技术论文 · jsonW 1.3.11

jsonW：一个数据模型、三种无损形式，以及格式内置的权限

Wortic Tecnologia · 2026 年 11 月 13 日 · 文本 1.1，基于 jsonW 1.3.11

jsonw.org/zh/paper · github.com/developer-wortic/jsonw

目录

摘要

JSON 和 JSONL 已成为数据的通用语言，但它们的设计目的是在程序之间交换消息，而不是存储和查询 AI 系统所使用的海量、多样的数据。读取 JSONL 文件中的一个字段必须从头解析文件；货币、计量和精确小数没有类型；向量变成文本；权限和完整性证明都在数据之外。本文介绍 jsonW（json Wondrous）1.3.11：一个拥有三种可无损互相转换形式的数据模型，即用于消息和实时场景的行形式、通过 `mmap` 读取的分块 `.jsw` 文件这一列形式，以及文本形式（每个合法的 JSON 也都是合法的 jsonW 文本）。该格式自带业务类型、向量和张量、十九项内置功能（搜索、排序、维护、拼写相近以及音频、文本和视频信号）、按权限的视图以及基于哈希的存证。在一台机器上对 100,000 条合成记录的测试中，`.jsw` 文件占用的空间是未压缩 JSONL 的 1/6.6，读取一列的速度约为我们自己的 JSON 读取器的 2,300 倍。在真实的 INPI 商标数据集（5530 万行、12.53 GB 的 CSV）上，`.jsw` 比 CSV 小 6.5 倍，比 XML 小 12.8 倍，并在 43 毫秒内统计出 230 万条有效记录，快于 DuckDB 读取 Parquet。

关键词：数据格式；列式存储；JSON；向量搜索；访问控制；完整性；后量子密钥交换。

1 引言

JSON [1] 的诞生是为了用人类可读的文本在程序之间交换消息。JSON Lines [2] 把这个思路带到了文件里：每行一条记录。两者都成了日志、导出、训练集和 API 响应的事实标准。它们简单、通用、可读，这些特点解释了它们的成功。然而，为 AI 处理数据，需要的却是它们原本就不是为之设计的东西。

1.1 JSON 和 JSONL 缺少什么

- **大小。**一切都是文本。一个 384 维的向量在每条记录里变成几千个字符：第 5 节测试中的 100,000 条记录以 JSONL 存储时占 519.3 MB。
- **访问。**没有索引。要读取某条记录的一个字段，必须从文件开头一直解析到它；要读取一列，就得解析整个文件。
- **类型。**规范没有规定数字的精度：RFC 8259 把上限留给各个实现，并建议只依赖 IEEE 754 双精度的精度 [1, 28]。没有带币种的货币类型、带单位的计量类型、精确小数、不能丢失前导零的代码、日期、向量或张量。每个系统各自发明约定，而在约定之间转换就会丢失数据。
- **权限。**谁能看到哪一列、哪一行，这件事存在于数据之外的另一层，每次查询都必须记得去套用它。
- **存证。**文件里没有任何东西能证明它没有被改动过。

在实践中，同一份数据要经过一条链：线路上用 JSON，存储用 CSV 或 Parquet，语义搜索用向量数据库，再加一层权限来决定谁能看到什么。每一步都要花时间和内存，有时还会损失数据本身：价格变成了文本，税号丢了前导零，重量到底是克还是千克没人说得清。

1.2 贡献

本文介绍 jsonW（json Wondrous）1.3.11 版本。它的贡献是：

1. 一个拥有三种可无损互相转换形式（行、列和文本）的数据模型，其中每个合法的 JSON 也都是合法的 jsonW 文本（第 2 节）；
2. 直接在模式中声明的业务类型和 AI 类型（第 2.3 节）；
3. 文件内置的十九项功能，从搜索、排序到音频、文本和视频信号，凡是公开工具有对应命令的都给出了演示（第 3 节）；
4. 作为格式一部分的列、行、单元格级权限和基于哈希的存证（第 4 节）；

5. 可复现的测量，包括方法、数据和测量的边界（第 5 节），以及与 JSON、JSONL、MessagePack、CBOR、Protobuf、Parquet 和 Arrow 的比较（第 6 节）。

1.3 本文不描述的内容

jsonW 的核心用 C 编写，除标准库外没有任何依赖，由 Wortic Tecnologia 维护；它不是开源的。公开的、也是本文所使用的部分，包括使用文档、C 语言 ABI（`sdk/jsonw.h`）、各语言 SDK、工具的二进制文件以及可观察的行为 [20]。内部的二进制布局、压缩编解码器和数据块的内部组织不在本文范围内。我们关于格式和工具的陈述都可以用公开的二进制文件验证；内部证据（模糊测试、恶意负载、跨 SDK 比对）由我们自己完成，并会注明。如果某项功能在格式中存在、但公开工具还没有对应命令，我们会明确说明。

2 数据模型

2.1 三种形式

jsonW 是同一个数据模型，有三种存储同一个值的方式（图 1）。从一种形式转到另一种，不会丢失任何值，也不会改变任何值的类型。

- **行：**二进制，一个值接一个值，用于消息、事件和实时场景。它取代 JSON、MessagePack、CBOR 和 Protobuf，是在线路上传输的形式（第 4.5 节）。
- **列：**`.jsw` 文件，按列存储的数据块，通过 `mmap` 直接从磁盘读取，无需复制。它取代 JSONL、CSV 和 Parquet。
- **文本：**jsonW 文本，供人阅读和书写。它取代 JSON、TOML 和 YAML。每个合法的 JSON 都已经是合法的 jsonW 文本；文本形式还为 JSON 缺少的类型增加了写法，例如 `12.50 BRL`、`250 g` 和 `t"2026-11-09T01:00:00.000000Z"`。

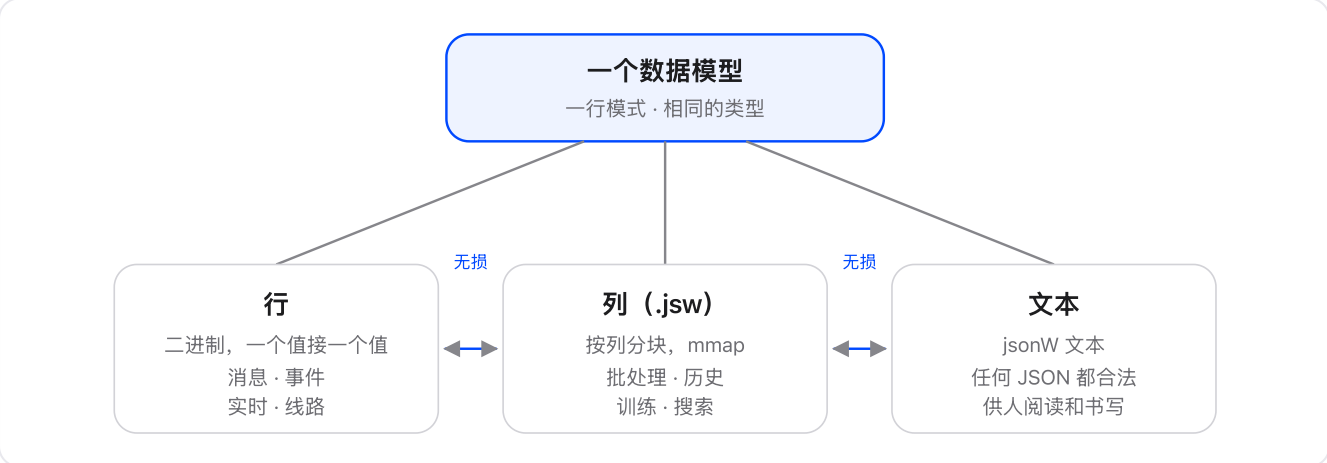


图 1. 一个模型，三种形式。三种形式的模式和类型都相同；在它们之间转换不会丢失任何值，也不会改变任何类型。

2.2 模式

模式是一行文本，由逗号分隔的 `名称:类型` 对组成。问号表示允许为空，类型后面的标记声明权限（第 4.3 节）。类型和命令的名称是葡萄牙语单词：`texto` 是文本，`dinheiro` 是货币，`medida` 是计量，`codigo` 是代码，`tempo` 是时间，`pessoal` 是个人信息。

```
id:i64,produto:texto,preco:dinheiro,peso:medida[g],sku:codigo,criado:tempo,cpf:codigo?+pessoal
```

