

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

David Charlot, Open Interface Engineering

Work in progress · OpenIE research.openie.dev

<https://research.openie.dev/papers/ai-reads-human/> · PDF <https://research.openie.dev/pdfs/ai-reads-human.pdf>

Measurement. Research topic, work in progress, not a journal final. Time and token counts are measured on one virtualised machine. No joules were measured; 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 and time today, and how the energy cost will be measured.

Version 0.2 (work in progress), 18 October 2026. 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. We name the extra work this causes the impedance tax, define it with units, and measure two parts of it on a reproducible corpus: parse time and token count. A first energy attempt on two laptops (section 5b) could not resolve any net energy of parsing, so no joule claim is made; this page says what was tried and what is planned.

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 test data, the same records took about 8 times longer to read as JSON (Python's built-in reader), and about 735 times longer as YAML, than as a plain fixed binary layout. Read by an AI model, YAML used about 2.2 times as many tokens as CSV.
3. We measured time and token counts. We also tried to measure energy on two laptops, but the signal was too small to separate from background power, so this page makes no claim about how many joules are saved.

Status: work in progress, and it will grow. Sections marked *Planned* are not done. Every number on this page is recomputed from files you can download below. Nothing here is a claim about energy savings; the one energy attempt (section 5b) is a null result.

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), we record 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
$\rho_{\{F,k\}} = B_F / T_{\{F,k\}}$	bytes per token	byte/token
$t_{\text{parse}}(F)$	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 $P_{\text{avg}} \cdot t$	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 :

$$Z(F) = E_{\text{enc}}(F) + E_{\text{parse}}(F) + E_{\text{tok}}(F,k) + e_{\text{prefill}} \cdot T_{\{F,k\}} + E_{\text{ctx}}(F)$$

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

$E_{\text{ctx}}(F)$ is the extra decode cost that a longer context causes; its coefficient depends on the model and is a measurement target (M4 below).

Three dimensionless ratios can be measured today without any energy meter:

- byte ratio $\beta_F = B_F / B_{\text{ref}}$
- token ratio $\kappa_{\{F,k\}} = T_{\{F,k\}} / T_{\{\text{ref},k\}}$
- parse-time ratio $\pi_F = t_{\text{parse}}(F) / t_{\text{parse}}(\text{ref})$

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

$$\kappa_F = \beta_F \cdot (\rho_{\text{ref}} / \rho_F)$$

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

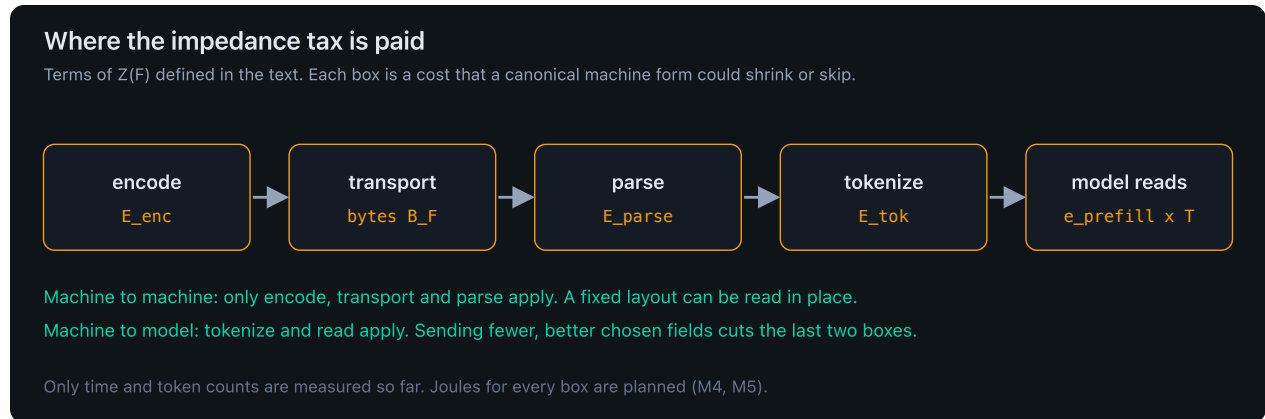


Figure 2. The terms of $Z(F)$. With no model in the path (machine to machine) only encode, transport and parse apply. With a model reading text, tokenize and read apply.

