VERSIONICS —ARTICLE TYPE

CAIDE: A Python toolkit for cost- and carbon-aware deployment planning of large language model services

PLACEHOLDER Given-name Family-name [a,*]

[a] *PLACEHOLDER Department, PLACEHOLDER Institution, PLACEHOLDER City, PLACEHOLDER Country*

* Corresponding author. E-mail: PLACEHOLDER@institution.edu

Highlights

- Derives efficiency multipliers from a roofline model instead of quoting constants
- Shows published constants err by up to 153% because they ignore batch regime
- Separates demand duty cycle from scheduler efficiency, which optimization cannot raise
- Reports every break-even crossing, plus the band where cost cannot decide
- Installs with one command; first analysis completes in under two minutes on a laptop

Abstract

Organizations deploying large language model services must choose between commercial endpoints and self-hosted infrastructure, yet the cost analyses guiding these decisions rely on fixed efficiency multipliers that are quoted as constants and multiplied together. We present CAIDE, an open-source Python toolkit that instead models each efficiency technique as a transformation of the deployment's physical state and measures its cost multiplier by re-running a roofline model on the transformed configuration. CAIDE separates compute-bound prefill from memory-bound decode, sizes grouped-query key-value caches, models mixture-of-experts activation as a function of batch, bisects for the largest batch meeting a latency objective, holds demand duty cycle and scheduler efficiency as distinct quantities, decomposes total cost of ownership into six layers with distinct scaling laws, charges capacity in whole replicas, locates every break-even crossing between architectures, and propagates input uncertainty to a rank-correlation sensitivity ranking. Applied to published multipliers, CAIDE shows that speculative decoding ranges from 0.40× to 0.81× across batch sizes and that stacking quantisation with paged attention delivers 29% less benefit than the product of their constants. Three worked deployments spanning engineering education, clinical documentation and public-service delivery show inference accounting for 2–10% of ownership cost. CAIDE is released under the MIT licence. This article publishes version 5.0.0.

Keywords: large language models; deployment planning; total cost of ownership; roofline model; inference serving; carbon accounting; capacity planning; uncertainty analysis

Artifact Manifest (Table 1, mandatory)

Nr.	Field	Value
A1	Artifact type (Versionics article type)	Software Tool Article
A2	Software name and reviewed version	CAIDE v5.0.0
A3	Public source repository	https://github.com/PLACEHOLDER-ORG/caide

A4	Archived release of the reviewed version (version DOI)	https://doi.org/10.5281/zenodo.PLACEHOLDER (v5.0.0)
A5	Concept DOI aggregating all versions	https://doi.org/10.5281/zenodo.PLACEHOLDER-CONCEPT
A6	Licence (SPDX identifier, in repository root)	MIT
A7	Pinned environment	Pure Python, no compilation; pip-installable with pinned requirements.txt in the release archive; no container required — hardware is modelled, not used
A8	Supported platforms	Linux, macOS, Microsoft Windows; and any other platform that supports CPython ≥ 3.9, etc.
A9	Executable distribution of the reviewed version	https://pypi.org/project/caide/5.0.0/
A10	Documentation	README (quick-start) and docs/model.md; examples bundled: caide examples --extract .
A11	Tests and continuous-integration status	245 automated tests, 89% statement coverage; GitHub Actions on Linux/macOS/Windows
A12	Worked example with sample data	Three shipped scenarios; examples/reproduce_paper.py regenerates every figure and number under a fixed seed
A13	Citation metadata	CITATION.cff and codemeta.json in repository root
A14	Support channel	Public issue tracker; PLACEHOLDER@institution.edu

All A-fields must match the repository state of the reviewed version exactly; mismatches are returned at technical check. Fields A4–A6 bind this article to a frozen release: the version DOI resolves to the archived artifact, the concept DOI aggregates all versions, and the SWHID/commit identifies source content.

