it receives, with the routes the model takes on the prompt, and the rest of the layer once (Appendix D); Figure 6 shows three prompts. First, at batch 1 an expert's weights dominate its traffic, so *E1* for this model is set by how many experts a prompt touches rather than by how many tokens each expert processes. *E1* is 3.45 J for all three prompts, against 14.2–14.4 J measured (8.1 J after subtracting idle power). Second, the same model and prompt executed with Transformers' per-expert loop instead of its grouped GEMM costs 8× the board energy at an identical *E1*; the idle-subtracted energies of the two agree within 3%, so the entire difference is idle power during a launch-bound loop. This is the situation *E1* is designed for: an architecture judged by the measured energy of its first implementation would look 8× worse than the same architecture a few months later.

Dense models. Qwen3-8B (36 layers, hidden size 4096) and Gemma 4 12B (48 layers, hidden size 3840) are scored the same way, in BF16 (Figure 6c); Gemma 4 has a 262K-entry vocabulary with a tied output head and alternates local and global attention layers. *E1* is 1.92 J for Qwen3-8B and 2.99 J for Gemma 4, and it has the same shape for both: the feed-forward blocks are 73% of the total, the attention cores about 1%, and the single read of the output head costs 88 mJ for Qwen3-8B (1.2 GB) and 143 mJ for Gemma 4 (2.0 GB). Measured board energy is 6.6 J and 11.3 J, 3.5–3.8× *E1*; with idle power subtracted, 3.5 J and 5.6 J, 1.8–1.9× *E1*.

Recurrent attention models. *E1* is not limited to Transformer-style graphs: Appendix G scores one decode step of a Mamba block, whose recurrent state enters and leaves the graph as an explicit tensor, at block widths from 256 to 2560.

4 Discussion and Future Work

Related work on measuring the energy of AI systems, hardware-independent cost proxies, analytical and learned cost models, accelerator mappers, and compilers and kernel generation is discussed in Appendix A.

***E1* in AI research.** For the workloads in this paper, *E1* is fast enough to run during model development: a single kernel maps in 15–70 s on a CPU, the segments of a dense decoder block in 20–100 s each, and the mixture-of-experts model of §3.4 in about 20 minutes on 24 cores. Appendix D describes the segmentation that keeps these times small and what it costs in accuracy. Its most direct use is within architecture studies: reported alongside accuracy under fixed input shapes, *E1* can steer modeling decisions and identify architectures whose energy headroom makes further systems optimization worthwhile. *E1* could also support cross-paper leaderboards: an estimated energy column, or an intelligence-per-Joule score in the spirit of Intelligence per Watt (Saad-Falcon et al., 2025), next to accuracy, would expose the headroom of new architectures rather than the maturity of their current kernels. Cross-study comparisons would require standard inputs and reporting rules, as in MLPerf Inference (Reddi et al., 2020).

Understanding gap between *E1* and empirical measurements. For kernels and models that keep the GPU busy, optimized implementations land within a small factor of *E1*; for short kernels and small batches, most of the measured energy is the GPU's idle power over the invocation and the host's time between launches, neither of which depends on the model. These costs belong to a deployment rather than to a model: the $8\times$ of §3.4 is entirely idle power, and a measurement at a larger batch or after kernel fusion moves toward *E1* without *E1* changing.

Hardware abstraction. *E1*'s hardware abstraction is simple because its goal is to give an algorithm an energy identity, not to quantify how well it maps to a specific device, which is what detailed accelerator simulators do. The energy table is the one place hardware enters, and because *E1* is linear in its entries, users can substitute another accelerator's constants or scale the memory and compute costs separately to see how the balance between memory-bound and compute-bound operators shifts.

Future work. Algebraic reformulations such as FlashAttention change the graph before scheduling and need a separate rewrite layer, which would let *E1* evaluate a larger space of implementation strategies without changing the energy model. Quantized inference is representable today through precision-specific costs and smaller tensors, and we plan to evaluate it. Finally, this work doesn't explore distributed deployments. Since energy is additive, extending *E1* to distributed inference would entail adding the communication energy of a parallelism strategy over a given network fabric.

5 Conclusion

Architecture decisions are made long before an optimized implementation can be measured; *E1* fills that gap. We introduced an infrastructure-agnostic energy metric that estimates the optimized energy potential of unoptimized model code by combining arithmetic work, memory movement, and a search over tiling and fusion schedules on an abstract hardware target. Across microbenchmarks, KernelBench, FlashInfer kernels and their history, and full model forward passes, *E1* behaves as intended: it tracks the energy of optimized implementations over three orders of magnitude, it does not move when the implementation changes, and the distance to it shows where an implementation still pays for something other than the math.

Our broader goal is to make energy visible at the same stage as accuracy. *E1* can help researchers compare not only which models are capable, but also which models have the potential to become efficient deployments. Future versions should broaden operator coverage, incorporate algebraic transformations, and extend the abstraction to distributed inference. The central direction, however, is simple: energy should become a first-class signal in model architecture exploration.