代码清单 1. 一个模式。`cpf`（巴西的个人税号）允许为空，并且属于个人数据：没有 `pessoal` 权限范围时，它输出为空。

模式可以演进而不破坏读取方：新字段在旧行中为空，一个名称永远不改变类型，读取方按名称解析字段。所以旧文件可以被新代码读取，新文件也可以被旧代码读取 [20]。

2.3 类型

表 1 列出了 1.3.11 版本的类型。业务类型在写入时校验：不符合类型的输入会被拒绝并说明原因，而不是被错误地存下来。

表 1. jsonW 1.3.11 的类型 [20]。

类型	在模式中	保证什么
精确小数	`decimal[2]`	从不经过浮点：12.50 始终是 12.50
货币	`dinheiro`，`dinheiro[4]`	每个值都带币种，用 ISO 4217 [27] 代码或符号表示（`12,50 BRL`、`US$`、`€`、`₿`）；只在同一币种内求和
计量	`medida`，`medida[kg]`	14 种量纲中的 89 个单位；写入时换算，拒绝不同量纲
代码	`codigo`	精确字节：税号、密钥、车牌和 SKU 从不被规范化或截断
文本	`texto`，`texto+nfc`	完整 Unicode，包括重音、emoji 和 `\0`；`+nfc` 按需规范化 [29]
重复文本	`dic`	每个文件一个字典：每段文本只存一次
时间	`tempo`	微秒，UTC
url	`url`	写入时校验（`https://…`、`mailto:`、`tel:`、`urn:`；最长 8 KiB）
向量	`vf32[n]` `vf16[n]` `bits[n]` `vq8[n]` `vq4[n]`	文件中连续的矩阵，可直接用于语义搜索
张量	`tensor<bf16>[768,768]`	元素可为 f32、f16、bf16、u8、i8、i16、i32
媒体	`midia`	哈希、类型、大小、时长、宽和高；jsonW 不固定编解码器：媒体保持其自身的开放编解码器
其他	`bool` `i64` `f64` `f32` `uuid` `bytes` `ref` `lista<t>` `obj{…}` `mapa<k,v>` `qualquer`	`qualquer`（任意）可存放开放的 JSON；嵌套最多 1000 层

14 种计量量纲是长度、质量、体积、能量、时间、面积、数据量、速度、温度、无量纲、功率、压强、碳和密度。十进制倍数之间的换算是精确的（12.5 mm = 0.0125 m），其他换算按列的精度四舍五入（1 mi = 1609.344 m，在没有小数位的 `medida[m]` 列中存为 1609 m；25 °C = 77 °F）。

2.4 逗号还是小数点

读取方接受两种分隔符：`12,50`、`1.234,56 BRL` 和 `1,234.56 USD`。两种都出现时，最后一个是小数点。只出现一种时，它就是小数点，但有两个例外：使用小数点的币种中逗号后跟三位数字（`USD 1,234` 是一千），或使用逗号的币种中小数点后跟三位数字（`R$ 1.234` 是一千）。规范输出始终使用小数点。而且实数从不依赖进程的区域设置：设为葡萄牙语的 JVM 或 PHP 不会调换分隔符。

2.5 一个实际运行的例子

代码清单 2 展示了从 JSONL 到 `.jsw` 再回来的过程，附有 `wjsw` 1.2.1 工具的真实输出。

```
# 两行 JSONL，就像来自任何系统的那样
{"id":1,"produto":"Café especial","preco":"12,50 BRL","peso":"250 g","sku":"CAF-0250"}
{"id":2,"produto":"Pão de queijo","preco":"8,90 BRL","peso":"1 kg","sku":"PDQ-1000"}

$ wjsw de-texto "id:i64,produto:texto,preco:dinheiro,peso:medida[g],sku:codigo" vendas.jsonl
vendas.jsw
2 linhas -> vendas.jsw
$ wjsw ver vendas.jsw
{id:1,produto:"Café especial",preco:12.50 BRL,peso:250 g,sku:"CAF-0250"}
{id:2,produto:"Pão de queijo",preco:8.90 BRL,peso:1000 g,sku:"PDQ-1000"}
$ wjsw para-texto vendas.jsw --json
{"id":1,"produto":"Café especial","preco":"12.50 BRL","peso":"250 g","sku":"CAF-0250"}
{"id":2,"produto":"Pão de queijo","preco":"8.90 BRL","peso":"1000 g","sku":"PDQ-1000"}

# 不符合类型的输入会被拒绝，并给出原因
$ wjsw de-texto "…,peso:medida[g],…" leite.jsonl leite.jsw     # peso: "2 L"
valor 1: medida de outra grandeza perto de ","sku":"LEI-0002"}
```

代码清单 2. `1 kg` 变成了 `1000 g`，因为该列是 `medida[g]`；升是体积而该列是质量，所以这一行被拒绝（“另一种量纲的计量”；消息会显示出错位置附近的输入）。货币保留币种，SKU 逐字节存储。工具的消息用葡萄牙语输出。

3 19 项功能

下面这些功能属于格式和核心本身，而不是外加的一层，公开工具提供了其中的大部分（表 2）。在别的技术栈里，这些部件分散在搜索引擎、向量数据库、索引和权限层之中。本节的例子使用 Linux x86_64 的公开 `wjsw` 1.2.1 二进制文件运行（代码清单 6B 使用 1.3.11），输入文件在 jsonw.org/paper/exemplos；较长的输出在 `…` 处截断。

表 2. 19 项功能及其在公开工具和 ABI 1.3.11 中的位置。

#	功能	作用	在工具中
1	按词搜索	带有按词权重（IDF）的稀疏向量，即搜索引擎使用的计算方法	`avaliar … esparso`
2	语义搜索	在 `vf32`、`vf16`、`vq8` 和 `vq4` 稠密向量上计算余弦	`avaliar … cosseno`
3	多线性网格	双线性、三线性及更高阶插值，直接从文件计算	`avaliar … grade`
4	按请求评分	权重、余弦、稀疏或网格，写在一行请求里	`avaliar`
5	前 k 个	基于堆的确定性 top-k：分数最高者在前，同分时行号小者在前	`avaliar`, `escolher`
6	区间	每个数据块每一列的最小值和最大值：搜索可以跳过整个数据块	`filtrar`, `info`
7	字典	重复的文本在每个文件中只存一次	`dic` 类型
8	键索引	直接定位到某个键所在的行，无需扫描	在核心中；没有单独的命令
9	词库	从磁盘读取的 Aho-Corasick 自动机；按整词匹配，不区分重音和大小写	`reconhecer`
10	实时写入	一个进程读取的同时另一个进程写入，永远不会读到半个数据块	在核心中；没有单独的命令
11	有效、已删除与历史	再次写入同一个键即为编辑；`_apagado: true` 即为删除；压缩只重写当前有效的内容	在核心中；没有单独的命令
12	按权限的视图	列、行和单元格，依据凭证决定	`ver`, `wjsw-cgi`, SDK
13	存证	每个数据块的 crc32c 和 SHA-256（或 BLAKE3），首尾相连成链	`conferir`, `reparar`
14	远程部分读取	借助文件头和文件尾，只下载被读取的行所在的数据块	`wremoto`
15	拼写与读音相近	按拼写和语音键的三元组给出 0 到 1 的分数，支持巴西葡萄牙语、美式英语、西班牙语和中文	`filtrar … ~~`
16	音频信号	音高、停顿、强度、jitter、shimmer、HNR、共振峰、MFCC、谱质心、语速和呼吸	ABI：`jsonw_sinal_audio`
17	文本信号	强化词、弱化词、极性、绝对词、否定、被动语态、代词、形式和反讽迹象，支持 pt、en、es 和 zh	ABI：`jsonw_sinal_texto`
18	视频分镜	镜头切换、每一段的度量和音频信号，以及帧的拼图	ABI：`jsonw_video_*`
19	按数据块和按租户的列	按数据块的局部字典、定宽数字、租户区间和批量比较	写入时；`filtrar`、`calcular`