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 $k_B = 1.380649 \times 10^{-23}$ 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 2.297×10^{-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. We do not use the limit to predict any number on this page.

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. We checked these numbers 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**: we have not read its primary source, so it is listed as unverified in section 9.

3. What was measured

All measurements ran on one virtualised Intel Xeon machine (8 vCPU, Linux 6.12, Python 3.13.5). The box has no RAPL energy counters (`/sys/class/powercap` is absent) and `perf stat` reports cycles as unsupported, so **only time and token counts were measured on this machine. No joules**. Energy attempts on other machines are in section 5b. 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.

One record, five encodings

Record 0 of the synthetic corpus (`scripts/gen_corpus.py`, seed 20261016). Same information, different bytes.

CSV (66 bytes)

```
0,epsilon omega sigma,514.799,false,sigma,kappa,omicron,748142501
```

JSON compact (122 bytes)

```
{"id":0,"name":"epsilon omega sigma","score":514.799,"active":false,"tags":["sigma","kappa","omicron"],"count":748142501}
```

YAML (128 bytes)

```
- id: 0
  name: epsilon omega sigma
  score: 514.799
  active: false
  tags:
    - sigma
    - kappa
    - omicron
  count: 748142501
```

XML (150 bytes)

```
<rec id="0" active="false" score="514.799" count="748142501"><name>epsilon omega sigma</name><tag>sigma</tag><tag>kappa</tag><tag>omicron</tag></rec>
```

Fixed binary (hex view) (23 bytes)

```
00 00 00 00 08 ac 1c 5a 64 16 80 40 00 a5 bf 97 2c 04 17 11 11 09 0e
```

Long lines are wrapped for display only. Byte counts are for the record text (line break included). The binary form needs the shared word list to be read back.

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

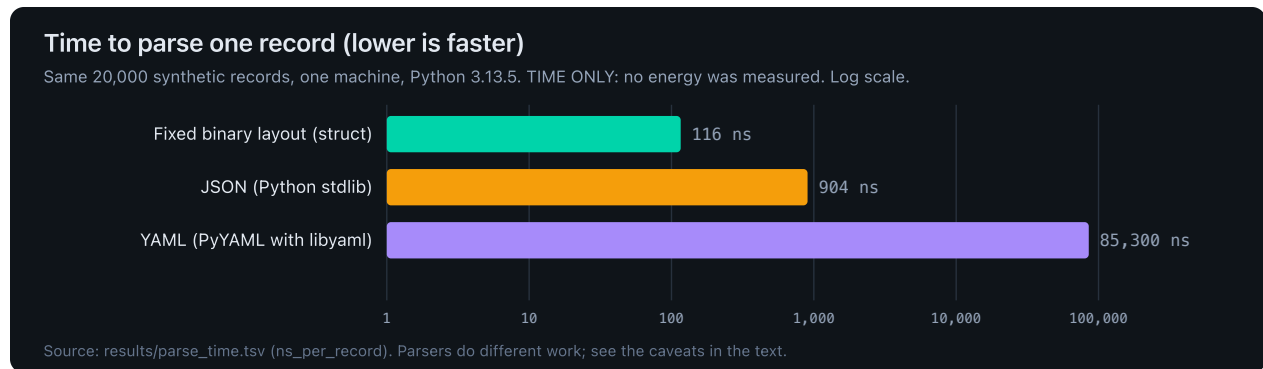


Figure 4. Minimum of 9 repeats over 20,000 records. Source: `/living/ai-reads-human/data/parse_time.tsv`, column `ns_per_record`. Log scale.

Format and parser	ns per record	ratio to fixed binary
Fixed binary, <code>struct.iter_unpack</code>	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. The full table of every format and parser tried is in Figure 5.