AI USE STATEMENT

We did not use generative AI tools to generate synthetic data sets, to help develop theoretical models or conceptual frameworks, to formulate mathematical claims, to provide critical ingredients for proving mathematical claims, to assist in the writing of proofs, to propose or refine hypotheses, to design or provide feedback on research methodology or experiments, to implement methods, to assist with translation, to clean and reformat data sets, to support qualitative and thematic data analysis, or to interpret results.

We did use generative AI tools for the following tasks: creating figures from our measured and computed data, editing software code, drafting early versions of parts of the paper text, analysing existing literature, and formatting the paper.

We have reviewed all AI-assisted work. Every figure was checked against the underlying data files by the authors; every code edit was read and tested by the authors before use; every drafted passage was rewritten or verified by the authors against our method and results; and every reference surfaced by literature analysis was read by the authors and cited from its published version. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 2 and 3 describe the E1 metric, abstract hardware target, energy model, mapspace search, and experimental methodology. Appendices B–G provide the H100 calibration and measurement protocol, full-model mapping procedure, workload configurations, kernel-version provenance, traffic-bound comparisons, and recurrent-model experiment. We will open-source the E1 and artifacts needed to reproduce the results in this paper.

REFERENCES

Jon Saad-Falcon, Avanika Narayan, Hakki Orhun Akengin, J. Wes Griffin, Herumb Shandilya, Adrian Gamarra Lafuente, Medhya Goel, Rebecca Joseph, Shlok Natarajan, Etash Kumar Guha, Shang Zhu, Ben Athiwaratkun, John Hennessy, Azalia Mirhoseini, and Christopher Ré. Intelligence per Watt: Measuring Intelligence Efficiency of Local AI. *arXiv preprint arXiv:2511.07885*, 2025.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *Conference on Language Modeling (COLM)*, 2024.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. *International Conference on Machine Learning (ICML)*, 2024.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple Linear Attention Language Models Balance the Recall-Throughput Tradeoff. *International Conference on Machine Learning (ICML)*, 2024.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry Hungry Hippos: Towards Language Modeling with State Space Models. *International Conference on Learning Representations (ICLR)*, 2023.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena Hierarchy: Towards Larger Convolutional Language Models. *International Conference on Machine Learning (ICML)*, 2023.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. RWKV: Reinventing RNNs for the Transformer Era. *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. *International Conference on Learning Representations (ICLR)*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30, 2017.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems*, 35, 2022.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. *International Conference on Machine Learning (ICML)*, 2023.

Qwen Team. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

NVIDIA. Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models. *arXiv preprint arXiv:2504.03624*, 2025.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon Emissions and Large Neural Network Training. *arXiv preprint arXiv:2104.10350*, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Tri Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *International Conference on Learning Representations (ICLR)*, 2024.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-Precision. *Advances in Neural Information Processing Systems*, 37, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*, 2020.

Qingqing Cao, Yash Kumar Lal, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. IrEne: Interpretable Energy Prediction for Transformers. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Zeyu Yang, Karel Adamek, and Wesley Armour. Part-time Power Measurements: nvidia-smi's Lack of Attention. *arXiv preprint arXiv:2312.02741*, 2023.

Chenxu Niu, Wei Zhang, Jie Li, Yongjian Zhao, Tongyang Wang, Xi Wang, and Yong Chen. TokenPowerBench: Benchmarking the Power Consumption of LLM Inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Nidhal Jegham, Marwan Abdelatti, Chan Young Koh, Lassad Elmoubarki, and Abdeltawab Hendawi. How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference. *arXiv preprint arXiv:2505.09598*, 2025.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, and others. MLPerf Inference Benchmark. *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Sunpyo Hong and Hyesoon Kim. An Integrated GPU Power and Performance Model. *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, 2010.

Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. CoSA: Scheduling by Constrained Optimization for Spatial Accelerators. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2021.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Fredo Durand, and Saman Amarasinghe. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2013.

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating High-Performance Tensor Programs for Deep Learning. *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

Qijing Huang, Po-An Tsai, Joel S. Emer, and Angshuman Parashar. Mind the Gap: Attainable Data Movement and Operational Intensity Bounds for Tensor Algorithms. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024.

Tanner Andrulis, Michael Gilbert, Vivienne Sze, and Joel S. Emer. Fast and Fusiest: An Optimal Fusion-Aware Mapper for Accelerator Design. *arXiv preprint arXiv:2602.15166*, 2026.

Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. Data Movement Is All You Need: A Case Study on Optimizing Transformers. *MLSys*, 2021.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, and others. In-Datacenter Performance Analysis of a Tensor Processing Unit. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

Murali Emani, Venkatram Vishwanath, Corey Adams, Michael E. Papka, Rick Stevens, Laura Florescu, Sumti Jairath, William Liu, Tejas Nama, and Arvind Sujeeth. Accelerating Scientific Applications With SambaNova Reconfigurable Dataflow Architecture. *Computing in Science & Engineering*, 2021.

Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, Temesghen Kahsai, Garrin Kimmell, and others. Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2020.

Sean Lie. Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning. *IEEE Micro*, 43(3), 2023.

NVlabs. SOLAR: Speed of Light Analysis for Runtime. https://github.com/NVlabs/SOLAR, 2025.

Mert Kurttutan. torchview: Visualize PyTorch Models. https://github.com/mert-kurttutan/torchview, 2024.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently Scaling Transformer Inference. *MLSys*, 2023.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs Write Efficient GPU Kernels? *International Conference on Machine Learning (ICML)*, 2025.

Tao Lin. Measuring Automated Kernel Engineering. https://metr.org/blog/2025-02-14-measuring-automated-kernel-engineering/, 2025.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and Customizable Attention Engine for LLM Inference Serving. *Proceedings of Machine Learning and Systems*, 2025.

Tyler Griggs, Xiaoxuan Liu, Jiaxiang Yu, Doyoung Kim, Wei-Lin Chiang, Alvin Cheung, and Ion Stoica. Melange: Cost Efficient Large Language Model Serving by Exploiting GPU Heterogeneity. *arXiv preprint arXiv:2404.14527*, 2024.

Epoch AI. Build times for gigawatt-scale data centers. https://epoch.ai/data-insights/data-centers-buildout-speeds, 2025.

NVIDIA. NVIDIA Blackwell: Top Energy-Efficient Inference Accelerator. https://perspectives.nvidia.com/energy-efficient-accelerator-inference-electricity-tco, 2025.

Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing Carbon: The Elusive Environmental Footprint of Computing. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

Sara Hooker. The Hardware Lottery. *Communications of the ACM*, 64(12):58–65, 2021.

Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The Efficiency Misnomer. *International Conference on Learning Representations (ICLR)*, 2022.

Mark Horowitz. Computing's Energy Problem (and What We Can Do About It). *IEEE International Solid-State Circuits Conference (ISSCC)*, 2014.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the Carbon Emissions of Machine Learning. *arXiv preprint arXiv:1910.09700*, 2019.

Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model. *Journal of Machine Learning Research*, 24(253):1–15, 2023.

Sasha Luccioni, Yacine Jernite, and Emma Strubell. Power Hungry Processing: Watts Driving the Cost of AI Deployment? *ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, 2024.

Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. Sustainable AI: Environmental Implications, Challenges and Opportunities. *Proceedings of Machine Learning and Systems (MLSys)*, 2022.

Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference. *IEEE High Performance Extreme Computing Conference (HPEC)*, 2023.

Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in AI Inference Energy Consumption: Beyond the Performance-vs-Parameter Laws of Deep Learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023.

Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, Jeremy Holleman, Csaba Kiraly, Pawan Ambalkar, Ritika Borkar, Ramesh Chukka, Trevor Cockrell, et al. MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Systems from Microwatts to Megawatts for Sustainable AI. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2025.

Jae-Won Chung, Jeff J. Ma, Ruofan Wu, Jiachen Liu, Oh Jun Kweon, Yuxuan Xia, Zhiyu Wu, and Mosharaf Chowdhury. The ML.ENERGY Benchmark: Toward Automated Inference Energy Measurement and Optimization. *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Jae-Won Chung, Yile Gu, Insu Jang, Luoxi Meng, Nikhil Bansal, and Mosharaf Chowdhury. Perseus: Removing Energy Bloat from Large Model Training. *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2024.

Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Iñigo Goiri, and Josep Torrellas. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. *arXiv preprint arXiv:2403.20306*, 2024.

Jovan Stojkovic, Chaojie Zhang, Iñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2025.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Iñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training Compute-Optimal Large Language Models. *Advances in Neural Information Processing Systems*, 35, 2022.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. LLM Inference Unveiled: Survey and Roofline Model Insights. *arXiv preprint arXiv:2402.16363*, 2024.

Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A Performance Model for Deep Neural Networks. *International Conference on Learning Representations (ICLR)*, 2017.

Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. NeuralPower: Predict and Deploy Energy-Efficient CNN Architectures. *Asian Conference on Machine Learning (ACML)*, 2017.

Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro*, 40(3):20–29, 2020.

Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators. *IEEE Transactions on Computers*, 70(8):1160–1174, 2021.

Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

Mikhail Isaev, Nic McDonald, Larry Dennison, and Richard Vuduc. Calculon: A Methodology and Tool for High-Level Co-Design of Systems and Large Language Models. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2023.

Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. LLMCompass: Enabling Efficient Hardware Design for Large Language Model Inference. *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024.

Abhimanyu Bambhaniya, Ritik Raj, Geonhwa Jeong, Souvik Kundu, Sudarshan Srinivasan, Suvinay Subramanian, Midhilesh Elavazhagan, Madhu Kumar, and Tushar Krishna. Demystifying AI Platform Design for Distributed Inference of Next-Generation LLM Models. *arXiv preprint arXiv:2406.01698*, 2024.

