”AI” Reads Human: Human Language Encoding at Machine Speed and the Energy Cost

David Charlot, Open Interface Engineering

Work in progress · Open Interface Engineering · research.openie.dev

Revision `af57f48` (22 November 2026) · Web: research.openie.dev/papers/ai-reads-human/ · Archival LaTeX: /pdfs/latex/ai-reads-human.pdf (.tex)

Measurement. Research study, not a journal final. Parse time, token counts, and Apple M5 Max dual-channel `reported_j` (`L3 proc_pid_rusage(RUSAGE_INFO_V6).ri_energy_nj` and `L2 macmon IOReport` across **`34 workloads`** and `170` windows on the reproducible `20,000`-record corpus: **`280.4 nJ/record`** fixed binary `struct` to **`2,212.1 nJ/record`** stdlib `JSON`, **`69,922.1 nJ/record`** C `libyaml`, **`575,799.3 nJ/record`** pure-Python `YAML`, and **`27,678.7 nJ/record`** `cl100k_base JSON` tokenization; `Falsifier 6` triggered on `109/170` shared-host `L2` windows) were measured locally. No hardware power rail or wall meter was attached (`measured_j = None`, `board_metered = false`); model energy figures are other authors' measurements.

Work in progress. Why so much of what machines exchange is written for people, what that costs in bytes, tokens, time, and per-process Apple Silicon kernel rusage energy (reported_j) across 34 parse and tokenize workloads, and why external wall-meter energy (measured_j) remains open.

Contents

Abstract 2

1. Background: formats written for people 2

2. The impedance tax, defined 3
2.1 Physical scale, and what this page does not claim . . . 4

3. What was measured 4
3.1 Parse time . . . 5
3.2 Tokens . . . 6

4. Why the hazards come with the readability 8

5. Planned: measuring energy 9

5b. Measured energy: no net effect resolved 10

5c. Dual-channel Apple M5 Max `reported_j` across all 34 parse and tokenize workloads 10
Telemetry channels and Falsifier 6 demonstration . . . 10
Parse-stage `L3 reported_j` (`16` workloads, `20,000` records) . . . 10
Tokenization-stage `L3 reported_j` (`18 tiktoken` workloads, `20,000` records) . . . 11

6. Where a canonical machine form fits 12
6.1 An open design question . . . 12

7. Reproduce this page 12

8. Threats to validity 12

9. Read ledger and references 13

10. What will be added 13

Work in progress, revision af57f48, 22 November 2026. Interactive observatory, data, code and read ledger: research.openie.dev/living/ai-reads-human/. Shared terms: glossary.

Abstract

Software agents pass data to each other in formats made for people to read, such as JSON, YAML, Markdown, XML and CSV. This study names the extra work this causes the impedance tax, defines it with units, and measures three parts of it on a seeded 20,000-record corpus: parse time, token count across five open tokenizers, and dual-channel Apple M5 Max operating-system/on-chip `reported_j` counters across all 34 parse and `tiktoken` workloads (section 5c). While an initial unisolated whole-chip attempt on two busy laptops could not separate parsing from background co-tenant drift (section 5b), isolating each workload inside a dedicated worker subprocess with macOS kernel per-process `proc_pid_rusage(RUSAGE_INFO_V6).ri_energy_nj` (L3, label D) resolves a `2,053.8×` parse-energy span — from `280.4 nJ/record` for fixed binary (`struct.iter_unpack`) to `2,212.1 nJ/record` for stdlib JSON (`7.89×`), `69,922.1 nJ/record` for C `libyaml` (`249.4×`), and `575,799.3 nJ/record` for pure-Python YAML (`2,053.8×`) — and shows that `cl100k_base` tokenization of compact JSON (`27,679.2 nJ/record`, `98.7×` fixed binary) spends `12.5×` the on-chip `reported_j` of `json.loads()` itself. Concurrently, whole-chip `IOReport` (L2, label A2) triggered Falsifier 6 (`> 5.0%` pre/post idle baseline drift) on `109` of `170` shared-host measurement windows, confirming why unisolated package subtraction fails on active hosts. No external inline power probe or wall meter was attached (`measured_j = None`, `board_metered = false`), and GPU model prefill/decode joules remain planned (M4).

In plain words

1. Computers and people often use the same written format. That format is easy for people to read, but the computer has to do extra work to read it back.
2. On our 20,000-record test corpus, reading fixed binary (`struct.iter_unpack`) cost `280.4 nJ/record` in Apple M5 Max per-process kernel `ri_energy_nj` (`reported_j`), compared with `2,212 nJ/record` (`7.89×`) for Python's built-in JSON reader, `69,922 nJ/record` (`249.4×`) for C-accelerated YAML, and `575,799 nJ/record` (`2,053.8×`) for pure-Python YAML.
3. Tokenizing compact JSON for an AI model with `tiktoken` (`cl100k_base`) cost `27,679 nJ/record` (`1.69×` CSV tokenization and `12.5×` `json.loads()` parsing itself), before the language model runs a single matrix multiply. All energy figures on this page are kernel/on-chip `reported_j` counters (`measured_j = None`, `board_metered = false`); GPU model prefill/decode energy and external hardware power probes are still planned.

   Status: work in progress, and it will grow. Sections marked *Planned* (such as GPU prefill/decode counters M4, natural-language corpora M6, and cross-language C/Rust parsers M8) are not done. Every number on this page is recomputed from downloadable files under `/living/ai-reads-human/`. On-chip energy counters are strictly `reported_j` (`measured_j = None`, `board_metered = false`).

1. Background: formats written for people

Most data that agents exchange today travels in formats whose specifications put people ahead of machines. The passages cited below were read for this page; each source's read status (FULL, PARTIAL or UNAVAILABLE) is in the read ledger.

- **TeX** began in 1982: the source file `tex.web` (header, first 19,000 bytes read; PARTIAL) records that version 0 was released in September 1982 and dates later versions from 3.0 (March 1990) onward.
- **Markdown** was developed by John Gruber with help from Aaron Swartz and released in 2004 as a syntax description and a Perl script that converts to HTML; the CommonMark specification (version 0.31.2, 28 January 2024) says that description "does not specify the syntax unambiguously" and lists questions it leaves open (CommonMark 0.31.2, sections 1.1 and 1.2).
- **CSV** was documented in October 2005 by RFC 4180, an *informational* RFC that says "there is no formal specification in existence" and writes down what most implementations seem to do.
- **JSON** is defined by RFC 8259 (December 2017, obsoleting RFC 7159) as "a lightweight, text-based, language-independent data interchange format." RFC 7493 (March 2015, I-JSON) narrows it for protocols.
- **Canonical and binary forms** also exist in the standards record: RFC 8785 (June 2020) defines a JSON Canonicalization Scheme, and RFC 8949 (December 2020, obsoleting RFC 7049) defines CBOR, a binary data format with a section on deterministic encoding (section 4.2) that protocols may adopt.

- **YAML** revision 1.2.2 (1 October 2021) lists its design goals "in decreasing priority," and the highest-priority goal is "YAML should be easily readable by humans" (section 1.1).