3.1 搜索与排序（功能 1 至 5）

`avaliar`（评估）命令为每一行打分并返回前 k 个。请求可以是数值列的加权和、向量列与查询之间的余弦、稀疏向量之间的乘积，或者一次网格查询。同样的请求也有 JSON 形式，在工具和 ABI 中完全一致，所有 SDK 都逐字节返回相同的结果 [20]。计算只并行读取请求中涉及的列。代码清单 3 使用五个商品，带有一个 `vf16[4]` 向量列和一个 `mapa<i64,f32>` 词项列。

```
$ wjsw de-texto
"id:i64,produto:texto,categoria:dic,preco:decimal[2],nota:f64,vet:vf16[4],termos:mapa<i64,f32>"
prod.jsonl prod.jsw
5 linhas -> prod.jsw

# 4 和 5. 按权重打分（价格为负向，评分为正向），取前 3 个
$ wjsw avaliar prod.jsw pesos "preco=-1,nota=10" 3
37.1    1       {id:2,produto:"Pão de queijo",categoria:"padaria",preco:8.90d,nota:4.6,…}
35.5    0       {id:1,produto:"Café especial",categoria:"bebida",preco:12.50d,nota:4.8,…}
31.1    2       {id:3,produto:"Chá verde",categoria:"bebida",preco:9.90d,nota:4.1,…}

# 2. 语义搜索：与查询的余弦
$ wjsw avaliar prod.jsw cosseno vet "1,0,0.2,0.1" 2
0.9869597291884717      2       {id:3,produto:"Chá
verde",…,vet:vf16[0.7998046875,0,0.300048828125,0.0999755859375],…}
0.9681576829495052      0       {id:1,produto:"Café especial",…}

# 1. 按词搜索：稀疏向量之间的乘积（词项 12 和词项 90）
$ wjsw avaliar prod.jsw esparso termos "12=1,90=0.5" 2
1.2000000476837158      0       {id:1,produto:"Café especial",…,termos:[[12,1.2000000476837158],
[40,0.40000000596046445]]}
1.0     4       {id:5,produto:"Suco de uva",…,termos:[[12,0.30000001192092896],
[90,1.399999976158142]]}

# 同样的请求用 JSON 表示，与 ABI 中一致
$ wjsw avaliar prod.jsw '{"pesos":{"preco":-1}}' 2 --json
{"linha":1,"nota":-8.9,"valor":{"id":2,"produto":"Pão de queijo",…}}
{"linha":2,"nota":-9.9,"valor":{"id":3,"produto":"Chá verde",…}}

# 5. 某一列中最大的 2 个值
$ wjsw escolher prod.jsw nota 2
4.9     3       {id:4,produto:"Bolo de milho",…}
4.8     0       {id:1,produto:"Café especial",…}
```

代码清单 3. 输出：分数、行位置和 jsonW 文本形式的行，从最好到最差。

输出中有两个细节值得注意。第一，以 `vf16` 存储的向量读回来是 `0.7998046875`，而输入是 `0.8`：半精度大约保留三位十进制数字。如果向量的精确度很重要，该列应使用 `vf32`；`vq8` 和 `vq4` 则用更多精度换取更小的体积。第二，按词搜索就是点积：IDF 权重 [22] 存在每一行的向量里，查询带上自己的权重。

网格（功能 3）。网格是一张沿多个坐标轴排列的数值表（例如按距离和重量计算的运费）。请求 `{"grade":{"campos":["x","y"],"arquivo":"g.jsw"}}` 把行中的列当作坐标，返回在相邻网格点之间插值得到的值作为分数：两个轴为双线性，三个轴为三线性，依此类推。在 1.3.11 版本中，网格表的格式还没有写入公开文档，所以这里不展示它的输出。

3.2 文件组织（功能 6 至 8）

区间。每个数据块都记录每一列的最小值和最大值。如果某个数据块不可能满足过滤条件，读取方就会跳过整个数据块。在第 5 节那个包含 103 个数据块的 100,000 条记录文件中：

```
$ wjsw filtrar com.jsw "id > 99990"
9 de 100000 linhas em 1.7 ms · blocos 103, pulados pela zona 102 · linhas lidas 833
$ wjsw info prod.jsw
…
dicionario 2 textos
zona      preco        890 .. 2400
zona      nota         3.8999999999999999 .. 4.9000000000000004
```

代码清单 4. 在 100,000 条记录的文件中（id 从 0 到 99,999），区间跳过了 103 个数据块中的 102 个（“pulados pela zona”）；在代码清单 3 的商品文件中，`info` 显示了 `categoria` 列的字典和区间，实数按工具的输出以双精度显示。

字典。 `dic` 列在每个文件中只存一次每个不同的文本。在测试的 100,000 条记录中，`tipo` 和 `fonte` 两列共有 10 个不同的文本，`wjsw info` 显示 `dicionario 10 textos`。

键索引。格式中有一个索引，可以直接定位到某个键所在的行，无需扫描文件。在公开工具 1.3.11 中，它既没有命令，也没有 ABI 函数；键的可见用法是 `comparar`（比较），它按某一列配对两个文件的行，并统计差异：

```
$ wjsw comparar ontem.jsw hoje.jsw sku
so em a 1 · so em b 1 · mudou 1 · iguais 0 (0.0 ms)
```

仅在 a 中：1 · 仅在 b 中：1 · 有变化：1 · 相同：0。

3.3 词库（功能 9）

模式为 `termo:texto,forma:dic,sinal:dic,peso:f32,idioma:dic,origem:dic` 的词项表，在打开时会被编译成一个 Aho-Corasick 自动机 [21]，只需一次扫描就能找出文本中的所有词项。匹配按整词进行，不区分重音和大小写，返回原文中的片段以及按信号汇总的权重：一个无需训练的文本分类器，可用于分流，以及把请求转给合适的专家。

```
$ wjsw reconhecer termos.jsw "Quero CANCELAR o pedido, ou então me diga quanto custa o orcamento
novo"
3 termo(s), 2 sinal(is)
  intenção de compra soma 2.2  vezes 2
  risco de perda       soma 2  vezes 1
  [6..14] CANCELAR  (termo 0, risco de perda, peso 2)
  [43..55] quanto custa  (termo 3, intenção de compra, peso 1.2)
  [58..67] orcamento  (termo 2, intenção de compra, peso 1)
```

代码清单 5. 一句葡萄牙语（“我想取消订单，或者告诉我新报价多少钱”）。“CANCELAR”匹配了“cancelar”，“orcamento”匹配了“orçamento”，因为匹配会忽略重音和大小写。信号分别是“购买意向”和“流失风险”。

3.4 写入与历史（功能 10 和 11）

`.jsw` 文件只追加写入。这带来两个保证。第一是**实时写入**：一个进程可以在另一个进程写入文件时读取它，就像一个永远不会读到半个数据块的 `tail -f`。第二是**历史**：再次写入同一个键是编辑，用 `_apagado: true` 写入该键是删除，压缩只把当前有效的内容重写到新文件里，历史留在原文件中。