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A Large-Scale Simulation Framework for LLM Inference. *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. *ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, 2019.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

Michael Gilbert, Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. LoopTree: Exploring the Fused-Layer Dataflow Accelerator Design Space. *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.

Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W. Fletcher. Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search. *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

Charles Hong, Qijing Huang, Grace Dinh, Mahesh Subedar, and Yakun Sophia Shao. DOSA: Differentiable Model-Based One-Loop Search for DNN Accelerators. *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

Yannan Nellie Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. Sparseloop: An Analytical Approach to Sparse Tensor Accelerator Modeling. *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

Nandeeka Nayak, Toluwanimi O. Odemuyiwa, Shubham Ugare, Christopher W. Fletcher, Michael Pellauer, and Joel S. Emer. TeAAL: A Declarative Framework for Modeling Sparse Tensor Accelerators. *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

Tanner Andrulis, Michael Gilbert, Vivienne Sze, and Joel S. Emer. AccelForge: Comprehensive Modeling and Co-Design Framework for AI Accelerators. *arXiv preprint arXiv:2609.11906*, 2026.

Michael Gilbert, Tanner Andrulis, Vivienne Sze, and Joel S. Emer. The Turbo-Charged Mapper: Fast and Optimal Mapping for Energy-Efficient and Low-Latency Accelerator Design. *arXiv preprint arXiv:2602.15172*, 2026.

Barna Saha and Christopher Ye. I/O Complexity of Attention, or How Optimal is FlashAttention? *International Conference on Machine Learning (ICML)*, 2024.

Jia-Wei Hong and H. T. Kung. I/O Complexity: The Red-Blue Pebble Game. *ACM Symposium on Theory of Computing (STOC)*, pp. 326–333, 1981.

Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. A Roofline Model of Energy. *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2013.

Ahmad Faiz, Sotaro Kaneda, Ruhan Wang, Rita Osi, Prateek Sharma, Fan Chen, and Lei Jiang. LLMCarbon: Modeling the End-to-End Carbon Footprint of Large Language Models. *International Conference on Learning Representations (ICLR)*, 2024.

Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. Energy Considerations of Large Language Model Inference and Efficiency Optimizations. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.

Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws. *International Conference on Machine Learning (ICML)*, 2024.

Song Bian, Minghao Yan, and Shivaram Venkataraman. Scaling Inference-Efficient Language Models. *International Conference on Machine Learning (ICML)*, 2025.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Seonho Lee, Amar Phanishayee, and Divya Mahajan. Forecasting GPU Performance for Deep Learning Training and Inference. *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G. Rogers, Tor M. Aamodt, and Nikos Hardavellas. AccelWattch: A Power Modeling Framework for Modern GPUs. *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

Arne Symons, Linyan Mei, Steven Colleman, Pouya Houshmand, Sebastian Karl, and Marian Verhelst. Stream: Design Space Exploration of Layer-Fused DNNs on Heterogeneous Dataflow Accelerators. *IEEE Transactions on Computers*, 74(1):237–249, 2025.

Robin Geens, Jonas De Schouwer, Marian Verhelst, and Thierry Tambe. The Hyperscale Lottery: How State-Space Models Have Sacrificed Edge Efficiency. *arXiv preprint arXiv:2604.07935*, 2026.

A Related Work

Measuring the energy of AI. A large body of work measures the energy and carbon footprint of training and inference on deployed systems, from early accounting of NLP training runs (Strubell et al., 2019; Patterson et al., 2021; Henderson et al., 2020; Lacoste et al., 2019; Luccioni et al., 2023; Wu et al., 2022a; Gupta et al., 2021) to recent studies of LLM inference (Luccioni et al., 2024; Samsi et al., 2023; Desislavov et al., 2023; Jegham et al., 2025; Niu et al., 2026; Saad-Falcon et al., 2025). Benchmarks such as MLPerf Power and the ML.ENERGY Benchmark standardize how such measurements are collected and reported (Tschand et al., 2025; Chung et al., 2025), and Yang et al. (2023) document the pitfalls of sampling GPU power counters. A related line optimizes the energy of a given deployment at runtime through frequency scaling, batching, and phase-aware scheduling (You et al., 2023; Chung et al., 2024; Stojkovic et al., 2024; 2025; Patel et al., 2024). All

of these operate after an implementation exists, and their numbers mix the architecture with kernel maturity, framework overhead, and hardware choice: Fernandez et al. (2025) show that FLOP-based estimates underestimate measured inference energy and that serving choices alone change it by tens of percent. New architectures without optimized kernels therefore look inefficient for systems reasons rather than algorithmic ones, a bias Hooker (2021) calls the hardware lottery and Geens et al. (2026) document for state-space models tuned to datacenter GPUs. *E1* targets the stage before implementation, where these measurements are not yet available.