Usability Summary (Table 2, mandatory)

Nr.	Field	Value
U1	Installation command	`pip install caide`
U2	Time to first result (author-measured, reviewer-verifiable)	Under two minutes from installation to a complete first analysis, on a laptop, no accelerator
U3	Interfaces provided	Command line (eight verbs); importable Python API; Markdown/CSV/single-file HTML dashboard output
U4	Quick-start path	`caide examples --extract . then caide run examples/university_tutoring.yaml --layers --out report/`
U5	Behaviour on invalid input	Strict validation: values range-checked, errors name the offending path, unrecognised keys reported with a suggested correction
U6	Intended users and stated limitations	Deployment planners, institutional IT, researchers costing LLM services; limitations catalogued in docs/model.md

The Usability Summary is verified during artifact review: an independent reviewer follows only U1 and U4 from a clean environment and reports installation time, obstacles and documentation gaps. The reviewer's account carries the same weight as the scholarly assessment.

1. Problem and intended users

Any organisation putting a large language model into service faces the same sequence of decisions: which model, on which accelerators, served by which stack, at what latency, and whether to build the capability or buy it. Each decision moves cost by an order of magnitude, and they interact. The intended users of CAIDE are the people who must answer these questions and defend the answer — deployment planners, institutional IT and procurement teams, and researchers costing language-model services — none of whom should need serving infrastructure of their own to run the analysis.

The analyses that guide these decisions are typically assembled from published efficiency multipliers. A recent framework for cost-aware generative AI deployment is representative: it tabulates four-bit quantisation at 0.65×, continuous batching at 0.30×, speculative decoding at 0.40×, and a fully optimised stack at 0.03–0.08×, then applies the product to a token tariff [1]. The approach is attractive because it requires no infrastructure to compute, and it is the method visible in institutional planning documents across sectors.

It has two defects, and both are structural rather than a matter of the constants being out of date.

First, a constant cannot express regime dependence. Speculative decoding accelerates decoding by having a draft model propose tokens that the target model verifies in parallel [2,3]. At batch one, where decoding is starved of arithmetic and bound entirely by streaming weights out of memory, the technique is transformative. At batch 256, where the accelerator is already compute-saturated, the draft model competes for the same floating-point units and the benefit largely evaporates. A single number is correct at one operating point and wrong everywhere else. Worse, the published 0.40× figure is characteristic of an unbatched deployment, yet it is routinely multiplied against a continuous batching multiplier that assumes the opposite regime.

Second, a constant cannot express interaction. Four-bit quantisation shrinks resident weights, freeing high-bandwidth memory [4,5]. PagedAttention recovers memory otherwise lost to key-value cache fragmentation [6]. Both interventions enlarge the feasible batch by attacking the same bottleneck, so applying the second after the first finds much less left to recover. Multiplying their constants overstates the combined benefit substantially in memory-constrained configurations and barely at all where headroom is ample. The error therefore has no reliable sign, which is what makes it difficult to correct by applying a safety margin.

Underlying both defects is a third, conceptual problem. Reference unit-cost tables report a fully loaded replica, whereas real services are provisioned for peak demand and idle through the night. CAIDE holds the demand duty cycle — a property of the traffic — apart from scheduler efficiency — a property of the stack — because optimisation can raise the second but no serving stack creates traffic. CAIDE exists to replace inherited constants with quantities measured against the user's own configuration, without requiring access to hardware.

2. Software architecture

CAIDE is a pure Python package requiring only NumPy, PyYAML and Matplotlib. It runs on any platform and needs no accelerator: it models hardware rather than using it. Fig. 1 shows the five bands of the architecture.

The input band accepts a declarative YAML scenario describing workload classes, candidate architectures, service objectives, cost layers and input distributions. Validation is strict: values are range-checked, errors name the offending path, and unrecognised keys are reported with a suggested correction — misspelling a review-rate field would otherwise silently zero a cost component that dominates every shipped example.