在 1.3.11 版本中，写入期间的读取、按键的“只看有效内容”视图以及压缩，在 `wjsw` 中都没有命令，在公开 ABI 中也没有函数。用公开工具时，历史会原样逐行显示：

```
$ wjsw ver dup.jsw
{id:1,nome:"a",_apagado:null}
{id:2,nome:"b",_apagado:null}
{id:1,nome:"a2",_apagado:null}
{id:2,nome:null,_apagado:true}
```

公开 ABI 在写入时确实保证的是**原子发布**：只有在 `fechar()`（关闭）时文件才完整地出现。中途因错误退出的写入方调用 `abandonar()`（放弃），目标文件保持原样。

3.5 权限、存证与远程访问（功能 12 至 14）

按权限的视图（12）和存证（13）是第 4 节的主题。**远程部分读取**（14）使用 HTTP `Range` 请求 [24]：读取方先下载文件头和文件尾，其中有模式、数据块索引和区间，然后只下载需要的数据块（图 4）。我们用 `wremoto` 1.2.1 对一个本地 nginx 进行了测量，它提供那个 100,000 条记录的文件（第 5 节的 `com.jsw`，此处命名为 `cartoes.jsw`）；把 HTTP 地址换成 `file://`，`wremoto` 在没有服务器的情况下给出相同的数字：

```
$ wremoto info http://127.0.0.1:18099/cartoes.jsw
… 100000 linhas em 103 blocos · arquivo 79.1 MB · abrir baixou 8.7 KB (cabecalho + rodape)
$ wremoto ver http://127.0.0.1:18099/cartoes.jsw 50000 1
{id:50000,tipo:"fator de emissão",titulo:"Fator de emissão de CO₂ do combustível nº 50000",…}
baixado no total: 777.8 KB em 5 pedidos (0.98% do arquivo)
$ wremoto filtrar http://127.0.0.1:18099/cartoes.jsw "id > 99990"
…
baixado no total: 667.7 KB em 5 pedidos (0.84% do arquivo)
```

代码清单 6. 为了读取一个 79.1 MB 文件中间的一行，读取方下载了文件的 0.98%（“baixado no total”）；过滤时下载了 0.84%，因为区间在下载之前就排除了其他数据块。

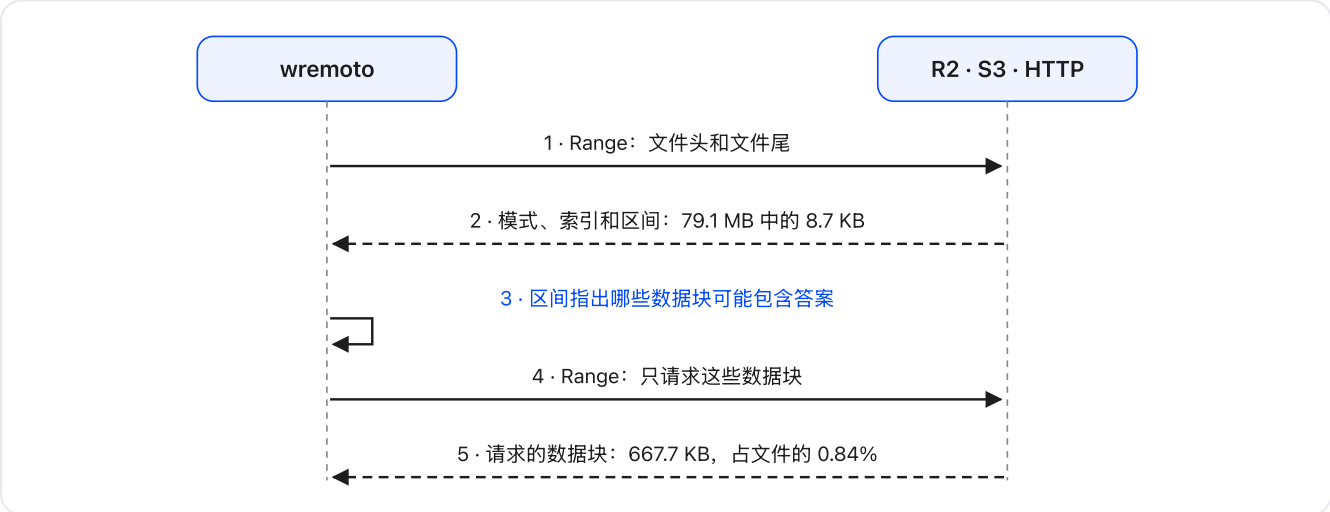


图 4. 对 100,000 条记录文件的远程部分读取，数字来自代码清单 6 的测量。使用 R2 或 S3 时，每个请求都用 AWS SigV4 签名 [25]。

3.6 拼写、信号与按数据块的列（功能 15 至 19）

拼写与读音（功能 15）。过滤器的 `~~` 运算符给每个值一个 0 到 1 的分数：取两个基于 Unicode 字符三元组的 Dice 系数中较大的一个，一个算在规范化的拼写上（去掉重音、不分大小写），另一个算在所请求语言的语音键上。目前有巴西葡萄牙语、美式英语、西班牙语以及用拼音书写的中文的语音键；对汉字，三元组

按字计算，而汉字的读音键需要一张读音表，本版本还没有。过滤器不使用索引：只并行扫描被请求的那一列（代码清单 6B）。

```
$ wjsw filtrar marcas.jsw "nome ~~pt Kokakola"
2 de 4 linhas em 0.0 ms · blocos 1, pulados pela zona 0 · linhas lidas 4
{id:1,nome:"Coca-Cola"}
{id:4,nome:"Kokakola Bebidas"}
```

代码清单 6B. 拼写与读音相近。"Kokakola" 能找到 "Coca-Cola"：葡萄牙语的语音键把硬音 "c" 和 "k" 归为同一个音。`~~en`、`~~es` 和 `~~zh` 切换语言；`~~pt@0.4` 改变阈值。

沟通信号（功能 16 至 18）。它们测量一个人如何说话、书写或出镜，但不做任何判断：解读由读取这一行的模型负责。音频（16 位 PCM）中，音高来自归一化自相关，与 Praat 的方法相同；jitter 和 shimmer 来自声门脉冲；共振峰来自线性预测（LPC）；停顿来自对强度取 Otsu 阈值，切分点取自数据本身而不是固定数值。文本中，语言的词表（pt、en、es 或 zh，手工编写）由功能 9 的自动机匹配，形式上的计数（大写、标点、emoji、笑声）不使用词典。中文不用空格分词，按字逐个匹配。视频中，由调用方解码并交出帧和声音；每一帧变成一张带亮度、对比度、颜色和形状的缩略图，镜头切换同样由帧间差异的 Otsu 阈值得出，且只在序列明显分成两组时才切。每个度量都以声明的模式输出为一行 jsonW 文本，可直接写入 `.jsw`。

按数据块和按租户的列（功能 19）。三种新的编解码器：每个数据块的局部字典（带或不带熵编码），用于在块内重复、但在整个文件中不重复的文本；以及定宽数字，用于只由数字组成的编码，例如申请号。每个数据块的区间也记录租户，在会话视图下，其他客户的数据块根本不会被打开。过滤和计算用向量指令（SSE2）批量比较字典编码，并一次性对整个数据块求值“包含”。用这些编解码器写入的文件无法用 1.2.x 的读取器打开。同一版本中，最高压缩级别（`--max`）新增了两种编解码器，把每个单元格和每个向量存为与上一行的差：相同的前缀和后缀，以及中间变化的部分。在 INPI 数据库上的效果见表 5C 和表 5D。