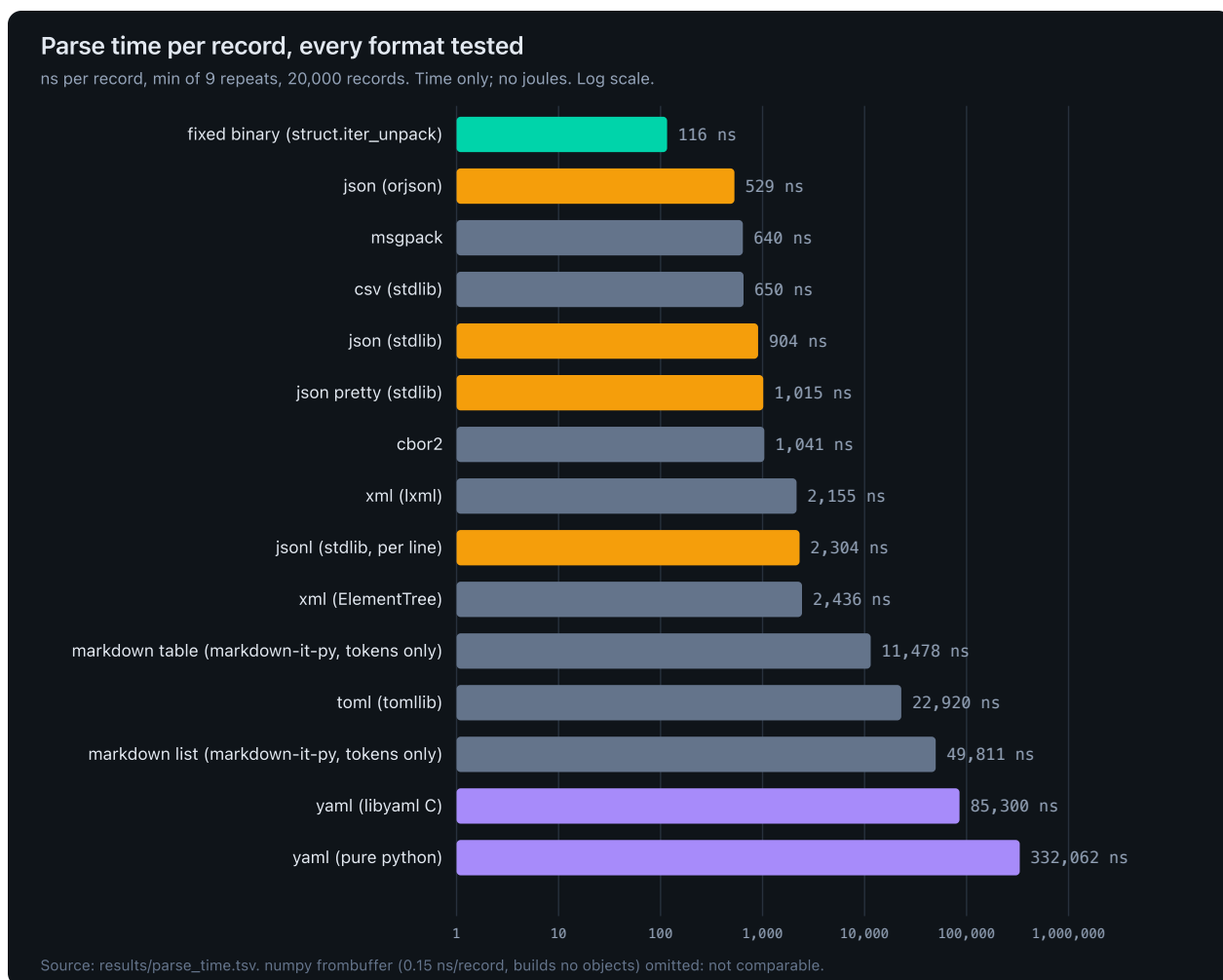


Figure 5. All rows of `data/parse_time.tsv` except the zero-copy numpy row, which builds no objects and is not comparable.

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 machine and one language runtime. (c) The corpus is synthetic and identifier-light. (d) Time is not energy: a faster operation can draw more power, so energy ratios need not equal time ratios. The Planned section below says how this will be tested.