The physics band holds `specs` (unit-checked dataclasses), `catalog` (nine model archetypes, six accelerators,

four pricing tiers, nine grids), `efficiency` (fifteen techniques as state transformations) and `roofline`. The economics band converts performance into money: `costing` assembles six ownership layers, `perturb` exposes the substitution used for uncertainty analysis, and `routing` solves the minimum-cost assignment of workload classes to a tier ladder. The analysis band contains `breakeven`, `uncertainty` and `scaling`. The output band renders Markdown, CSV or a dependency-free single-file HTML dashboard, and exposes eight command-line verbs alongside the Python API.

The feedback arrow in Fig. 1 is the design's essential feature: transformed states return to the roofline for re-evaluation rather than carrying a precomputed multiplier forward.

[Insert figure here] *Fig. 1. CAIDE architecture. Efficiency techniques transform the deployment state; the resulting cost multiplier is measured by re-evaluating the roofline on that transformed state rather than supplied as a constant.*

3. Software functionalities

Roofline serving model. Prefill cost is $2 \cdot N_{active} \cdot T_{in}$ floating-point operations plus a causal attention term $2 \cdot L \cdot T_{in}^2 \cdot d$, negligible below roughly two thousand tokens and dominant above thirty thousand — the regime retrieval-augmented pipelines actually occupy. Decode cost is the maximum of memory traffic and arithmetic, not their sum, since the two overlap on real hardware. Memory traffic comprises the weight stream, which is independent of batch, and the key-value cache stream, which is not. Because both terms are linear in batch, which one binds is determined by context length rather than batch size: above a crossover, a workload remains memory bound at every batch, so batching cannot reach the compute roof. Key-value cache sizing is grouped-query aware [9]. Mixture-of-experts weights are treated as resident in full while the expected fraction of experts touched by B independent tokens follows $1-(1-k/E)^B$ [10], which is why such models forfeit their bandwidth advantage precisely when batching would otherwise supply one. Decode FLOP utilisation rises with batch, because a step at batch one is a matrix-vector product and at batch 256 a matrix-matrix one; holding it constant places the memory-to-compute transition at an artificially low batch. Time to first token is obtained by treating prefill as an M/D/1 queue whose utilisation is its share of the serving cycle.

Service-objective solver. `solve_batch_for_slo` bisects for the largest batch still meeting time-to-first-token and time-per-output-token targets. Throughput rises monotonically with batch while latency degrades monotonically, so the feasible set is an interval and its boundary is exact. Serving at that boundary rather than at the scheduler cap is what separates a cost model respecting user experience from one that does not. Requests exceeding the model's context window, or batches exceeding the KV cache, are reported as infeasible rather than priced.

Efficiency techniques as transformations. Fifteen techniques edit physics rather than multiply cost. Quantisation alters bytes per parameter [4,5,19]; PagedAttention raises usable memory [6]; continuous batching raises the achievable batch and the scheduler efficiency [11]; chunked prefill smooths the latency–throughput tradeoff [12]; FlashAttention raises achievable prefill utilisation [18]; prefix caching shortens effective prompts; distillation substitutes a smaller student of proportionate shape. No technique may raise the demand duty cycle, because no serving stack creates traffic. Mutually exclusive combinations are rejected rather than silently composed.

Six-layer ownership model. Model access scales linearly with volume; compute and serving as a step function, since capacity arrives in whole replicas; retrieval sublinearly, because an index is built once and queried many times; integration and assurance are volume-free; workforce redesign is front-loaded and decays. Only the first is linear, so a model containing only that layer understates cost at low volume and overstates the benefit of efficiency work at high volume. The asymmetry between an affordable marginal inference cost and

an unaffordable fixed training cost [16,17] makes the surrounding layers, not the model itself, the object of the decision.

Human oversight. Review cost is charged per query, and the implied reviewer hours and full-time equivalents are reported alongside. That check exists because it caught an error during development: an early draft of one shipped example silently assumed 359 full-time reviewers while its dollar total looked reasonable. Separately, `baseline_minutes` records the human time a task consumed before the system existed; displaced effort is reported but never subtracted, since a labour saving and a cash outlay are different instruments.