4 安全与存证

4.1 存证

文件的每个数据块都可以独立成立，并带有两项校验：一个快速的 CRC32C [11]，用于发现意外损坏；以及一个密码学哈希，默认为 SHA-256 [12]，也可选用 BLAKE3 [13]。这些哈希组成一条覆盖整个文件的链：数据、区间或字典中任何一个字节被改动，`conferir`（校验）都会失败（图 2）。

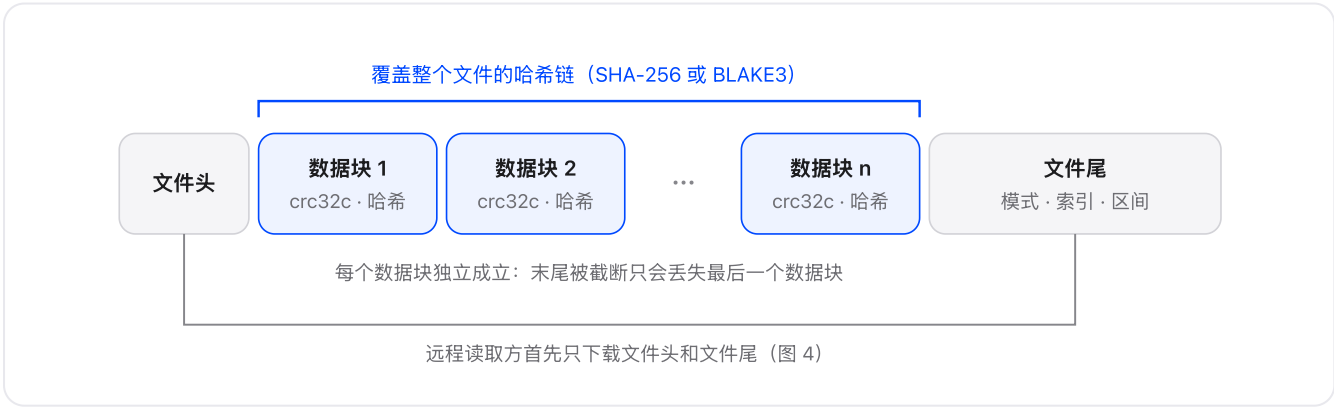


图 2. 从外部看 `.jsw` 文件，所示层次与公开文档的描述一致。文件头、数据块和文件尾的内部组织没有公开（第 1.3 节）。

```
$ wjsw conferir vendas.jsw
ok: 2 linhas, 1 blocos, sha256 e crc32c conferidos
# 改动文件中间的一个字节
$ wjsw conferir adulterado.jsw
FALHOU: bloco 0: nao veio ou nao confere
# 截掉最后 20 个字节，就像写入被中断一样
$ wjsw conferir cortado.jsw
FALHOU: sem rodape: rode wjsw reparar
$ wjsw reparar cortado.jsw && wjsw conferir cortado.jsw
ok
ok: 2 linhas, 1 blocos, sha256 e crc32c conferidos
# 用 BLAKE3 代替 SHA-256
$ wjsw regravar vendas.jsw vendas-b3.jsw --hash blake3 && wjsw conferir vendas-b3.jsw
ok: 2 linhas, 1 blocos, blake3 e crc32c conferidos
```

代码清单 7. 存证、篡改和修复，附真实输出（“FALHOU”= 失败；“sem rodape: rode wjsw reparar”= 缺少文件尾：请运行 wjsw reparar）。

> 存证证明的是完整性，而不是来源。不带密钥的哈希可以被重写整个文件的人重新计算。要证明某个文件就是从某个来源发出的那一个，请把 `wjsw info` 显示的文件标识另外保存，或用独立的工具签名，然后比对。

4.2 读取恶意文件

读取方把每个文件都当作恶意文件：在使用前检查每个索引和每个边界，拒绝超过 1000 层的嵌套，并在测试套件中用模糊测试（随机生成的输入）进行检验。引号、逗号、斜杠以及文本中的任何字节都是数据，永远不会破坏读取。

4.3 按权限的视图

在 jsonW 中，权限在模式中声明，由提供文件的一方在响应之前执行：会话无权看到的内容永远不会离开服务器（图 3，表 3）。

表 3. 按权限视图的各个层次。

层次	如何声明	效果
列	`+escopo(financeiro)`，`+pessoal`	没有相应权限范围时，该列输出为空，甚至不能用于过滤
行	一个名为 `tenant` 或 `tenant_id` 的整数列，或标记为 `+tenant` 的列	只返回凭证所属租户的行；其他租户的数据块根本不会打开；`tenant:todos`（全部）打开所有行；租户是整数：如果 `tenant` 列是文本，则不返回任何行
单元格	两者同时使用	客户自己的行，但不含个人数据列

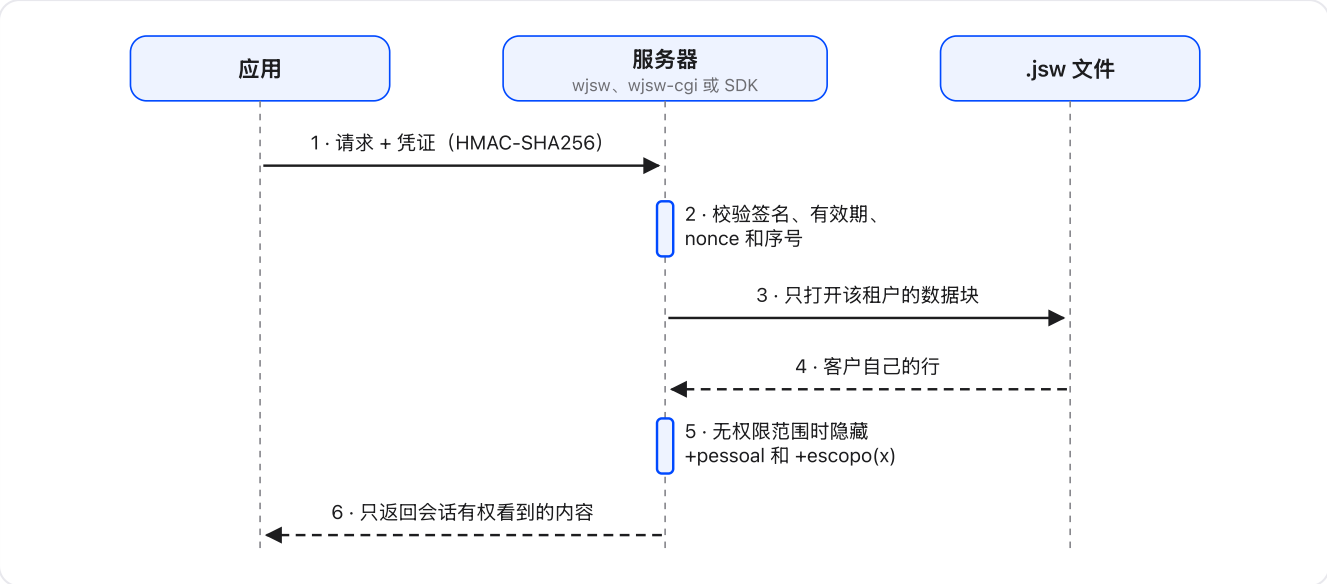


图 3. 按权限的视图。第 2 至 5 步在服务器内部完成；文件永远不会完整地发给请求方。

一家有两个院区的诊所可以说明这三个层次。1 号院区的前台能看到 1 号院区的预约，但看不到税号和金额；1 号院区的财务能看到金额，仍然看不到税号；1 号院区的任何人都看不到 2 号院区：

```
$ wjsw de-texto "paciente:texto,cpf:codigo?
+pessoal,valor:dinheiro+escopo(financeiro),data:tempo,tenant:i64" clin.jsonl clin.jsw
# 1 号院区前台
$ WJSW_ESCOPOS="tenant:1" wjsw ver clin.jsw
{paciente:"Ana Souza",cpf:null,valor:null,data:t"2026-11-12T13:00:00.000000Z",tenant:1}
{paciente:"Carla Dias",cpf:null,valor:null,data:t"2026-11-12T15:30:00.000000Z",tenant:1}
# 1 号院区财务
$ WJSW_ESCOPOS="financeiro tenant:1" wjsw ver clin.jsw
{paciente:"Ana Souza",cpf:null,valor:180.00 BRL,data:t"2026-11-12T13:00:00.000000Z",tenant:1}
{paciente:"Carla Dias",cpf:null,valor:180.00 BRL,data:t"2026-11-12T15:30:00.000000Z",tenant:1}
```