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:

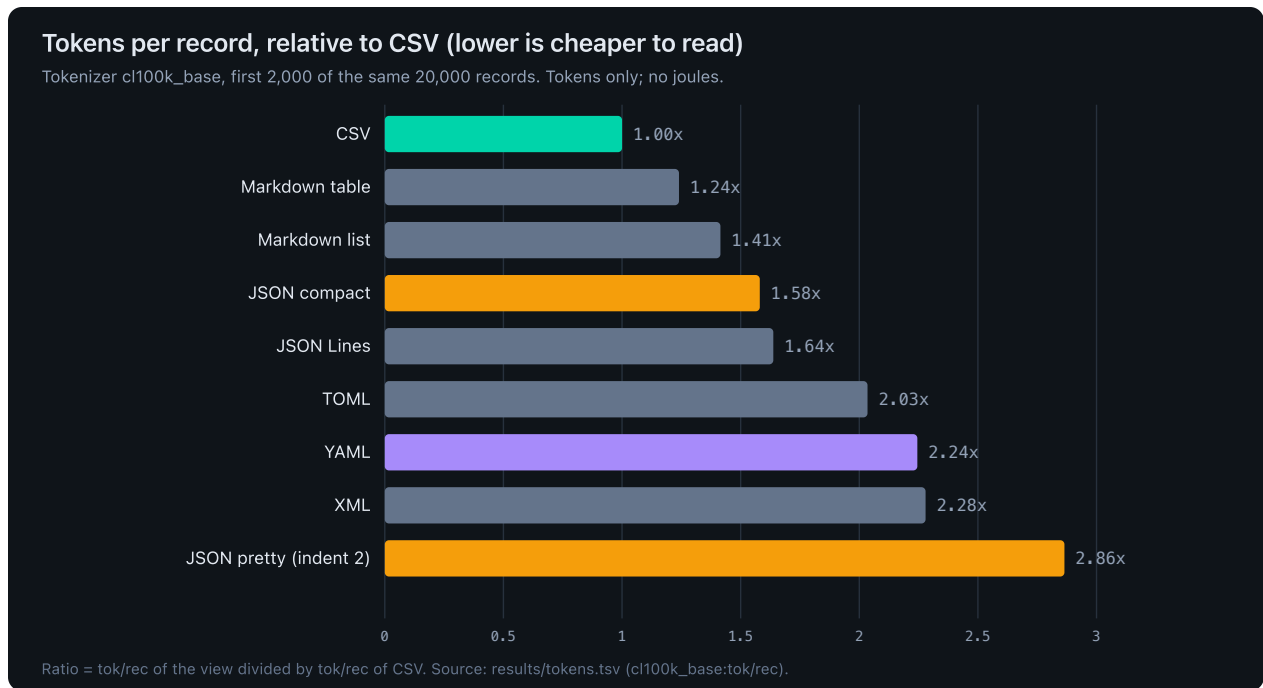


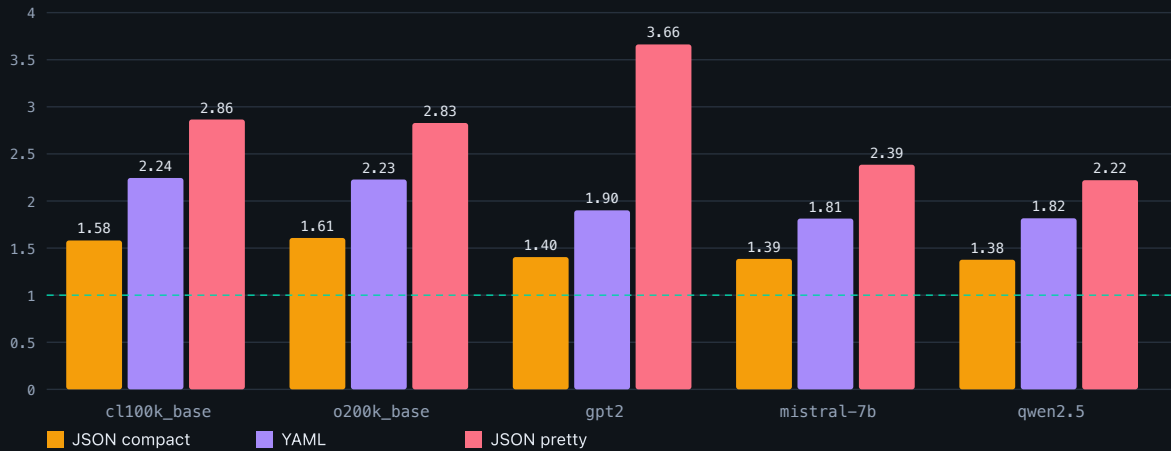
Figure 6. κ = tokens per record of the view divided by tokens per record of CSV. Source: `/living/ai-reads-human/data/tokens.tsv`.