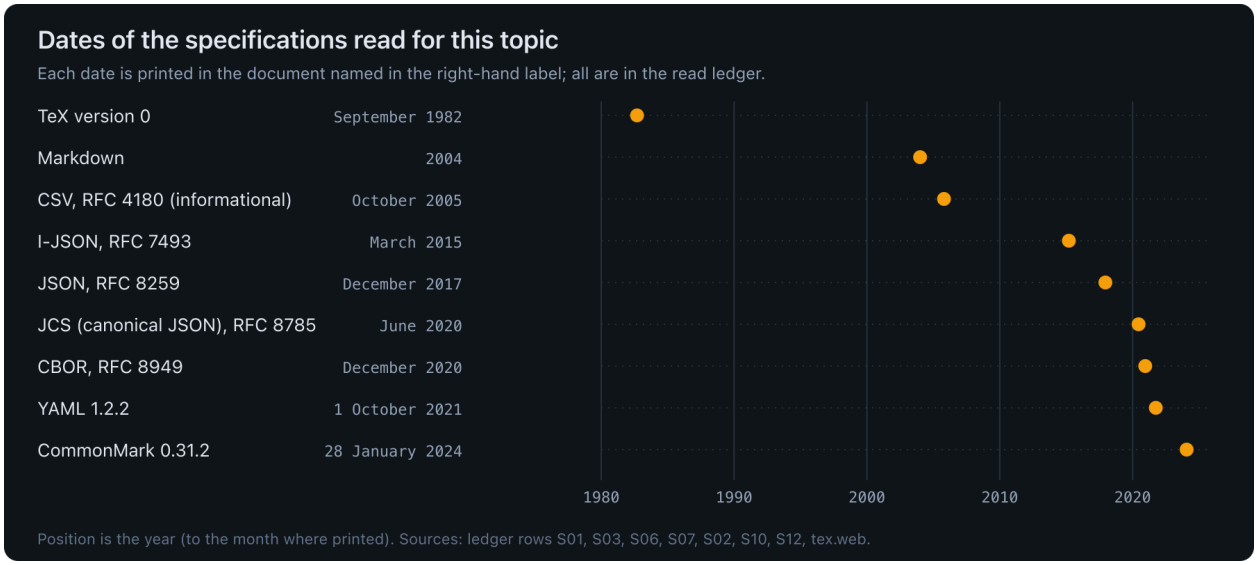

Figure 1. Dates printed in the documents listed in the read ledger. Generated by `scripts/make_charts.py`.

These facts do not say that human-readable formats are mistakes. They say that readability by people was a stated design goal, and that the cost of reading them back by machine was not the thing being minimised. Where other work reaches machine-oriented forms (CBOR, canonical JSON, Protocol Buffers, Avro, safetensors, GGUF), this study records it as a convergence observation: several communities arrived at fixed or deterministic encodings for machine exchange while keeping text for people.

2. The impedance tax, defined

Let a set of records R carry information that a machine consumer needs. Let F be a format with encoder enc_F and decoder dec_F, and let M be a canonical machine form: one valid byte string per value.

Symbol	Meaning	Unit
B_F	encoded size of R in format F	byte
N	number of records	1
k	a tokenizer	n/a
T_{F,k}	tokens produced by k on the encoding of R	token
d_{F,k} = B_F / T_{F,k}	bytes per token	byte/token
t_parse(F)	wall time to decode R to in-memory values	s
P	average power during an operation	W = J/s
E	energy of an operation, E = $\int$ P dt, approximately P_avg · t	J
e_prefill	model energy per input token	J/token

The **impedance tax** of F relative to M is the extra energy a consumer spends because the information arrives in F:

$$Z(F) = E_enc(F) + E_parse(F) + E_tok(F,k) + e_prefill \cdot T_\{F,k\} + E_ctx(F)$$

$$\tau(F) = Z(F) - Z(M) \text{ (joules)}, \quad \theta(F) = Z(F) / Z(M) \text{ (dimensionless)}$$

E_ctx(F) is the extra decode cost that a longer context causes; its coefficient depends on the model and is a measurement target (M4 below). Section 5c measures `E_parse(F)` and `E_tok(F,k)` in per-process kernel `reported_j` (`ri_energy_nj`) on Apple M5 Max.

Three dimensionless ratios can be measured without any energy counter:

- byte ratio β_F = B_F / B_ref

- token ratio κ_{F,k} = T_{F,k} / T_{ref,k}
- parse-time ratio π_F = t_parse(F) / t_parse(ref)

and they are linked by one identity that the token numbers below obey:

κ_F = β_F · (d_ref / d_F)

so a format can cost more tokens because it has more bytes, because its bytes split into more pieces, or both.

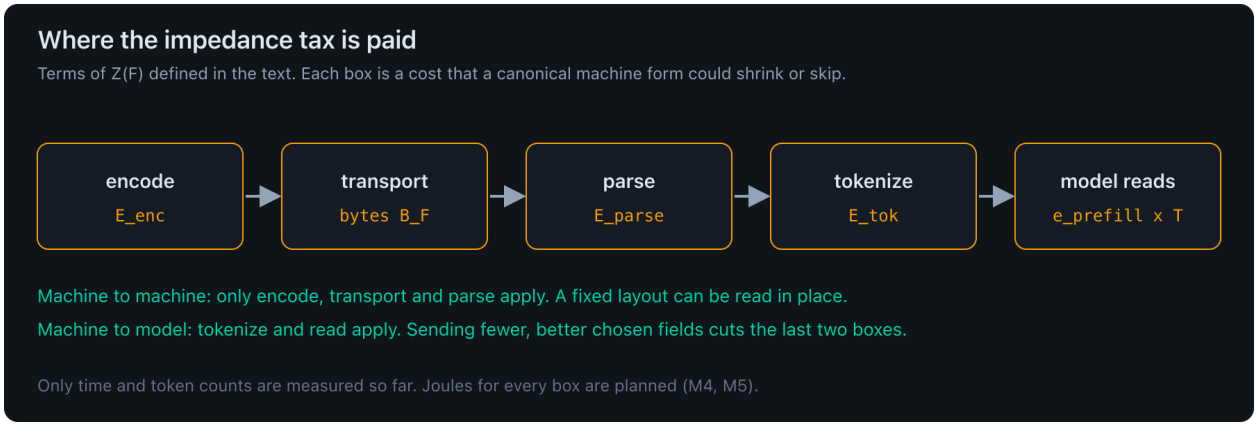

Figure 2. The terms of Z(F). With no model in the path (machine to machine) only encode, transport and parse apply. With a model reading text, tokenize and read apply.

2.1 Physical scale, and what this page does not claim

The thermodynamic floor for erasing one bit at temperature T is the Landauer limit E = k_B T ln 2. With k_B = 1.380649e-23 J/K (exact by the 2019 SI definition; the value was checked on 18 October 2026 against the NIST CODATA 2022 page for the Boltzmann constant) and T = 300 K this is **2.871e-21 J** per bit, recomputed in `scripts/verify_numbers.py`. Eight bits is 2.297e-20 J. This is a lower bound for irreversible logic, not an estimate of what parsing costs: real software pays orders of magnitude more, in memory traffic, instruction execution and cooling. This page does not use the limit to predict any number.