代码清单 8. 同一个查询，两种凭证，两种结果。示例数据均为虚构。

> 视图不是加密。它保护的是离开服务器的内容；谁持有 `.jsw` 文件，谁就持有全部数据，而且 `para-texto` 会导出全部内容，因为它是给文件所有者用的副本。要向用户展示数据，请使用 `ver`、`wjsw-cgi` 或带 `visao()` 的 SDK；要保护静态存储的文件，请使用磁盘或存储加密。

4.4 凭证

随请求一起发送的凭证本身就是一个 jsonW 值，包含签发方、主体、租户、权限范围和有效期，并用 HMAC-SHA256 [18] 签名。签名以恒定时间校验，nonce 加序号防止请求被重放。它运行在 TLS 之上，而不是取代 TLS。

4.5 后量子传输

行形式通过一个基于 QUIC [15] 的协议传输，使用 TLS 1.3 [16] 和混合密钥交换 X25519MLKEM768 [17]，它把 X25519 曲线与 NIST 在 FIPS 203 [14] 中标准化的 ML-KEM-768 结合在一起。后量子的论证分两部分。密钥交换使用一种旨在抵御量子计算机的机制。存证和凭证只使用 256 位的哈希和 HMAC，它们不依赖因数分解或离散对数；要找到原像或 HMAC 密钥，Grover 算法 [19] 只带来平方级加速，仍留下约 128 位的安全性；要找到碰撞，已知的量子界限更低，用 Brassard、Høyer 和 Tapp 算法 [35] 攻击 SHA-256 约需 2^{85} 次运算，并且需要巨大的量子存储器。

`wjsw medir`（第 5 节）的最后一行测量行形式：一条测试记录用 JSON 表示占 5,157 字节，作为消息占 1,073 字节，在每次运行中都小 4.8 倍。

> 公开的 1.3.11 包既不包含该协议的服务器，也不包含客户端。

4.6 Web 服务器

该包包含 `wjsw-cgi`，可在 Apache、nginx、LiteSpeed、Caddy 和 IIS 上运行 [20]。所有 `.jsw` 请求都经过它：没有标记列且没有租户的文件不经过滤直接输出，支持 `Range` 和 CORS；其他文件经过过滤输出，并执行视图规则。权限范围来自身份验证之后的服务器，而不是客户端发送的请求头；通过 `campos=` 或 `onde=` 请求隐藏列会返回 403，因此不会通过推断泄露任何内容。在 1.2.0 版本中，示例配置会在请求没有参数时让服务器直接输出原始文件；1.2.1 修复了这个问题，安装过旧配置的人应当替换它 [20]。

5 测量

本节将 jsonW 1.3.11 与 JSONL 进行对比测量，并同时说明测量不能证明什么。这些数字是在 2026 年 11 月 13 日用发布版的二进制文件为本文重新测量的，所用机器与 11 月 12 日的测量相同。

5.1 方法

- **工具：**公开二进制文件自带的 `wjsw medir`（测量），使用 100,000 行：`wjsw medir <目录> 100000`。二进制文件为公开仓库中的 `bin/linux-x86_64/wjsw`，版本 1.3.11（预发布版，SHA-256 `cb119d7473e7d786…`）。
- **机器：**一台虚拟服务器，AMD EPYC 9355P，8 个 vCPU，18 GB 内存，Ubuntu 26.10，Linux 内核 7.3。文件放在 `tmpfs`（内存）上，因此测量不包括磁盘。这台机器没有与其他负载隔离。
- **数据：**100,000 条合成记录，形如知识库中的卡片：`id:i64,tipo:dic,titulo:texto,corpo:texto,fonte:dic,valor:decimal[4],vetor:vf16[384],tags:lista<texto>`。文本按模板重复，只改变行号；`tipo` 和 `fonte` 在 5 个值之间轮换；向量有 384 个分量，都是 1/1024 的倍数，`vf16` 可以无损存储。
- **JSON 一侧：**同一组数据写成 JSONL，由 jsonW 自己的文本读取器读取，它解析每一行并转换模式中的类型。两个文件都在计时开始前载入内存。
- **重复：**连续运行 10 次。我们报告每个时间的中位数，以及倍数在 10 次中的中位数和范围。原始输出随本文一起发布。
- **校验：**每次运行时，工具都会检查两次读取都数到了 100,000 行，且总和相同。

5.2 时间

表 4. 写入和读取 100,000 条记录的时间。时间：10 次运行的中位数。倍数：每次运行中 JSONL ÷ jsonW，给出 10 次中的中位数和范围。8 线程一行的倍数是相对于单线程读取 JSONL 而言。JSONL 由 jsonW 自己的文本代码写入和读取。

操作	JSONL	jsonW	倍数（范围）
写入	15,211 ms	131 ms	114×（88 至 127）
全部读取，所有字段	3,687 ms	78 ms	46×（43 至 51）
全部读取，8 线程	（无并行版本）	9.0 ms	419×（365 至 532）
读取一列（`valor`）	3,687 ms	1.59 ms	2,310×（1,592 至 2,784）
跳到第 N 行，字段 `titulo`	从头读取	0.09 µs（0.05 至 0.16）	无确定倍数

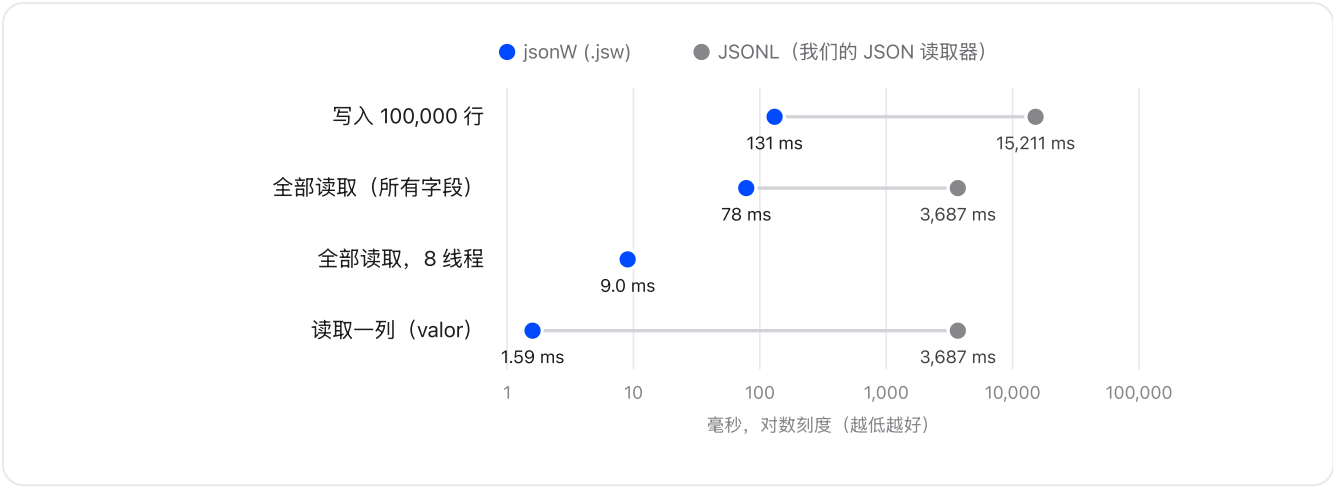


图 6. 表 4 的时间，对数刻度（10 次运行的中位数）。每一行两点之间的水平距离就是倍数。

图 6 以对数刻度展示同样的时间。差距最大的是读取一列：JSONL 必须解析每一整行才能取出一个字段，而列式文件只读取这一列。这是结构上的优势，任何列式格式相对行式格式都有。对于“跳到第 N 行”，JSONL 无法跳转：必须从头读到那一行，所以倍数取决于 N，我们不报告它。