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 $\rho_{\text{CSV}}/\rho_{\text{JSON}} = 0.826$, 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.

The same ratio under five tokenizers

Tokens per record divided by CSV tokens per record, same tokenizer. Tokens only.

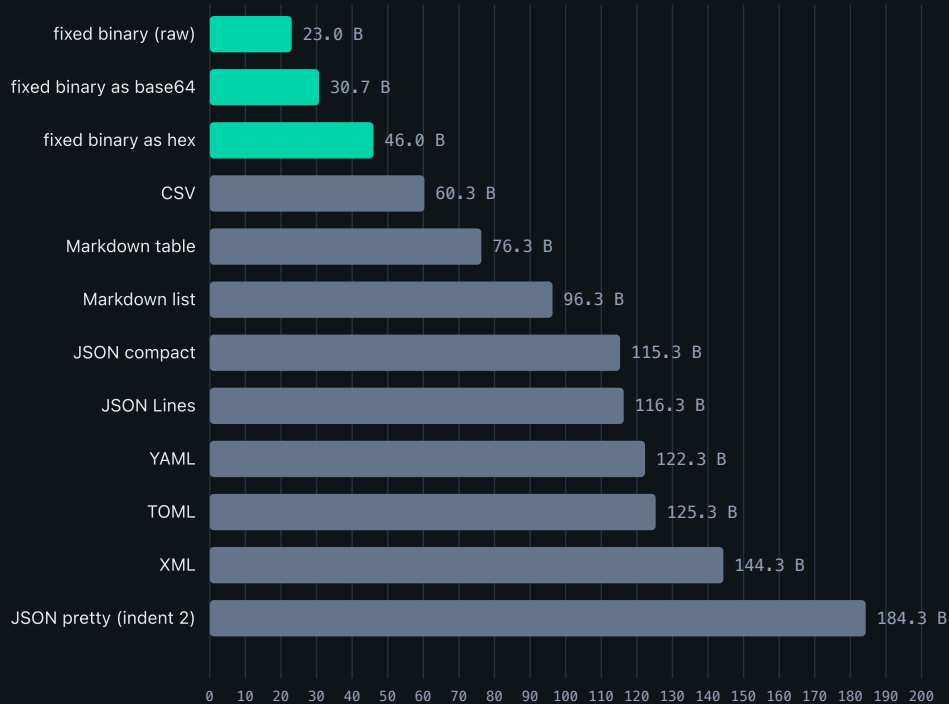


Source: results/tokens.tsv. Dashed reference at 1.0 = CSV. No closed commercial tokenizer was tested.

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.

Encoded bytes per record

Same logical records. Fixed binary uses a 23-byte layout whose words are indexes into a shared 24-word list (a dictionary).



Source: results/tokens.tsv (bytes_per_record); raw binary 23 B = bytes column of parse_time.tsv divided by 20,000 records. Dictionary is not counted.