Energy figures for model inference come from other authors' measurements on their hardware, and are background only, not our measurement:

- Samsi et al., "From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference" (arXiv:2310.03003), define energy per output token as total GPU energy divided by decoded output tokens, and report about 3 to 4 J per output token for a 65-billion-parameter LLaMA model at a maximum generation length of 512, measured with NVIDIA DCGM on their MIT SuperCloud setup (V100 GPUs for the 8, 16 and 32 shard runs). That paper does not separate prefill from decode energy.
- Luccioni, Jernite and Strubell, "Power Hungry Processing: Watts Driving the Cost of AI Deployment?" (ACM FAccT 2024, DOI 10.1145/3630106.3658542), measured energy per 1,000 inferences by task with CodeCarbon on an 8-GPU A100 node; their Table 2 reports mean 0.002 kWh for text classification and 0.047 kWh for text generation. These numbers were checked against the ACM Corrected Version of Record V1.1 (6 November 2024), which gives the same Table 2 values and the same setup (8 A100 80GB GPUs on AWS, CodeCarbon). Cite V1.1: the paper notes that an earlier version used 0.012 kWh in one comparison. The rest of the paper was read only in part.

A per-word DRAM-read energy figure (a widely quoted 640 pJ per 32-bit read) is **not used on this page**. Its attributed source, Horowitz (2014), was read: the paper gives about 1 to 2 nJ per DRAM access and the plenary slides 1.3 to 2.6 nJ per 64-bit access; the 640 pJ figure appears only in later slides that cite it. Section 9 records this status.

3. What was measured

Initial parse-time and token-count baselines ran on one virtualised Intel Xeon machine (8 vCPU, Linux 6.12, Python 3.13.5), which exposes no RAPL energy counters (`/sys/class/powercap` is absent), followed by Apple M5 Max

energy campaigns reported in sections 5b and 5c. The corpus is synthetic and seeded (`scripts/gen_corpus.py`, seed 20261016): 20,000 records, each with an id, a three-word name, a float score, a boolean, three tags and a 31-bit integer.

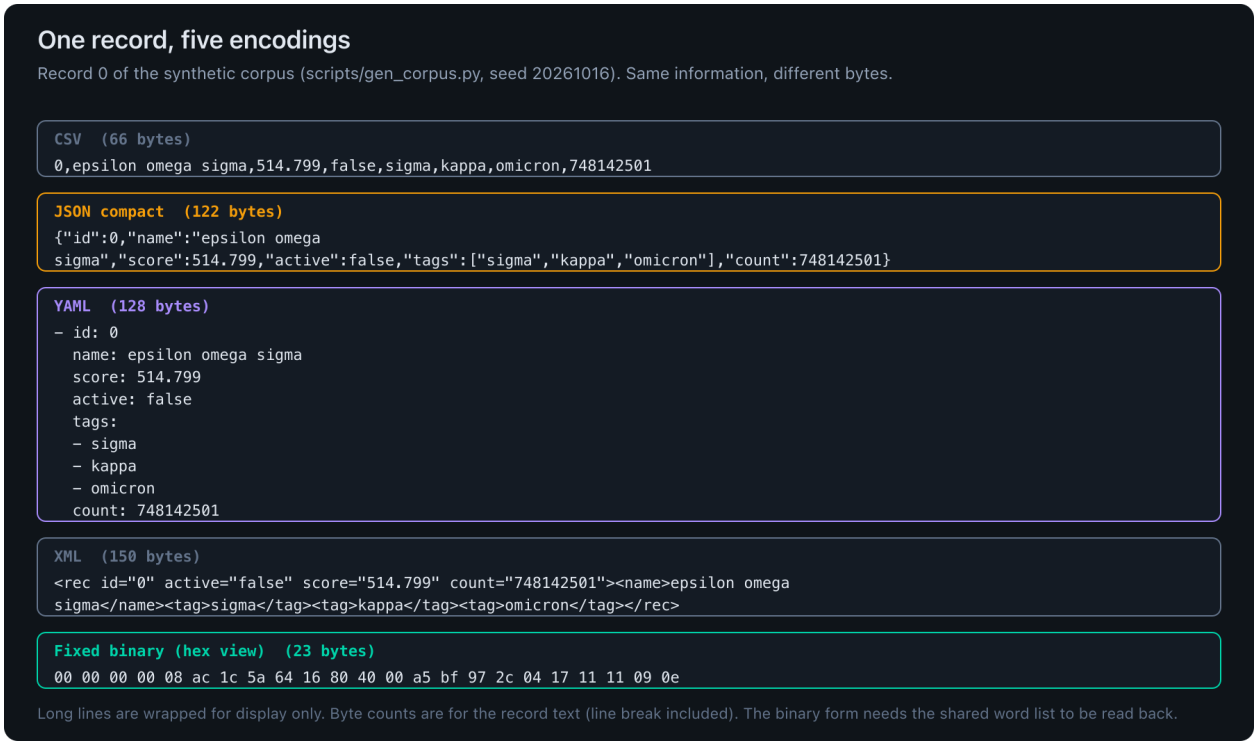

Figure 3. Record 0 in CSV, compact JSON, YAML, XML and a 23-byte fixed binary layout. The binary layout stores each word as a one-byte index into a shared list of 24 words, so reading it back needs that list. Byte counts of the text forms include the line break.

3.1 Parse time

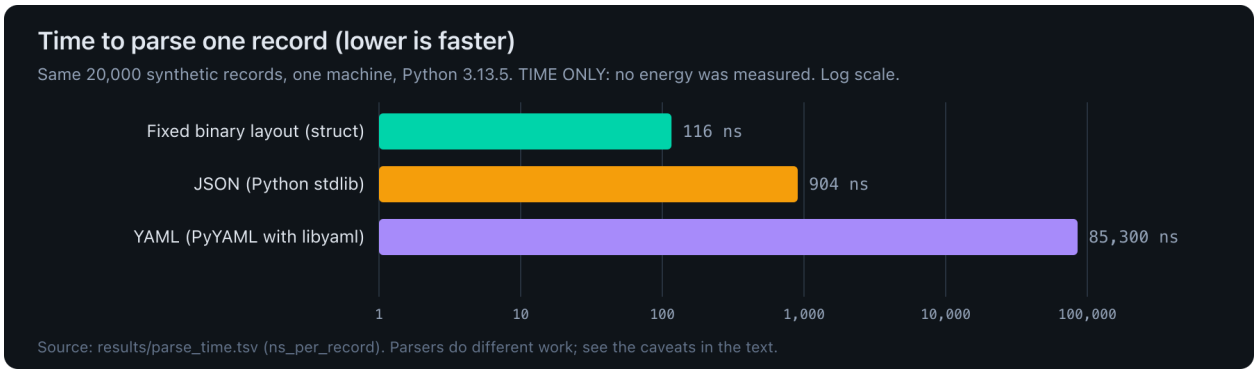

Figure 4. Minimum of 9 repeats over 20,000 records on the Linux Xeon baseline. Source: `/living/ai-reads-human/data/parse_time.tsv`, column `ns_per_record`. Log scale.

Format and parser	ns per record (Xeon)	ratio to fixed binary
Fixed binary, `struct.iter_unpack`	116	1.0
JSON, Python standard library	904	7.8
YAML, PyYAML with libyaml	85,300	735.3

YAML took 94.4 times as long as stdlib JSON on the same records on the Xeon host (and `22,404 ns` vs `570 ns` = `39.3×` for C `libyaml`, or `161,227 ns` = `282.9×` for pure-Python YAML, on Apple M5 Max in section 5c). The full Xeon table of every format and parser tried is in Figure 5.