5.3 大小

为了与通用压缩工具比较大小，我们重写了同样的 100,000 条记录：随本文发布的 `awk` 生成器产生 JSONL；`wjsw de-texto` 写出 `.jsw`；`wjsw para-texto --json` 返回的 JSONL 有 519,264,563 字节，按 MB 计与 `wjsw medir` 为自己的 JSONL 报告的大小相同（519.3 MB）；`.jsw` 有 79,076,300 字节（两者都是 79.1 MB）。生成器产生的 JSONL 多 200,000 字节，因为其中的小数值带了引号；表 5 使用 jsonW 重写的 JSONL，若改用生成器的文件，倍数只在小数点后第一位变化（gzip：19.7× 而不是 19.8×）。JSONL 分别用 gzip 1.14 [32]、zstd 1.5.7 [31] 和 xz 5.8.3 [33] 以各自常用的最高级别压缩。

表 5. 100,000 条记录的大小，单位 MB（10^6 字节），含与不含向量列。倍数 = 未压缩 JSONL ÷ 该行的大小。

文件	含向量	倍数	不含向量	倍数
JSONL	519.3	1.0×	38.2	1.0×
`.jsw`， `vf32` 向量	155.9	3.3×	不适用	不适用
`.jsw`， `vf16` 向量，默认级别	79.1	6.6×	2.22	17.2×
`.jsw`， `vq8` 向量（有损）	41.1	12.6×	不适用	不适用
JSONL + gzip -9	26.3	19.8×	1.42	26.9×
JSONL + zstd -19	2.67	194.3×	0.63	60.6×
JSONL + xz -9	1.56	332.5×	0.47	81.3×
`.jsw`， `vf16` 向量， `--max`	1.36	381.1×	0.0175	2,185×

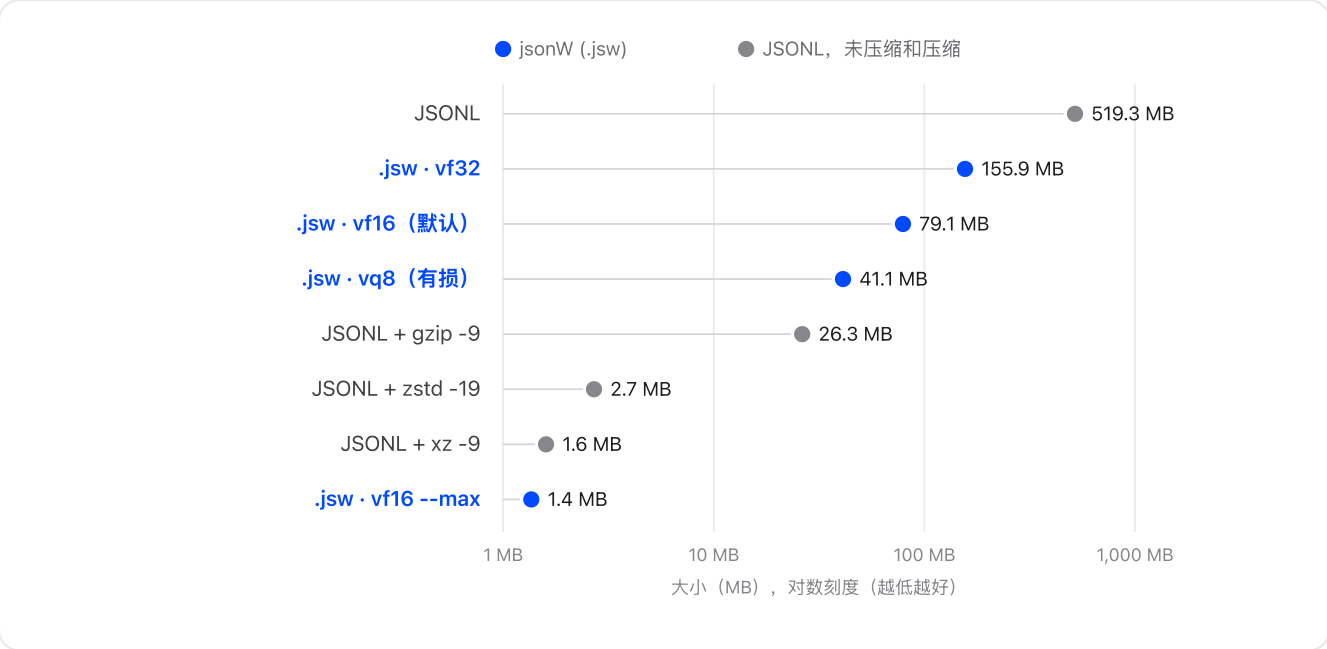


图 5. 表 5 中含向量的大小，对数刻度。

图 5 显示了带向量时的大小。与未压缩的 JSONL（JSONL 通常就以这种方式存储和交换）相比，默认级别的 `.jsw` 小 6.6 倍，而它的大小几乎全是向量：100,000 行、每行 384 个 2 字节的分量就是 76.8 MB。与用 zstd 或 xz 压缩的 JSONL 相比，默认级别的 `.jsw` 更大。在最高级别下，1.3.11 比两者都小：带向量 1.36 MB，而 zstd 为 2.67 MB、xz 为 1.56 MB；不带向量 17.5 kB，而它们分别为 0.63 MB 和 0.47 MB；在 1.2.1 中，同一级别是 3.75 MB 和 2.19 MB。收益来自最高级别新增的两种编解码器，它们把每个单元格和每个向量存为与上一行的差（相同的前缀和后缀，以及中间变化的部分）；这里收益很大，是因为测试数据非常重复：文本只换一个数字就重复出现，向量遵循算术规律。往返是无损的：最高级别文件的 `para-texto` 与默认级别文件的逐字节相同（带向量和不带向量的 SHA-256 都一致）。在真实数据上收益要小得多：在 INPI 数据库上，1.3.11 的最高级别比 1.2.1 的小 16.5%（表 5C）。真实模型的嵌入向量通常也压缩得很少；我们在这里没有测量，但如果是这样，带向量的 `.jsw` 的大小会接近向量的原始大小。

通用压缩文件给不了的是访问能力。要读取 zstd 压缩的 JSONL 中的一个字段，必须先全部解压（在同一台机器上、测量环境之外运行 3 次，解压 `com-volta.jsonl.zst` 需要 0.13 至 0.24 秒），然后再解析文本；而 `.jsw` 直接读取该字段，借助区间跳过数据块，校验存证，并且可以远程分段读取。

5.4 真实数据：CSV、XML、XLSX 和 Parquet

上面的测量使用的是生成的数据。为了与最常见的电子表格和交换格式比较，我们使用了公开数据：巴西国家工业产权局（INPI）商标开放数据中 `MARCAS_DADOS_BIBLIOGRAFICOS.csv` 文件的前一百万行（13 列：申请号、日期、表现形式、性质、商标名称、状态）。一个不依赖任何库的 C 程序按 RFC 4180（字段内的引号、逗号和换行）读取 CSV，并把同样的行写成四种形式：重写后的 CSV；每行一个元素、每列一个子元素的 XML；由程序自己组装的 XLSX（Office Open XML，内联文本单元格，用 `zip -6` 压缩）；以及模式手工编写的 `.jsw`（日期为文本；表现形式、性质和状态为字典）。空字段在 XML 和 jsonW 中不出现。XLSX 每张工作表最多 1,048,576 行，所以取一百万行。