Hardware-independent proxies. Scaling laws and Green AI use FLOPs or parameter counts as hardware-independent cost proxies (Kaplan et al., 2020; Hoffmann et al., 2022; Schwartz et al., 2020); inference-aware scaling laws fold deployment cost into the same objective (Sardana et al., 2024; Bian et al., 2025), and hardware-aware architecture search does so with latency measured on one device (Wang et al., 2020). These proxies are easy to compute from a model definition, but Dehghani et al. (2022) show that FLOPs, parameters, and measured throughput frequently disagree about which model is cheaper. The reason is data movement: energy per byte moved off-chip exceeds energy per arithmetic operation by orders of magnitude (Horowitz, 2014; Sze et al., 2017), and roofline-style analyses show that memory traffic, not arithmetic, bounds many operators (Williams et al., 2009; Ivanov et al., 2021; Yuan et al., 2024). Choi et al. (2013) extend the roofline to energy with per-operation and per-byte costs, the two-term decomposition *E1* also uses. *E1* keeps the early-stage, hardware-agnostic advantage of counting from the model definition while modeling memory traffic explicitly, and it does so under an optimized schedule rather than for the naive one.

Analytical and learned cost models. Analytical performance and power models predict the cost of a network on a given platform from its structure (Qi et al., 2017; Hong and Kim, 2010; Kandiah et al., 2021), learned predictors such as NeuralPower, IrEne, and NeuSight fit per-platform models to measurements (Cai et al., 2017; Cao et al., 2021; Lee et al., 2025), and LLMCarbon projects training and inference footprints from parameter counts and assumed hardware utilization (Faiz et al., 2024). For LLM systems, Calculon, LLMCompass, GenZ, and Vidur model latency and throughput of inference or training on specified hardware (Isaev et al., 2023; Zhang et al., 2024; Bambhaniya et al., 2024; Agrawal et al., 2024). All of these take the target hardware as input and predict the implementation they model or were fitted to; *E1* fixes a minimal three-component abstraction instead and scores the model.

Mappers and accelerator modeling. The energy of a tensor program on an accelerator is decided by its mapping, i.e., how loops are tiled, ordered, and fused across the memory hierarchy (Chen et al., 2016; Yang et al., 2017). Timeloop and Accelergy made this explicit by pairing an analytical model of data movement with a mapspace search over a detailed accelerator specification (Parashar et al., 2019; Wu et al., 2019), and MAESTRO, ZigZag, Interstellar, and Stream explore the same space with different cost models and hardware targets (Kwon et al., 2020; Mei et al., 2021; Yang et al., 2020; Symons et al., 2025). Because the mapspace is enormous, much of the work is in the search: CoSA casts it as constrained optimization (Huang et al., 2021), Mind Mappings and DOSA make it differentiable (Hegde et al., 2021; Hong et al., 2023), and Sparseloop and TeAAL extend the modeling to sparse tensors and einsum cascades (Wu et al., 2022b; Nayak et al., 2023). Orojenesis bounds the data movement any mapping can attain (Huang et al., 2024), connecting the search to I/O-complexity lower bounds from the red-blue pebble game to attention (Hong and Kung, 1981; Saha and Ye, 2024), and LoopTree, Fast and Fusiest, and the Turbo-Charged Mapper add fusion to the mapspace and search it optimally in seconds (Gilbert et al., 2024; Andrulis et al., 2026b; Gilbert et al., 2026), now packaged as AccelForge (Andrulis et al., 2026a). These tools are built to evaluate hardware: they take a detailed architecture as input and optimize a mapping for it. *E1* borrows their machinery, using AccelForge as its fusion-aware mapper, but inverts the question: it fixes one abstract accelerator with a single energy table and uses the optimal mapping to characterize the model.

Compilers, autotuners, and kernel generation. Compilers and autoschedulers such as Halide, TVM, Ansor, Triton, and TorchInductor search tiling, fusion, and loop-order spaces to produce fast code for a specific device (Ragan-Kelley et al., 2013; Chen et al., 2018; Zheng et al., 2020; Tillet et al., 2019; Ansel et al., 2024), and hand-written kernels such as FlashAttention and FlashInfer show how much optimized implementations can gain (Dao et al., 2022; Dao, 2024; Shah et al., 2024; Ye et al., 2025). KernelBench and studies of LLM-driven kernel engineering evaluate how well

automated agents can reproduce this work (Ouyang et al., 2025; Lin, 2025). These systems produce implementations and need executable code and target hardware to do so. *E1* estimates what they could reach before they exist, and gives them an energy target to approach.