Break-even analysis. Because a staircase and a line may intersect many times, CAIDE brackets every sign change on a log-spaced scan and bisects in logarithmic space. When crossings proliferate, enumerating them implies precision the model lacks, so the tool reports the volume band within which the options differ by less than a tolerance and states plainly that inside it cost does not decide.

Uncertainty and sensitivity. Monte Carlo propagation over five distribution families, with Spearman rank correlation for sensitivity because step functions and service-objective cliffs make monotone-but-curved relationships common. Failed draws are counted rather than discarded, since a configuration infeasible in a third of draws is a finding.

Scaling dynamics. With unit cost falling geometrically and demand responding with constant price elasticity ε, spend evolves as $(c/c_0)^{1-\varepsilon}$. Above unit elasticity — the regime Jevons identified for coal and observed since for electricity and bandwidth [13] — falling prices raise total spend. `estimate_elasticity` fits ε from an organisation's own history, replacing the assumption with data. Facility energy follows from board power, duty cycle and power usage effectiveness; carbon and water follow from grid intensity [7,8].

4. Installation and first use

Installation and a first analysis take under two minutes. The example scenarios ship inside the package, so nothing here requires a repository checkout:

```
pip install caide
caide examples --extract .
caide run examples/university_tutoring.yaml --layers --out report/
```

The library is equally usable directly:

```
from caide import *

state = DeploymentState(get_model("dense-70b"),
                        get_hardware("h100-sxm"),
                        ServingConfig(n_accelerators=4, max_batch=256,
                                      demand_duty_cycle=0.42,
                                      scheduler_efficiency=0.45))

perf = solve_batch_for_slo(state,
                           WorkloadClass("tutor", 1.0, 1500, 400),
                           SLO(ttft_seconds=1.5, tpot_seconds=0.045))
```

Misuse fails legibly rather than silently: scenario values are range-checked, errors name the offending path, and an unrecognised key is reported with a suggested correction. Requests that exceed the model's context window, and batches that exceed the key-value cache, are declared infeasible instead of being priced. All quantitative results in Section 5 are regenerated by examples/reproduce_paper.py under a fixed seed, on the

shipped example scenarios, in about ninety seconds on a laptop.

5. Illustrative examples and validation

Regime dependence. Sweeping batch size for a 70-billion-parameter model on four accelerators (Fig. 2), speculative decoding's measured multiplier moves from 0.401× at batch one to 0.822× at batch 256, a spread of 2.1×. The published constant of 0.40× is accurate at batch one and wrong by 51% at batch 256. Four-bit quantisation ranges from 0.257× to 0.503× against a published 0.65×, a discrepancy reaching 153%. Semantic caching, which bypasses inference rather than accelerating it, is genuinely batch-invariant at 0.700× — so the method distinguishes techniques that are legitimately constant from those that are not, rather than merely disputing every published figure. The command `caide sweep` reproduces this scan for any scenario and technique.

[Insert figure here] *Fig. 2. Measured cost multipliers (solid) against the fixed constants a published table assigns (dashed). Only semantic caching, which bypasses inference entirely, is genuinely batch-invariant.*

Interaction. In a memory-constrained configuration, four-bit quantisation alone yields 0.199× and PagedAttention alone 0.704×. Their product predicts 0.140×; the measured pair delivers 0.198×. The product understates cost by 29%. In a configuration with ample headroom the same comparison diverges by only 1.5%.

Utilisation. A fully loaded replica with a perfect scheduler and no infrastructure overhead serves a retrieval-augmented turn for $0.000102. Isolating each departure: a demand duty cycle of 0.42 costs 2.38×, infrastructure overhead 1.34×, and scheduler efficiency nothing further because the optimised stack has already raised it to its ceiling. Together, $0.000326 — a factor of 3.19×. The decomposition is only possible because the two utilisation terms are held apart; with one parameter the stack would appear to raise utilisation above declared demand, which no scheduler can do.