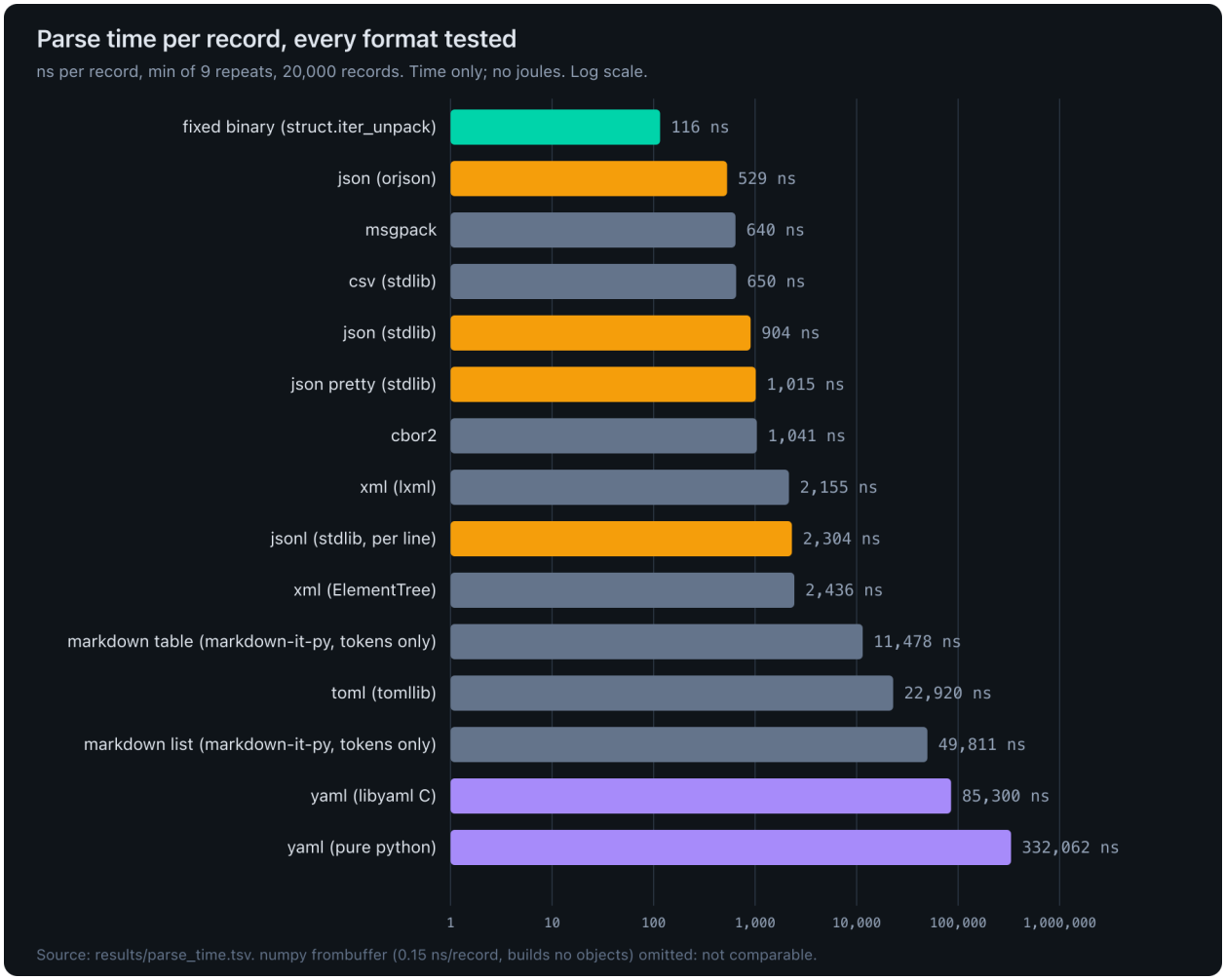

Figure 5. All rows of `data/parse_time.tsv` *except the zero-copy numpy row, which builds no objects and is not comparable.*

Limits of this measurement. (a) Parsers do different work: the fixed binary reader returns tuples, JSON and YAML build typed Python objects, CSV returns strings, and the XML and Markdown rows build trees or token streams without converting values, so the ratios compare end-to-end library behaviour, not the intrinsic cost of each format. (b) One language runtime (CPython 3.13). (c) The corpus is synthetic and identifier-light. (d) Time alone is not energy: section 5c therefore measures per-process kernel `ri_energy_nj` (`reported_j`) directly across all 34 parse and tokenize workloads.

3.2 Tokens

A language model reads tokens, not bytes. The same records, counted with the `cl100k_base` tokenizer on the initial 2,000 records of each view (and verified across all 20,000 records in section 5c):

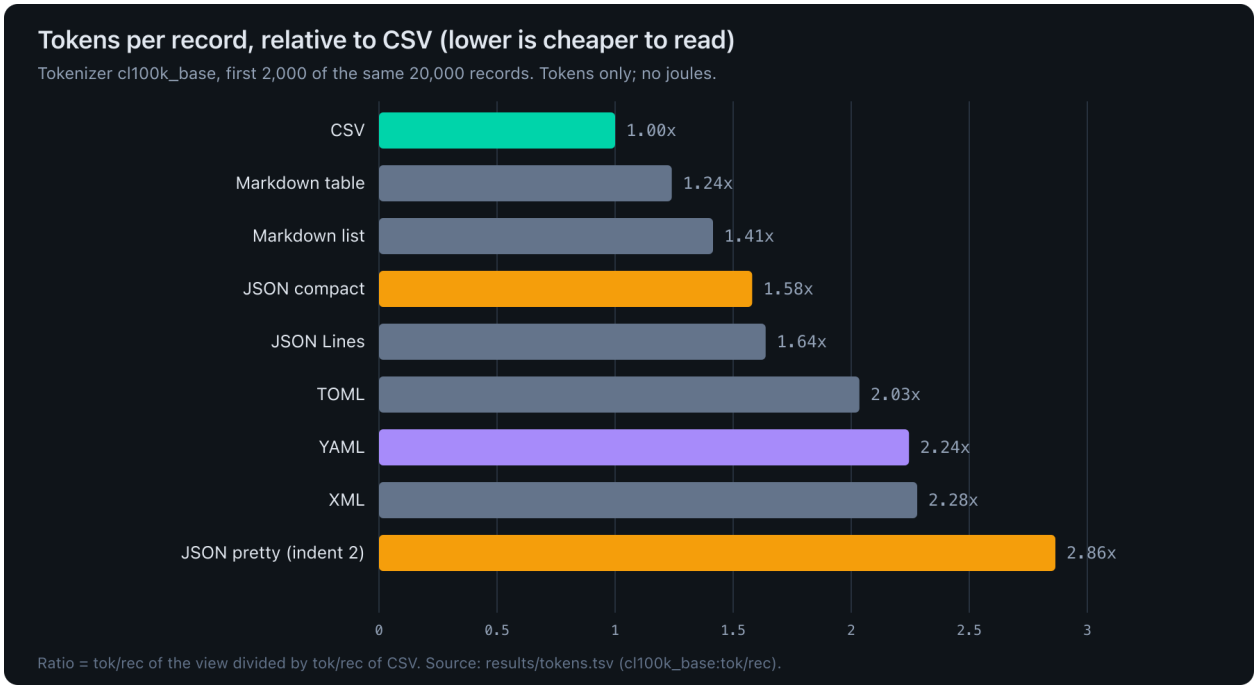

Figure 6. κ *= tokens per record of the view divided by tokens per record of CSV. Source:* `/living/ai-reads-human/data/tokens.tsv`.