B Measurement Protocol

All measurements in this paper were taken on an NVIDIA H100 NVL (400 W power limit), the GPU on which the energy table of Table 1 was calibrated. Each measured number is the mean over three independent runs, each in its own process; each run executes the workload repeatedly over a fixed window on a rotating pool of independent inputs and outputs larger than twice the L2 cache, so that consecutive invocations do not hit cached data. Energy is the difference of the GPU's cumulative energy counter over the window, cross-checked against sampled board power integrated over the same window. Kernels are replayed as CUDA graphs; full models run through the standard Transformers implementation. The optimized implementations of Figure 3 are the library paths frameworks dispatch to (cuBLAS for `matmul`, cuBLAS plus a bias addition for `linear`, cuDNN with its default algorithm heuristic for convolutions) and, for element-wise and reduction kernels, hand-written CUDA kernels with vectorized memory access, warp-shuffle reductions, and shared-memory tiling. Idle power is measured before and after each run, and the idle-subtracted energy reported in some figures is the board energy minus idle power times the invocation time. Every measured implementation is checked against an FP64 reference of the same operation before it is timed. Figure 3 reports idle-subtracted energy; the implementation comparison of Figure 5a uses one run per implementation; the dense models of Figure 6c pool all prompts and runs.

Calibration of the energy table. We run kernels that saturate either the tensor cores or the memory system and read the board's power through NVML. For the tensor-core paths at each precision, a saturating cuBLAS GEMM runs and an idle baseline measured alongside is subtracted. For scalar arithmetic and for DRAM reads and writes, CUDA probes are swept over problem sizes and energy is fit linearly against operation count; the slope is the per-operation or per-byte cost, and the intercept absorbs fixed costs that scale with time, such as static power. The entries are board-level: they include everything the GPU spends per operation or per byte, not only the arithmetic units.

C Eager, Optimized, and *E1* for Individual Kernels

Figure 7 complements Figure 3 with three bars per kernel: the PyTorch eager expression as a researcher would first write it, the best implementation available for the same operation, and *E1*.

D Method Details

Non-MAC operations. *E1* counts the arithmetic of every einsum in MAC-equivalent operations and attributes to each the MAC cost of the einsum's operand precision. Exponentials, divisions and comparisons are therefore counted as MACs. Per-operator compute costs are a direct extension of the energy table.

Mapping full models. For a full model, *E1* traces each block type once with Solar and maps it once with AccelForge, cutting the graph only at tensors that the eager implementation materializes anyway: the projection outputs, the attended tensor, the normalized residual, and, for mixture-of-experts layers, the rows gathered for each expert and the expert outputs. The traffic of every cut is accounted for. For the models in §3.4 these cuts add about 2% to the estimate. The cuts are also what keeps the search time small: the mapper's cost grows quickly with the number of einsums in one graph, so a single kernel of 2–12 einsums maps in 15–70 s, the 8–22-einsum segments of a decoder block in 20–100 s each, while an uncut attention block of 34 einsums takes more than 15 minutes. The cost is accuracy in one direction only: a cut forbids fusion across it, so a segmented score can only be higher than the whole-graph optimum, by the traffic of the cut tensors (the 2% above). Repeated blocks with identical shapes are mapped once and counted per block with each block's own weights. A few small pieces of the models in §3.4 are accounted for from their byte counts at the table's costs rather than mapped, like the dispatch and combination of expert rows: the attention core, Q/K norms

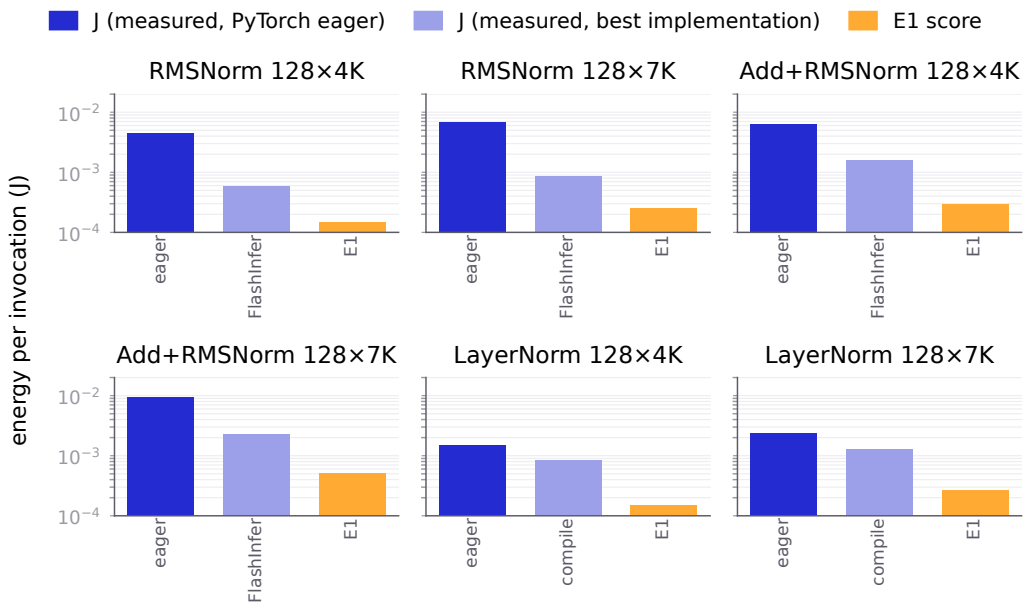


Figure 7: Energy per invocation of BF16 normalization kernels at 128 rows and 4K or 7K columns on an NVIDIA H100, in Joules on a logarithmic axis: PyTorch eager (dark blue), the best available implementation (light blue; FlashInfer for the RMSNorm family, `torch.compile` for LayerNorm), and the *E1* score (orange). Measured bars are the mean of three independent runs.