Break-even structure. Comparing an economy API against a self-hosted 70-billion-parameter deployment across three orders of magnitude (Fig. 3) yields three crossings between 646 million and 1.29 billion queries per year, because each added replica overshoots demand and returns the advantage until it fills. CAIDE reports the indistinguishable band — 619 million to 1.35 billion queries annually — and directs the decision to latency, data residency, control and staffing.

[Insert figure here] *Fig. 3. Integral replica capacity turns the self-hosted curve into a staircase, so a single break-even threshold misrepresents a band in which cost does not decide. Shading marks which architecture is cheaper.*

Cross-domain application. Three shipped scenarios cover engineering education (9 million queries annually), clinical documentation (6.6 million) and public-service delivery (420 million). Inference accounts for 10.4%, 2.2% and 2.4% of ownership cost (Fig. 4). Candidate architectures differ by 14.4%, 1.4% and 2.8% — in the latter two, less than the uncertainty on the fixed layers, so the model choice is not the decision that matters. The clinical scenario reports $97.4M of displaced clinician effort against $24.4M of cost, a comparison invisible to any model counting only outlays.

[Insert figure here] *Fig. 4. Annual cost of ownership by layer for the engineering-education scenario. Assurance and governance, which does not shrink when token prices fall, is the tall band in every architecture.*

Uncertainty. Propagating six input distributions through the education scenario, the self-hosted architecture's annual cost spans $2.18M to $3.95M between the fifth and ninety-fifth percentiles, and the commercial endpoint is cheaper in every one of 4,000 draws. Sensitivity (Fig. 5) attributes 52.0% of explained variance to volume uncertainty and 47.3% to the review rate; accelerator price, the input most often negotiated hardest, contributes 0.8%. That figure is reported for the self-hosted architecture on purpose: on a commercial-endpoint

architecture the cost model contains no accelerator price term at all, so the near-zero sensitivity it returns would be a statement about the architecture rather than evidence about prices.

[Insert figure here] *Fig. 5. Sensitivity of annual cost for the self-hosted architecture. Bars span the median outcome at the tenth and ninetieth percentile of each input.*

Validation. Predicted decode throughput was compared against four published measurements spanning three serving frameworks. Two fall within a factor of two and the model over-predicts in three, placing the uncalibrated roofline at a factor of two to three with a bias toward optimism — adequate for ranking architectures, whose bias is largely common to them, and inadequate for absolute capacity commitments. The calibration module fits a multiplicative correction to decode utilisation from a user's own measurements; on the four reference points it lifts the fraction inside the band from 50% to 75%. Part of the residual is structural: a roofline models the accelerator, while published figures are wall-clock and include the scheduler and API layer, which one framework's profiling put at 62% of its time on an 8B model. The conclusions are not artefacts of the modelling choices: repeating the sweep under nine alternative assumptions — decode saturation batch, prefill and decode utilisation, tensor-parallel penalty — leaves the spread above 1.3× in all nine and the discrepancy against the published constant between 151% and 154%, while semantic caching stays exactly batch-invariant.

Scaling. Under a 38% annual price decline, elasticity 0.6 reduces five-year spend to 0.73×; elasticity 1.35 raises it to 3.07×; elasticity 1.8 to 7.26×. An institution forecasting from the unit-cost trend alone gets the direction wrong, not merely the magnitude.

6. Quality assurance and reproducibility

The reviewed version is covered by 245 automated tests at 89% statement coverage, run by continuous integration on Linux, macOS and Windows; three tests exist specifically to detect defect classes an earlier test suite could not. Each generated report records the software version, a digest of the scenario, the random seed and the scenario itself, so a figure quoted in a budget submission can be regenerated years later from a single artefact rather than merely verified as unchanged. Bundled prices carry an epoch, and the tool warns when they age. The scenarios and examples/reproduce_paper.py — which regenerates every figure and numerical result in this article in about ninety seconds — are packaged with the software rather than living only in the repository, and are included in the archived release identified in fields A4–A6.