View	tokens per record	κ (relative to CSV)
CSV	22.9	1.00
JSON, compact	36.2	1.58
YAML	51.4	2.24
JSON, pretty-printed (indent 2)	65.6	2.86

The identity in section 2 holds on these data: for compact JSON, $\beta = 1.912$ and d_CSV/d_JSON $= 0.826$, giving $\kappa = 1.58$; for YAML, $2.028 \times 1.107 = 2.24$; for pretty JSON, $3.056 \times 0.936 = 2.86$. Compact JSON costs more tokens mainly because it has about 1.9 times the bytes of CSV, and pretty JSON mostly for the same reason plus indentation. YAML has both more bytes and shorter tokens.

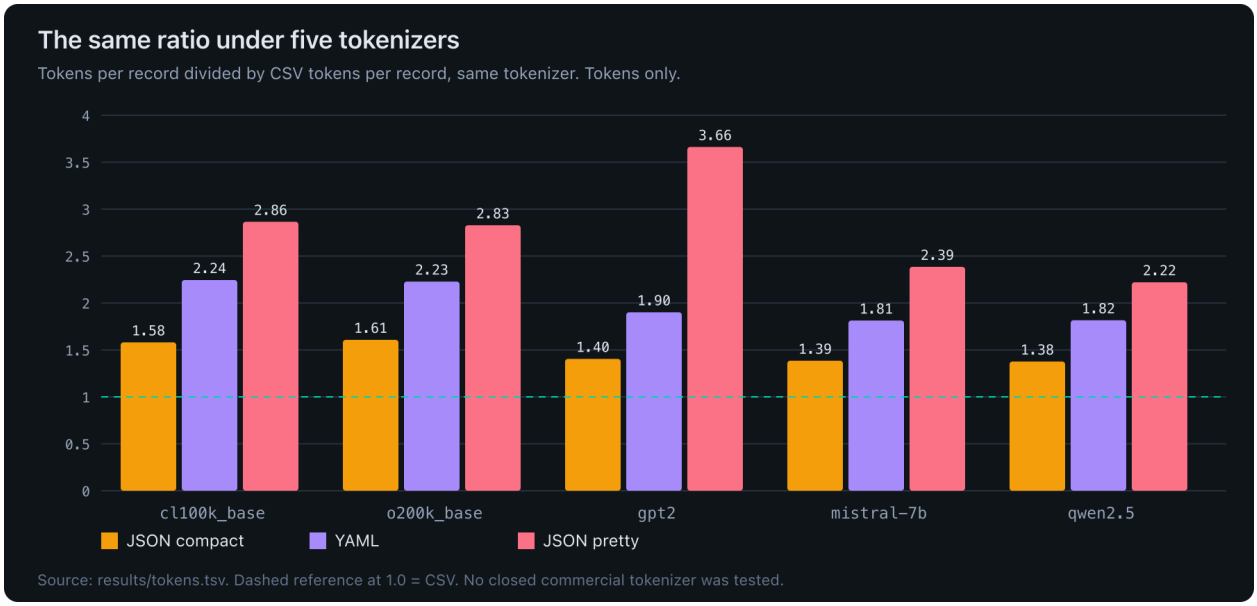

Figure 7. The same ratio under `cl100k_base`, `o200k_base`, GPT-2, Mistral-7B-v0.1 and Qwen2.5-0.5B tokenizers. The ordering holds in all five; the size of the gap does not (pretty JSON is 3.66 times CSV under GPT-2 and 2.22 times under Qwen2.5). No closed commercial tokenizer was tested.

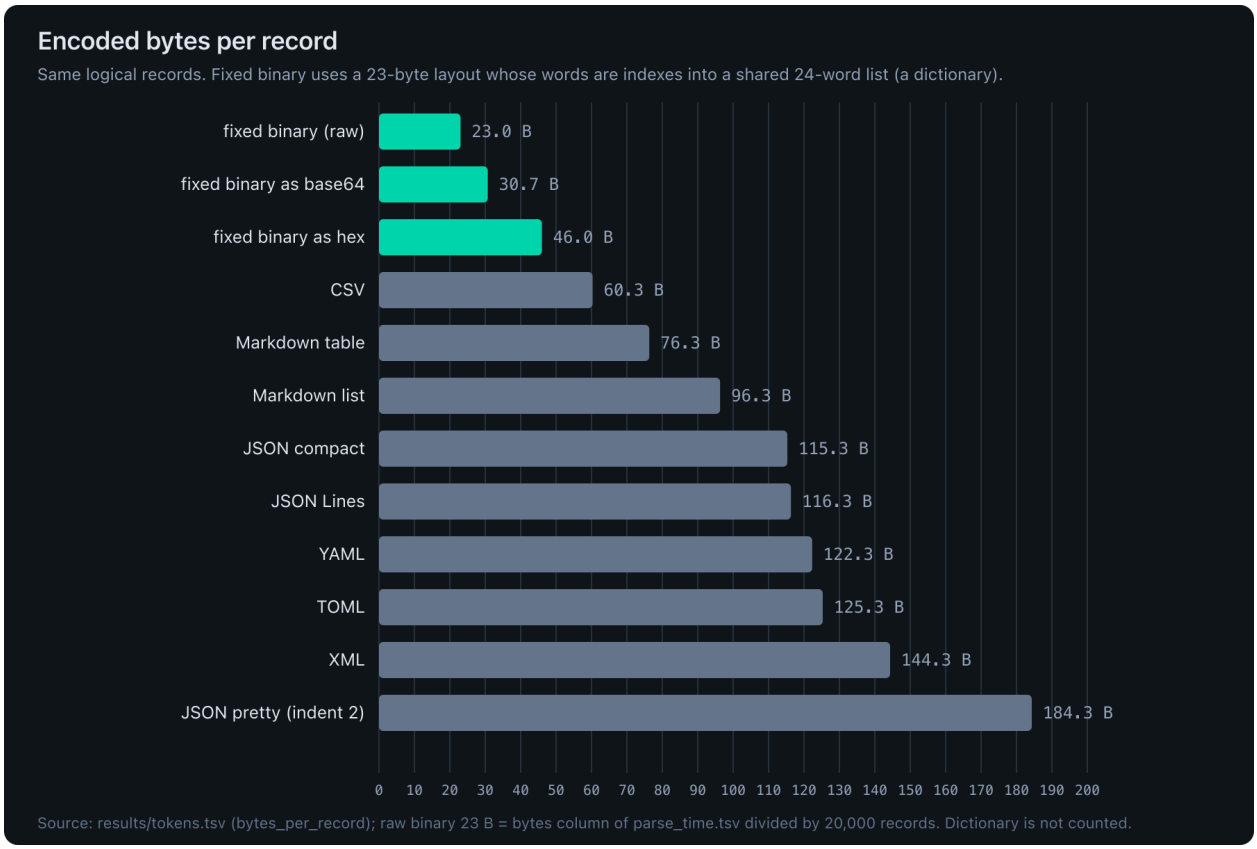

Figure 8. Bytes per record. The 23-byte binary figure does not count the shared word list; any honest size claim for a binary form must name the list (or dictionary) it depends on, by version and hash.

A negative result worth stating. Writing the binary layout as text for a model does not remove the tax: as base64 it was 20.3 tokens per record ($\kappa = 0.89$) and as hex 22.7 ($\kappa = 0.99$), about the same as CSV, because a text model needs a text view. The saving for a model comes from sending fewer, better chosen fields, not from changing the encoding of what is sent. That claim is Planned (M7), not measured.