and rotary embeddings of Qwen3-30B-A3B (0.4% of its total), the rotary embedding of the query in Qwen3-8B (0.4%), and the pre-attention norms and rotary embeddings in Gemma 4 (0.5%). The attention weights themselves are in the mapped projection segments.

Mixture-of-experts layers. The router is mapped as a linear layer and a softmax; the dispatch of token rows to experts and the combination of expert outputs are accounted for from their byte counts at the table's costs; and each expert is mapped once for every distinct number of tokens it receives, using the routes observed on the prompt. Top-k selection and address generation are not counted. An expert invocation costs 0.67 mJ for its first token and $4.8\,\mu$J for each additional one, because at batch 1 the expert's weights dominate its traffic. Figure 8 shows the composition of the estimate.

E Kernel History Details

Table 2 lists the versions of FlashInfer's fused add+RMSNorm kernel measured in §3.3. Each version is the unmodified kernel source at that commit, compiled for the H100 and called through the same 40-line binding; the residual output is bit-identical across versions and the normalized output is within one BF16 unit of an FP64 reference for all of them.

Table 2: Versions of FlashInfer's fused add+RMSNorm kernel used in Figure 5b.

Version	Date	Pull request	Change	Energy vs. F0
F0	2024-08	#419	kernel introduced; second pass re-reads input, residual and weight	–
F1	2024-09	#495	drop the redundant reload of the input in the second pass	-0.2%
F2	2024-11	#587	stage the FP32 row in shared memory (scalar accesses)	$+10\%$
F3	2024-11	#592	vectorized shared-memory staging	$+1\%$
F4	2025-03	#930	programmatic dependent launch and stride arguments	$+2\%$
F5	2026-03	#2428	kernel rewritten in the CuTe-DSL backend (installed 0.6.10)	-6%

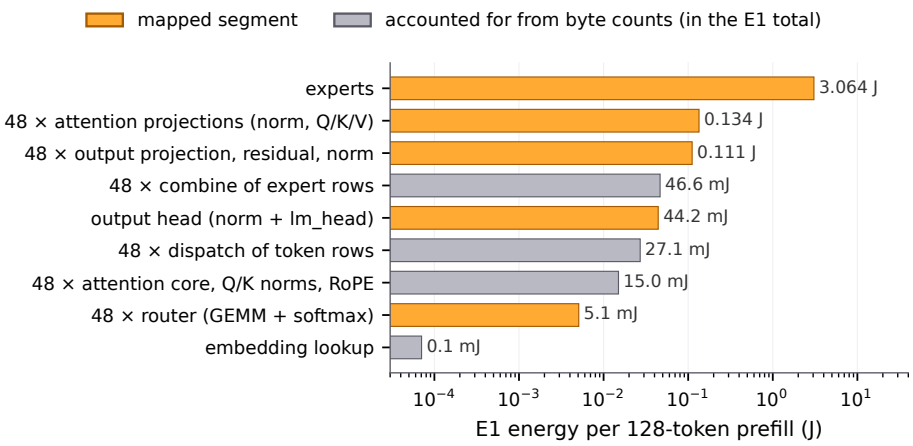


Figure 8: Composition of the *E1* score for one Qwen3-30B-A3B prefill, in Joules on a logarithmic axis: the mapped segments (orange; the experts, the attention projections with their norms, the router, the output head) and the pieces accounted for from their byte counts (grey; dispatch and combination of expert rows, attention core with Q/K norms and rotary embeddings, embedding lookup). The experts account for 89% of the estimate and their weight reads for most of that.

F Where *E1* Sits Between Simple Traffic Bounds

E1's memory term is the traffic of the schedule that the mapper selects. To show what the search buys, Figure 9 compares it with three bounds that need no schedule at all, each evaluated on the same einsum graph with the same energy table. The *optimistic* bound reads every input of the graph once and writes every output once, and nothing else ever moves: it is the least traffic any schedule could have. The *unfused* bound lets every operator read its inputs and write its output once, so every intermediate tensor takes a round trip to memory: it is what an execution without fusion pays even with perfect tiling inside each operator. The *pessimistic* bound fetches every operand for every use with no on-chip reuse at all. The compute term is identical in all cases.