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 we ran 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
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 $\pm$	RFC 8259 sections 6 and 9; RFC 7493 sec-

Class	Format	Hazard	Evidence
		$(2^{53})-1$.	tion 2.2
Number hazards	JSON	<code>1e400</code> becomes <code>inf</code> in both Python and Node; <code>-0</code> is handled differently.	T3, N3, T4, N4
Number hazards	YAML 1.1 resolver	In PyYAML, <code>010</code> is 8, <code>12:30:45</code> is 45045 and <code>country: N0</code> is <code>{country: False}</code> .	Y rows
Number hazards	TOML	The spec says an integer that cannot be stored losslessly is an error; Python's <code>tomllib</code> accepted 2^{63} .	TOML 1.0.0; T rows
Unicode	JSON	<code>"\uDEAD"</code> (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 <code>=</code> is plain data to the parser and a formula to some spreadsheet programs.	OWASP CSV Injection; C row
Injection	XML	With entity resolution on, <code>lxml</code> 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 <code>RecursionError</code> 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

Nothing in this section is done. It is the order we intend to follow.

ID	Measurement	Needs
M1, M2, M3	Parse time, tokens, tokenizer throughput	Done, time and counts only (this page)

ID	Measurement	Needs
M4	Prefill energy and decode energy per token against context length, same model, GPU with a hardware energy counter	A machine with NVML or DCGM energy counters
M5	Parse energy for the M1 workloads by RAPL package energy, with an idle baseline and interleaved formats	A machine where Intel RAPL is readable (a macmon attempt on two laptops is in section 5b: null result)
M6	A second corpus: natural-language documentation and logs against identifier-heavy telemetry	Corpus and scripts
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	Task set and an agent harness
M8	Other parsers per format (simdjson, rapidyaml, libxml2 C, pulldown-cmark) to bound language-runtime effects	Scripts

Each result will be published with its raw files and the idle baseline. A joule claim will be made only from M4 or M5 data.

5b. Measured energy: no net effect resolved

Measured 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), which reads Apple's IOReport and SMC sensors as a normal user (no root). all_power is CPU plus GPU plus ANE package power; sys_power is whole-system 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: the net energy of parsing (load power minus idle baseline, times time, divided by records) could not be resolved. On every machine, power counter and format, the mean net energy per record is smaller than about 1.5 standard errors from zero (largest t value 1.45, 10 rounds), and several means are negative. The cause is that both laptops were running other work during the test (one-minute load average 41 to 68), so the idle baseline moved by 2 to 7 W between the before and after windows, while one parsing thread adds only a watt or two to a package drawing 18 to 32 W.

What can be said, labelled by how obtained:

- *Measured, upper bound only:* the gross energy per record, which includes all background work on the machine, was about 7e-6 J (48 GB) and 1.1e-5 J (128 GB) for fixed binary, 5.4e-5 J and 9.0e-5 J for stdlib JSON, and 3.6e-3 J and 7.2e-3 J for YAML (libyaml). The

true parse energy is no larger than these. Gross energy here is mostly background power times time, so these ratios track the time ratios and are not an independent energy result.

- *Measured, time*: on these machines JSON took about 7.6 times (48 GB) and 8.9 times (128 GB) as long as fixed binary per record, and YAML about 470 and 750 times.
- *Not measured*: any statement that one format costs fewer joules than another. The page still makes no such claim.

A valid result needs a quiet machine or an inline USB-C power meter on the charger measured with the machine otherwise idle.

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 but leaves choices to each protocol. A length-prefixed binary form removes most parsing work, but a text model cannot read it directly, so a text view or a projection of the needed fields is still required. Which trade-off wins for a given workload is a question for M4, M5 and M7 in section 5, not for argument. This page measures none of these forms except the fixed binary layout in section 3.

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 measured results.
- `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 depend on the machine; expect the ordering to hold and the nanosecond values to differ.

8. Threats to validity

- Time is not energy; power differs by workload, so no joule inference from the time ratios is valid. The gross energy figures in section 5b are upper bounds that include background work.
- Python library overhead dominates some rows (M8 would bound it).
- One synthetic, identifier-light corpus; natural language is untested (M6).
- Tokenizers tested are open ones; closed commercial tokenizers are untested.
- The binary layout depends on a shared dictionary that the size figures do not count.
- All measurements come from one run on one machine; none was repeated elsewhere.

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

Item	Identifier
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", source of the quoted 640 pJ per 32-bit DRAM read	unverified, not read, 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: the energy measurements of section 5; a natural-language corpus; per-task token counts for agent workloads; 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.