4. Why the hazards come with the readability

Formats designed for people and for loosely coupled programs let one input mean different things to different readers. The table lists hazards found in the read specifications, and in tests run on named library versions (Python 3.13.5, Node v20.19.2, jq 1.7, PyYAML 6.0.2, lxml 6.1.3). Test lines (T, U, X, Y, C codes) are in `tests-formats/RESULTS.txt` and describe only those versions. "Spec" means the text of the cited document.

Class	Format	Hazard	Evidence
Ambiguity	JSON	Duplicate member names give "unpredictable" behaviour; implementations report the last, all, or fail. Three tools tested kept the last value.	RFC 8259 section 4; T1
Ambiguity	Markdown	The original description does not specify syntax unambiguously; dialects diverged.	CommonMark 0.31.2 section 1.2
Parser differential	JSON numbers	9007199254740993 is exact in Python and rounds to …992 in Node.	RFC 8259 section 6; T2 and N2
Parser differential	JSON	Python's `json` accepts bare `NaN` and `Infinity`, which the spec says are not permitted.	RFC 8259 section 6; T6
Non-canonical forms	JSON	`1`, `1.0`, `1e0`, `10E-1`, `0.1e1` are one number written five ways, and two Python types.	U3

Class	Format	Hazard	Evidence
Non-canonical forms	JSON	Member order does not change the meaning but changes the bytes and any hash of them.	T8
Ordering	JSON canonicalization	Sorting by UTF-16 code units differs from code point order for U+1F600 versus U+FFEE.	RFC 8785 section 3.2.3; U2
Non-canonical forms	CBOR	Deterministic encoding is an optional profile that protocols may adopt; it is not the default.	RFC 8949 section 4.2
Non-canonical forms	Protocol Buffers	"Do not assume the byte output of a serialized message is stable."	Protobuf encoding guide, "Implications"
Number hazards	JSON	Implementations may limit range and precision; the exactly representable integer range is $\pm(2^{53})-1$.	RFC 8259 sections 6 and 9; RFC 7493 section 2.2
Number hazards	JSON	`1e400` becomes `inf` in both Python and Node; `-0` is handled differently.	T3, N3, T4, N4
Number hazards	YAML 1.1 resolver	In PyYAML, `010` is 8, `12:30:45` is 45045 and `country: NO` is `{country: False}`.	Y rows
Number hazards	TOML	The spec says an integer that cannot be stored losslessly is an error; Python's `tomllib` accepted 2^{63}.	TOML 1.0.0; T rows
Unicode	JSON	`"\uDEAD"` (an unpaired surrogate) is allowed by the grammar; it parses, then fails on UTF-8 encode.	RFC 8259 section 8.2; T5
Unicode	JSON	NFC and NFD forms of the same text are distinct keys.	U1
Injection	CSV	A cell starting with `=` is plain data to the parser and a formula to some spreadsheet programs.	OWASP CSV Injection; C row
Injection	XML	With entity resolution on, lxml read a local file through an external entity.	X rows
Resource blow-up	YAML	166 bytes of aliases expanded to 59,049 logical leaves.	Y alias test
Resource blow-up	XML	202 bytes of internal entities expanded to 10,000 characters (49.5 times).	X rows
Resource blow-up	JSON	Depth limits are implementation-defined; Python raised `RecursionError` at 100,000 nesting.	RFC 8259 section 9; T9
Totality	any parser	Library exceptions are control flow outside the function's return type.	T5, T9

None of these is a reason to avoid the formats for people. Each is a reason not to use the same text as the single authority between machines.

5. Planned: measuring energy

The measurement roadmap separates completed parse/tokenize `reported_j` counters from open GPU inference and external hardware power probes:

ID	Measurement	Status
M1, M2, M3	Parse time, tokens, tokenizer throughput	**Done** (`data/parse_time.tsv`, `data/tokens.tsv`, `data/tokenize_time.tsv`)
M5	Parse and `tiktoken` energy across all 34 workloads via dual-channel Apple M5 Max `reported_j` (`L3` per-process `ri_energy_nj` + `L2` whole-chip `IOReport` with Falsifier 6 idle-drift gating)	**Done for `reported_j`** (section 5c; `data/energy_reported_j.tsv`; `measured_j = None`, `board_metered = false`)
M4	Prefill energy and decode energy per token against context length, same model, GPU with a hardware energy counter or external rail/wall probe	Planned (needs dedicated GPU counter or inline power probe)
M6	A second corpus: natural-language documentation and logs against identifier-heavy telemetry	Planned
M7	Task level: tokens an agent needs to answer a fixed question set from a projection of the fields it needs versus the full document	Planned
M8	Other parsers per format (simdjson, rapidyaml, libxml2 C, pulldown-cmark) to bound language-runtime effects	Planned

External wall-plug or inline USB-C power-probe readings (`measured_j`) remain uncollected; section 5c reports operating-system/on-chip `reported_j` counters strictly.

5b. Measured energy: no net effect resolved

First whole-chip subtraction attempt, run on 2026-10-18 on two Apple M5 Max laptops (48 GB on battery power, 128 GB on AC), macOS 27.0.1, Python 3.13.16, power counter macmon 0.10.0 (48 GB) and 0.9.0 (128 GB), reading whole-SoC `IOReport` and SMC sensors (`all_power` and `sys_power`). Nine parse workloads on the same 20,000 records, each looped for 10 s on one thread, 10 interleaved rounds, with a 10 s idle window before and after every round. Files: `/living/ai-reads-human/data/energy_joules.tsv` and raw samples in `/living/ai-reads-human/data/energy_raw_m5max48/` and `energy_raw_m5max128/`.

Result of the unisolated whole-chip attempt: net-above-idle package subtraction could not resolve single-thread parsing on a busy host. On both laptops (one-minute load average 41 to 68), the pre/post idle baseline moved by 2 to 7 W between windows while one parsing thread added only 1 to 5 W to a chip drawing 18 to 36 W, leaving every net mean within 1.5 standard errors of zero (several negative). That failure motivated the dual-channel protocol in section 5c: pairing whole-chip `IOReport` (L2, gated by Falsifier 6 so drifted net subtractions are automatically rejected) with macOS kernel per-PID `proc_pid_rusage(RUSAGE_INFO_V6).ri_energy_nj` (L3), which attributes CPU energy strictly to the dedicated worker subprocess executing each workload.

5c. Dual-channel Apple M5 Max `reported_j` across all 34 parse and tokenize workloads

Executed on 2026-11-22 (`16:44:52` to `16:53:22 UTC`) on Apple M5 Max (`128 GB RAM`, `macOS 27.2.1 arm64`, `Python 3.13.14`, `macmon 0.12.0`), seeded corpus `20261016` (`20,000` records across 12 SHA-256-pinned files), `5` interleaved shuffled rounds × `34` workloads = `170` dedicated worker subprocess windows. Files: `/living/ai-reads-human/data/energy_reported_j.tsv` and `/living/ai-reads-human/data/energy_reported_j_summary.json`. Interactive explorer: `/living/ai-reads-human/`.