Over 175 workloads (microbenchmarks, KernelBench L1 and L2, FlashInfer kernels) *E1* lies between the optimistic and the unfused bound for all but twelve; for those twelve, all large contractions or long fused chains, it exceeds the optimistic bound by 3–21%, because the operands no longer fit the on-chip buffer or part of the chain stays unfused. Neither naive bound describes an optimized schedule: the pessimistic bound is unusable for any contraction, and the optimistic bound is exact only for isolated memory-bound kernels, while an execution without fusion of the KernelBench L2 chains and the FlashInfer kernels pays 2–10$\times$ its memory energy. Only a fusion-aware search recovers it, which is what *E1*'s mapper does.

G Recurrent Blocks

A Mamba block updates a small recurrent state at every token, so at batch 1 a decode step reads the block's weights once, updates the state, and emits one token. We write one decode step as a graph whose previous state and convolution columns are explicit inputs and whose new state is an explicit output, capture it by executing it once on the CPU, and map it like any other kernel; the result is checked against Transformers' Mamba decode to 10^{-6}. Figure 10 shows the *E1* score of one step against the measured energy of the same step in eager PyTorch and under `torch.compile`, for block widths from 256 to 2560 (the width of Mamba-2.8B). Weights are 95–99% of the *E1* score at every width, so *E1* grows with the parameter count while the measurement grows more slowly: at small widths a step is a few launches' worth of time and idle power dominates, at $D = 2048$ and above the step is memory-bound and the idle-subtracted measurement is within 2$\times$ of *E1*.

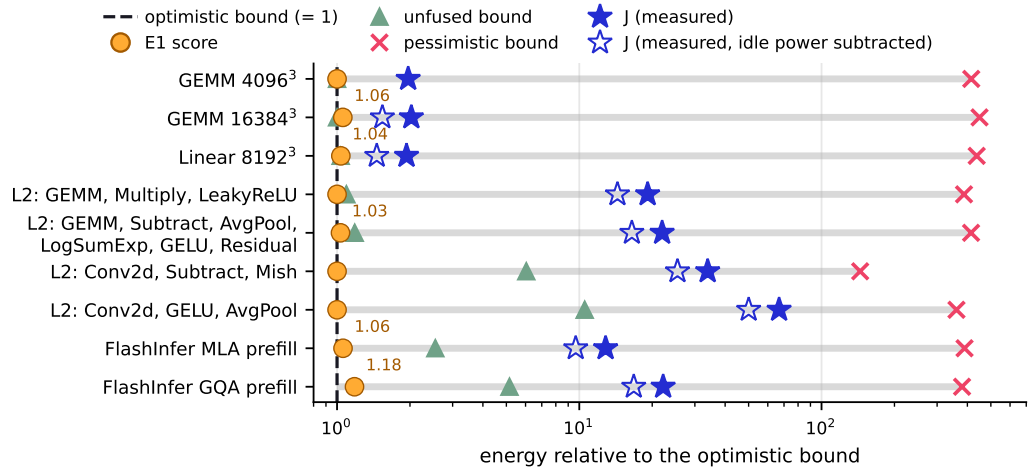


Figure 9: Energy of nine workloads relative to the optimistic bound (dashed line), on a logarithmic axis: the unfused bound (green triangles), the pessimistic bound (red crosses), the *E1* score (orange circles, labelled where it exceeds one), and the measured energy of each workload's PyTorch implementation on an NVIDIA H100 (blue stars, filled for board energy and hollow with idle power subtracted). For the single-operator GEMMs *E1* is at the optimistic bound or within 6% of it; for multi-operator programs the unfused bound is 1.1–10.5× higher and *E1* stays at or near the optimistic level by keeping intermediates on chip, exceeding it by 1.03–1.18 only where operands do not fit the buffer or part of a chain stays unfused. The measurements sit 2× above *E1* for the large GEMMs and 10–70× above it for the short programs, where idle power over a millisecond-scale invocation is most of the board energy.

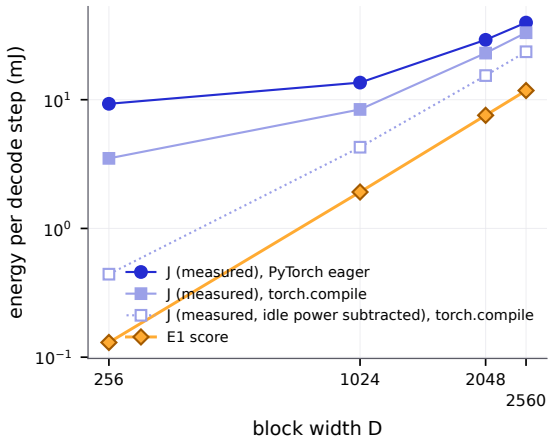


Figure 10: Energy of one Mamba decode step at batch 1 on an NVIDIA H100 against the block width D (inner width $2D$, state size 16), in mJ on logarithmic axes: measured energy of PyTorch eager (dark blue) and `torch.compile` (light blue, filled), the latter also with idle power subtracted (light blue, hollow), and the *E1* score (orange).