7. Impact and limitations

CAIDE changes what a deployment analysis can establish. Cost analyses in this area have been narrative, their quantitative content inherited constants that could not be checked against a specific configuration. CAIDE makes those constants testable, and the tests are not reassuring: two of the three examined depart from measured behaviour by more than half their value across the operating range production deployments span. Any exercise built by multiplying such constants inherits an error whose direction cannot be anticipated.

For practitioners the immediate change is in what gets debated. In all three worked deployments, inference is a minority of ownership cost and the candidate architectures are separated by less than the uncertainty on the surrounding layers. Effort spent negotiating accelerator prices — 0.8% of explained variance — is effort not spent on the review policy, which carries 47%. Making that visible redirects planning attention toward the quantities that decide the outcome.

Applicability extends beyond any single discipline. The three shipped scenarios were chosen because the shape

of the answer inverts between them: assurance dominates the educational case, mandatory clinical review the second, sheer volume the third. Nothing in the model is domain-specific; a workload class is a token profile with a review policy, and any sector deploying language model services is described by the same objects. Being importable as a library, CAIDE can also serve as a costing component inside larger procurement systems. Energy, carbon and water are computed alongside cost from the same model, so emissions can be examined without a second analysis that might disagree about the configuration.

The principal limitations are stated in Section 5 and catalogued in docs/model.md: the uncalibrated roofline is accurate to a factor of two to three with a bias toward optimism, adequate for ranking architectures but not for absolute capacity commitments without local calibration; published wall-clock figures include scheduler and API layers outside the model's scope; and multi-adapter serving and spot capacity are not yet represented.

8. Version history and maintenance

Reviewed version. This article publishes CAIDE v5.0.0, the version archived under the DOI in field A4 and identified by the tag, commit and SWHID in field A6. The concept DOI in field A5 aggregates all versions, past and future.

Version history. CAIDE follows Semantic Versioning. Versions 1–4 were internal development milestones and are retained in the repository history; v5.0.0 is the first publicly released and first published version, so this article carries no relation to a prior Versionics publication. PLACEHOLDER — if any earlier version was released or archived publicly, list each with its DOI and one line of scope here.

Patch policy. Patch releases (5.0.x) — bug fixes, price-catalogue refreshes and documentation corrections — will be published in the repository and archived under the concept DOI without amendment of this article; a patch that materially alters any numerical result reported here will instead be accompanied by a formal correction.

Future substantial releases. The planned development line follows the limitations in docs/model.md: multi-adapter serving and spot-capacity modelling. A release delivering material new capability of that kind will be submitted as a Version Release Article, linked to this article and to the versions it supersedes through Crossref relation metadata, with differential results against v5.0.0.

Maintenance commitment. The reviewed version will remain installable from the archived release regardless of later development. Issues are triaged through the public tracker in field A16; deprecations will be announced one minor version in advance and documented with migration notes in the changelog.

9. Conclusions

CAIDE is an open-source toolkit for planning large language model deployments that treats efficiency multipliers as quantities to be derived rather than constants to be quoted. By modelling each technique as a transformation of the deployment's physical state and measuring its effect with a roofline model, it reproduces regime dependence and interaction effects that fixed multipliers cannot express, and shows published constants departing from measured behaviour by up to 153%. Holding demand duty cycle apart from scheduler efficiency, six-layer ownership accounting, integral-replica capacity, break-even bands and Monte Carlo sensitivity together produce analyses that state their own confidence. The software installs with one command, completes a first analysis in under two minutes, and ships the scenarios and script that regenerate every published result.

CRediT authorship contribution statement

PLACEHOLDER Given-name Family-name: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization.

Declaration of competing interest

The author declares no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The author further declares no affiliation with any vendor of hardware, models or serving infrastructure named or priced in the shipped catalogues.

Funding

PLACEHOLDER — state grant numbers, or: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Artifact and data availability

All source code, scenario files and the script that reproduces every figure and numerical result in this article are openly available in the repository (field A3), permanently archived at the version DOI (field A4) with all versions aggregated under the concept DOI (field A5), and additionally packaged with the software. No data other than the shipped example scenarios were used.