Telemetry channels and Falsifier 6 demonstration

Every measurement window recorded two rootless telemetry channels simultaneously while keeping `measured_j = None` and `board_metered = false`:

1. **Per-process L3 `reported_j` (shared-host label `D`, co-tenant isolated)**: `proc_pid_rusage(worker_pid, RUSAGE_INFO_V6).ri_energy_nj`, integrated by the XNU kernel strictly over the threads scheduled inside the dedicated worker subprocess for that single format/tokenizer pass.
2. **Whole-SoC L2 `reported_j` (shared-host label `A2`, Falsifier 6 gated)**: `macmon pipe` sampling Apple Silicon `IOReport` `chip_power` alongside `0.8 s` pre- and post-window idle baselines. Under **Falsifier 6**, any window whose pre/post idle baseline drifts by `> 5.0%` has its `derived_marginal_j` automatically invalidated rather than reported. On this shared host (mean 1-minute load average `12.92`, idle chip draw `30.5–35.4 W`), Falsifier 6 invalidated **`109 of 170 windows (64.1%)`** across the 34 workloads (`61 / 170` valid, `0 / 34` workloads passing all `5 / 5` replicates) — reproducing section 5b's diagnosis quantitatively while L3 `ri_energy_nj` cleanly resolved every single workload.

Parse-stage L3 `reported_j` (16 workloads, 20,000 records)

Format and parser	Corpus bytes	Wall ms / 20k (mean ± SD)	L3 nJ / record (mean ± SD)	L3 nJ / byte	Instructions / rec	L3 energy ratio vs binary
Fixed binary (`numpy.frombuffer`, zero-copy view)	460,000	0.0002 ± 0.0000	0.04 ± 0.01	0.002	0.3	0.00× (view only)
Fixed binary (`struct.iter_unpack`, reference)	460,000	1.359 ± 0.351	280.36 ± 111.38	12.19	1,397.4	1.00×
JSON (`orjson`, C/Rust SIMD)	2,325,428	7.053 ± 1.359	1,405.93 ± 372.34	12.09	6,187.0	5.01×
CSV (`csv`, Python stdlib)	1,205,469	8.804 ± 1.309	1,798.22 ± 435.28	29.83	8,812.4	6.41×
MessagePack (`msgpack`)	1,660,773	10.817 ± 3.321	1,802.46 ± 593.84	21.71	9,593.3	6.43×
JSON (`json.loads`, Python stdlib)	2,325,428	11.396 ± 1.219	2,212.12 ± 409.29	19.03	12,700.6	7.89×
JSON pretty (`json.loads`, indent 2)	3,685,429	11.716 ± 2.225	2,648.45 ± 669.32	14.37	13,433.3	9.45×
CBOR (`cbor2`)	1,660,877	15.286 ± 1.250	2,719.58 ± 365.78	32.75	14,514.3	9.70×

Format and parser	Corpus bytes	Wall ms / 20k (mean ± SD)	L3 nJ / record (mean ± SD)	L3 nJ / byte	Instructions / rec	L3 energy ratio vs binary
XML (lxml C library)	2,885,487	22.204 ± 2.392	3,800.46 ± 331.49	26.34	19,521.4	13.56×
XML (xml.etree.ElementTree)	2,885,487	26.354 ± 0.750	6,029.30 ± 370.65	41.79	27,495.7	21.51×
JSON Lines (json.loads per line)	2,325,427	22.729 ± 1.481	6,130.27 ± 946.08	52.72	27,414.8	21.87×
Markdown table (markdown-it-py, tokens)	1,525,499	115.160 ± 7.576	22,824.50 ± 2,464.64	299.24	115,255.0	81.41×
TOML (tomllib, Python stdlib)	2,505,426	298.582 ± 60.674	57,803.43 ± 16,269.64	461.43	260,140.9	206.17×
YAML (PyYAML with C libyaml)	2,445,427	448.071 ± 95.009	69,922.12 ± 13,405.09	571.86	371,347.0	249.40×
Markdown list (markdown-it-py, tokens)	1,925,427	417.700 ± 54.289	99,595.04 ± 17,044.11	1,034.52	377,878.0	355.23×
YAML (PyYAML, pure Python)	2,445,427	3,224.548 ± 375.725	575,799.27 ± 56,674.36	4,709.19	2,706,372.9	2,053.75×

Tokenization-stage L3 reported_j (18 tiktoken workloads, 20,000 records)

When a language model ingests text rather than structured memory, byte-pair tokenization (E_tok(F,k) in section 2) dominates CPU decode cost on compact formats:

Tokenizer & view	Tokens / 20k	Wall ms / 20k (mean ± SD)	L3 nJ / record (mean ± SD)	L3 nJ / token	Ratio vs CSV (same tok)	Ratio vs fixed binary parse
cl100k_base · CSV (recs.csv)	459,916	73.442 ± 20.454	16,384.84 ± 4,362.40	712.51	1.000×	58.44×
cl100k_base · Markdown table	569,973	105.364 ± 18.819	23,385.73 ± 5,597.57	820.59	1.427×	83.41×
cl100k_base · Markdown list	649,947	105.199 ± 11.723	26,749.70 ± 4,445.05	823.13	1.633×	95.41×
cl100k_base · JSONL (recs.jsonl)	753,109	154.114 ± 43.130	26,983.02 ± 9,347.95	716.58	1.647×	96.24×
cl100k_base · JSON compact (recs.json)	733,111	124.295 ± 7.051	27,679.19 ± 2,697.37	755.12	1.689×	98.73× (12.51× json.loads)
cl100k_base · XML (recs.xml)	1,046,596	171.844 ± 21.381	36,730.51 ± 6,968.26	701.90	2.242×	131.01×
cl100k_base · TOML (recs.toml)	933,109	178.111 ± 23.760	38,499.73 ± 6,633.02	825.19	2.350×	137.32×
cl100k_base · YAML (recs.yaml)	1,029,947	208.434 ± 33.837	51,880.14 ± 12,381.40	1,007.43	3.166×	185.05×
cl100k_base · JSON pretty (recs_pretty.json)	1,313,111	266.827 ± 21.754	70,590.45 ± 10,239.93	1,075.16	4.308×	251.78× (26.65× pretty json.loads)
o200k_base · CSV (recs.csv)	464,995	51.864 ± 5.906	9,570.91 ± 1,012.53	411.66	1.000×	34.14×
o200k_base · JSON compact (recs.json)	754,794	93.277 ± 8.521	15,125.65 ± 1,287.77	400.79	1.580×	53.95× (6.84× json.loads)
o200k_base · Markdown table / list	574,818 / 654,792	68.091 / 68.666	15,693.90 / 17,362.58	546.05 / 530.32	1.640× / 1.814×	55.98× / 61.93×
o200k_base · JSONL / XML / TOML	774.8k / 1,054.8k / 934.8k	83.02 / 114.79 / 106.68	21,530.58 / 22,997.48 / 25,307.32	555.78 / 436.05 / 541.45	2.250×–2.644×	76.79×–90.27×
o200k_base · YAML (recs.yaml)	1,034,792	141.338 ± 24.980	29,401.44 ± 7,308.15	568.26	3.072×	104.87×
o200k_base · JSON pretty (recs_pretty.json)	1,314,794	193.648 ± 18.210	37,070.18 ± 3,651.35	563.89	3.873×	132.22×

Three findings emerge from the 170-window campaign:

1. **Tokenizing JSON costs an order of magnitude more CPU reported_j than parsing it**: tiktoken (cl100k_base, implemented in compiled Rust) spends 27,679.2 nJ/record (0.554 J per 20,000 records, 129,192 instructions/record) on compact JSON — **12.51× Python's json.loads() (2,212.1 nJ/record)**, **19.69× orjson.loads() (1,405.9 nJ/record)**, and **98.73× struct.iter_unpack (280.4 nJ/record)**. Indenting JSON (indent=2) inflates cl100k_base tokenization energy a further 2.55× to 70,590.5 nJ/record (4.31× CSV tokenization) simply to scan whitespace runs.
2. **Parse energy ratios closely track instruction ratios, with minor effective-wattage variation (3.28–5.65 W per worker PID)**: C libyaml draws 3.31 W effective worker power (249.4× binary reported_j

vs 329.8× wall time), whereas per-line jsonl and pretty-JSON tiktoken draw 5.37–5.46 W (21.87× and 251.78× reported_j vs 16.73× and 196.37× wall time), confirming why direct per-process ri_energy_nj counters are preferable to multiplying wall time by a constant wattage guess.

3. **Provenance boundary**: ri_energy_nj is an operating-system/firmware counter (reported_j, telemetry level L3), not an external hardware shunt or wall-plug power analyzer (measured_j = None, board_metered = false), and still covers host CPU/memory work prior to neural-network forward passes (e_prefill · T_{F,k} in M4).

6. Where a canonical machine form fits

One response to the tax is to keep one canonical machine form for storage and exchange and emit human text only when a person asks to read it. Several projects have taken that route for parts of the problem: deterministic CBOR, canonical JSON, schema-bound binary formats. The properties such a form would need are testable: one valid byte string per value, a declared version, closed schema, bounds on size and depth stated before decoding, and a size claim that names the dictionary it depends on.

6.1 An open design question

Which canonical exchange form is best is not decided here. Canonical JSON (RFC 8785) keeps a text form but needs rules for number formatting and key order; deterministic CBOR (RFC 8949 section 4.2) is binary (2,719.6 nJ/record in Python cbor2, 9.70× fixed binary) but leaves choices to each protocol. A length-prefixed or fixed-stride binary form (280.4 nJ/record in struct.iter_unpack, or 0.04 nJ/record for a zero-copy buffer view) removes almost all parsing work between machines, whereas a text model still requires a projected text view (16,384.8 nJ/record for CSV vs 70,590.5 nJ/record for pretty JSON under cl100k_base). Which end-to-end trade-off wins once GPU model prefill (M4) and field-projection tasks (M7) are included remains open.

7. Reproduce this page

Everything needed is under /living/ai-reads-human/ in this site:

- data/parse_time.tsv, data/tokens.tsv, data/tokenize_time.tsv, data/ENV.txt, data/SUMMARY.txt: the Linux Xeon time and token baselines.
- data/energy_reported_j.tsv and data/energy_reported_j_summary.json: the 34-workload, 170-window Apple M5 Max dual-channel reported_j (L3 ri_energy_nj + L2 IOReport Falsifier 6) dataset (data/energy_joules.tsv retains the earlier October 2026 unisolated whole-chip null run).
- scripts/gen_corpus.py (seeded corpus), bench_parse.py, bench_tokens.py, bench_tokenize_time.py, summarize.py: the measurements.
- scripts/make_charts.py: rebuilds every chart from the result files, and asserts the headline values.
- scripts/verify_numbers.py: recomputes every number quoted in the text; exits non-zero on a mismatch.
- tests-formats/: the scripts behind the hazard test lines.

```
python3 -m venv --system-site-packages venv
venv/bin/pip install tiktoken tokenizers msgpack cbor2 orjson ruamel.yaml markdown-it-py
    psutil
venv/bin/python scripts/gen_corpus.py 20000
venv/bin/python scripts/bench_parse.py 9 > results/parse_time.tsv
venv/bin/python scripts/bench_tokens.py 2000 > results/tokens.tsv
venv/bin/python scripts/make_charts.py charts && venv/bin/python scripts/verify_numbers.py
```

The tokenizer vocabularies for GPT-2, Mistral-7B-v0.1 and Qwen2.5-0.5B are downloaded from a model hub by bench_tokens.py. Timings and ri_energy_nj counters depend on the host processor and operating system; expect the multi-order-of-magnitude ordering across formats to hold while absolute nanosecond and nanojoule figures shift with hardware.

8. Threats to validity

- Kernel ri_energy_nj (L3 reported_j) is an Apple Silicon model-backed operating-system counter, not an inline DC rail or AC wall-plug meter (measured_j = None, board_metered = false), and GPU model pre-

fill/decode joules (M4) are not yet measured here.
- On shared hosts, whole-SoC IOReport (L2) net-above-idle subtraction is vulnerable to co-tenant drift (109 / 170 windows invalidated by Falsifier 6 in section 5c; null result in section 5b).
- Python object-allocation overhead contributes to several parse rows (compiled C/Rust parsers in M8 will bound runtime overhead; note orjson and tiktoken already execute compiled native code).
- One synthetic, identifier-light corpus (seed = 20261016); natural-language and telemetry logs remain untested (M6).
- Tokenizers tested are open ones; closed commercial tokenizers are untested.
- The 23-byte binary layout depends on a shared 24-word dictionary (SHA-256 pinned in data/energy_reported_j_summary.json) that per-record byte figures do not repeat.

9. Read ledger and references

Status per source is in the read ledger (FULL, PARTIAL, UNAVAILABLE). Cited: RFC 8259, RFC 7493, RFC 8785, RFC 8949, RFC 4180, CommonMark 0.31.2, YAML 1.2.2 (section 1.1), TOML 1.0.0, Protocol Buffers encoding guide, OWASP CSV Injection, tex.web (header only, PARTIAL), Samsi et al. arXiv:2310.03003, Luccioni et al. DOI 10.1145/3630106.3658542. HTML sources were read as tag-stripped text, so figures in them were not read.

Not read (paywalled or unavailable), listed with identifiers and not cited for any claim:

Item	Identifier
ISO 32000-1:2008, Portable Document Format	ISO catalogue (paywalled)
ISO/IEC 29500, Office Open XML	ISO/IEC 29500 (free equivalent ECMA-376 only partly read)
ISO/IEC 26300, OpenDocument	ISO/IEC 26300 (free OASIS ODF 1.3 Part 1 read)
IEEE 754-2019, Floating-Point Arithmetic	DOI 10.1109/IEEESTD.2019.8766229
Sennrich, Haddow, Birch, "Neural Machine Translation of Rare Words with Subword Units"	DOI 10.18653/v1/P16-1162
The Unicode Standard and UAX #15	unicode.org (not fetched)
ECMA-262 (number serialisation referenced by RFC 8785)	not fetched
The TeXbook, printed edition	ISBN 978-0-201-13447-6
Horowitz, "Computing's energy problem", attributed source of the quoted 640 pJ per 32-bit DRAM read	**read; 640 pJ not found in the paper or plenary slides (they give 1 to 2 nJ, and 1.3 to 2.6 nJ per 64-bit access); not used**
Strubell et al. 2019; Patterson et al. 2021 and 2022; Wu et al. 2021	cited only inside the Samsi and Luccioni papers; not read, no claim made

10. What will be added

Planned expansions, in no fixed order: GPU model prefill and decode energy per token (M4) and external inline wall-meter validation (measured_j); compiled C/Rust native parsers (M8); a natural-language and structured-log corpus (M6); per-task token counts for agent field-projection workloads (M7); more formats from the ledger (PDF 2.0, C2PA, Markdown dialects, LaTeX) with their hazards; and a worked comparison of canonical binary projections for models.

© 2026 Open Interface Engineering, Inc. Text, figures, data and scripts: MIT OR Apache-2.0, at your option.