Declaration of generative AI in the writing process

PLACEHOLDER — complete per Versionics policy. If generative AI tools were used in preparing this manuscript or the software, state the tool and purpose here, and confirm that the authors reviewed, tested and take full responsibility for the content and code. If no such tools were used, state so explicitly; do not delete this section.

References

[1] PLACEHOLDER — companion cost framework. Cost-aware deployment of generative AI systems in engineering education: unit economics, pipeline architectures, and scaling dynamics. Complete with journal, volume and year on acceptance.

[2] Leviathan Y, Kalman M, Matias Y. Fast inference from transformers via speculative decoding. In: Proceedings of the 40th International Conference on Machine Learning (ICML); 2023. p. 19274–86.

[3] Chen C, Borgeaud S, Irving G, Lespiau J-B, Sifre L, Jumper J. Accelerating large language model decoding with speculative sampling. arXiv:2302.01318; 2023.

[4] Frantar E, Ashkboos S, Hoefler T, Alistarh D. GPTQ: accurate post-training quantization for generative pre-trained transformers. In: International Conference on Learning Representations (ICLR); 2023.

[5] Lin J, Tang J, Tang H, Yang S, Dang X, Han S. AWQ: activation-aware weight quantization for LLM compression and acceleration. In: Proceedings of Machine Learning and Systems (MLSys); 2024.

[6] Kwon W, Li Z, Zhuang S, Sheng Y, Zheng L, Yu CH, et al. Efficient memory management for large language model serving with PagedAttention. In: Proceedings of the 29th Symposium on Operating Systems Principles (SOSP); 2023. p. 611–26.

[7] Patterson D, Gonzalez J, Le Q, Liang C, Munguia L-M, Rothchild D, et al. Carbon emissions and large neural network training. arXiv:2104.10350; 2021.

[8] Luccioni AS, Viguier S, Ligozat A-L. Estimating the carbon footprint of BLOOM, a 176B parameter language model. J Mach Learn Res 2023;24(253):1–15.

[9] Ainslie J, Lee-Thorp J, de Jong M, Zemlyanskiy Y, Lebrón F, Sanghai S. GQA: training generalized multi-query transformer models from multi-head checkpoints. In: Proceedings of EMNLP; 2023. p. 4895–901.

[10] Fedus W, Zoph B, Shazeer N. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. J Mach Learn Res 2022;23(120):1–39.

[11] Yu G-I, Jeong JS, Kim GW, Kim S, Chun B-G. Orca: a distributed serving system for transformer-based generative models. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2022. p. 521–38.

[12] Agrawal A, Kedia N, Panwar A, Mohan J, Kwatra N, Gulavani B, et al. Taming throughput–latency tradeoff in LLM inference with Sarathi-Serve. In: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI); 2024. p. 117–34.

[13] Jevons WS. The coal question: an inquiry concerning the progress of the nation, and the probable exhaustion of our coal-mines. London: Macmillan; 1865.

[14] Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures. Commun ACM 2009;52(4):65–76.

[15] Pope R, Douglas S, Chowdhery A, Devlin J, Bradbury J, Levskaya A, et al. Efficiently scaling transformer inference. In: Proceedings of Machine Learning and Systems (MLSys); 2023. p. 606–24.

[16] Hoffmann J, Borgeaud S, Mensch A, Buchatskaya E, Cai T, Rutherford E, et al. Training compute-optimal large language models. In: Advances in Neural Information Processing Systems (NeurIPS); 2022.

[17] Kaplan J, McCandlish S, Henighan T, Brown TB, Chess B, Child R, et al. Scaling laws for neural language models. arXiv:2001.08361; 2020.

[18] Dao T, Fu DY, Ermon S, Rudra A, Ré C. FlashAttention: fast and memory-efficient exact attention with IO-awareness. In: Advances in Neural Information Processing Systems (NeurIPS); 2022.

[19] Dettmers T, Pagnoni A, Holtzman A, Zettlemoyer L. QLoRA: efficient finetuning of quantized LLMs. In: Advances in Neural Information Processing Systems (NeurIPS); 2023.