四种形式的查询相同：统计有效注册（`codigo_situacao = IPAS023`）。四种形式的结果一致，都是 352,248 条，`wjsw conferir` 校验了 `.jsw` 所有数据块的 SHA-256 和 CRC32C。每个查询在文件已进入缓存后运行 5 次，取最小值。机器：8 核服务器，Linux 7.3，jsonW 1.2.0，2026 年 11 月 13 日。

表 5B. 一百万行 INPI 真实数据的四种格式。大小单位 MB（10^6 字节）；查询 = 统计有效注册，单位毫秒。

格式	大小	zstd -3 后	查询	查询方式
XML	502.1	46.0	159	对整个文件 `grep -c`
CSV	141.1	36.1	97	对整个文件 `grep -c`
XLSX	74.3	不适用	9,364	对工作表 `unzip -p` + `grep`
`.jsw`	51.2	不适用	119	`wjsw calcular --onde`，只读该列

在大小上，`.jsw` 比 CSV 小 2.8 倍，比 XLSX 小 1.5 倍，比 XML 小 9.8 倍。用 zstd 压缩的 CSV 和 XML 比 `.jsw` 更小（36.1 和 46.0 MB 对 51.2 MB），但与第 5.3 节一样，读取任何内容之前都必须完整解压。在查询上，`.jsw` 比 XLSX 快 79 倍，因为 XLSX 每次读取都要解压。对 CSV 和 XML，`grep` 更快或相当：它只是在整个文件中查找一段文本，既不分列也不识别类型，如果同一个代码出现在别的列里，结果就会出错。在这组数据中没有出现这种情况，计数一致。要在 CSV 上得到同样可靠的查询，必须读取并切分每一行，在本次测量的第一轮中 `awk` 用了 1,597 ms，而且没有处理引号。

这三种格式本身都不具备的：带类型的模式（日期、小数、货币、向量）、不读其他列而只读一列、按词和按语义搜索、按权限的视图、已删除行的历史，以及按数据块的存证。XLSX 有单元格类型和多张工作表，但限制了工作表的大小，并且必须解压才能读取。

整个数据库。表 5B 用的是一个文件的一百万行。为了看格式在完整数据上的表现，我们转换了 INPI 商标开放数据 [38] 的全部七个文件（2026 年 11 月 12 日下载），一行不删：共 55,278,212 行、12.53 GB 的 CSV，包含所有列以及每个申请的完整决定历史（仅 `MARCAS_DESPACHOS.csv` 就有 24,978,976 行）。每个文件都有手工编写的模式。数字、日期和代码保留各自的类型；文本保留原样的字节。C 编写的转换器会当场检查每个带类型的值：数字和日期必须还原成与 CSV 中相同的文本，否则转换停止。这发现了源数据中的两个真实情况，都按原样保存：一个年份为 1018 的优先权日期（原文中的录入错误），以及尼斯分类文件 5,333,893 行中有一行没有申请号。

Parquet [3] 由 DuckDB 2.0.0 [39] 写入，使用相同的类型和 zstd 压缩。XML 沿用表 5B 的结构。XLSX 没有参与：七个文件中有六个超过每张工作表 1,048,576 行的上限，决定历史文件接近该上限的 24 倍。八种形式在三个查询中的计数都一致，只有一个例外，见下文。`wjsw conferir` 校验了七个 `.jsw` 所有数据块的 SHA-256 和 CRC32C。每个查询在文件已进入缓存后运行 5 次，取最小值。写入 jsonW 使用一个核心。DuckDB、zstd 和 xz 使用全部八个核心，`wjsw calcular` 也是。机器：与表 5B 相同，31 GB 内存，2026 年 11 月 13 日。

表 5C. INPI 完整商标数据库（七个文件，5530 万行）在各种格式下的情况。大小单位 MB（10^6 字节），为七个文件之和；写入 = 从 CSV 生成该格式所用的时间，单位秒。

格式	大小	比 CSV 小的倍数	写入	能否不打开其余部分只读一列?
XML	24,804.7	0.5	不适用	否
CSV（原始）	12,527.4	1.0	不适用	否
`.jsw`，1.2.1	2,804.8	4.5	176.7	能
`.jsw`，1.3.11	2,579.2	4.9	不适用	能
`.jsw`，1.2.1，`--max`	2,321.8	5.4	184.9	能
XML + zstd -3	2,210.0	5.7	21.4	否，需全部解压
`.jsw`，1.3.11，`--max`	1,938.1	6.5	不适用	能
CSV + zstd -3	1,908.9	6.6	13.2	否，需全部解压
Parquet（zstd）	1,665.8	7.5	16.6	能
CSV + xz -6	1,161.0	10.8	544.5	否，需全部解压

表 5D. 在整个数据库上的三个按列查询，单位毫秒（5 次中的最小值），于 2026 年 11 月 13 日 06:21 UTC 第五次测量，发布版 `wjsw` 1.3.11 与 1.2.1 在同一时间测量。有效：在 670 万个申请中统计 `codigo_situacao = IPAS023`。历史：在 2500 万条记录中取一个申请的全部决定（`numero_inpi = 818602856`）。第 35 类：在 530 万条记录中统计 `classe_nice = 35`。括号内为 DuckDB 单核结果。XML 来自第一轮，因为该文件之后已被删除（占 24.8 GB）。

格式与读取方	有效	历史	第 35 类
正确答案	2,343,607	10	1,076,928
Parquet，DuckDB	54 (76)	129 (521)	41 (60)
`.jsw --max` 1.3.11，`wjsw calcular --onde`	43	207	142
`.jsw` 1.3.11，`wjsw calcular --onde`	44	254	152
`.jsw --max` 1.2.1，`wjsw calcular --onde`	49	310	145
`.jsw` 1.2.1，`wjsw calcular --onde`	54	353	198
CSV，DuckDB 读取 CSV	375 (1,447)	1,205 (4,523)	592 (1,979)
CSV，`grep -c`	519	6,690	1,278（错误：1,076,929）
CSV + zstd，`zstd -dc | grep -c`	798	3,129	2,786（错误）
CSV + xz，`xz -dc | grep -c`	1,050	3,025	2,105（错误）
CSV + xz，`xz -dc | grep -c`	971	2,628	1,950（错误）
XML，`grep -c`（第一轮）	3,017	9,925	3,430

整个数据库说明了什么。在大小上，1.3.11 的 `.jsw` 在默认级别下比 CSV 小 4.9 倍，最高级别下小 6.5 倍，比 XML 小 12.8 倍；1.2.1 比 CSV 小 4.5 倍和 5.4 倍，而 1.3.11 最高级别比 1.2.1 小 16.5%。在查询上，两边都用 8 个核心，1.3.11 统计有效记录在最高级别用时 43 毫秒、默认级别 44 毫秒，快于 DuckDB 读取 Parquet 的 54 毫秒；用 1 个核心时，DuckDB 读取 Parquet 在这一计数上用时 76 毫秒，在历史查询上用时 521 毫秒。在全部三个查询中，`.jsw` 都快于 DuckDB 读取 CSV、`grep`、压缩后的 CSV 和 XML。在单个案卷的历史和第 35 类上，`.jsw` 用时 207 和 142 毫秒，8 个核心的 Parquet 用时 129 和 41 毫秒：这是第 8 节

中的下一步性能工作。在大小上，Parquet（1,665.8 MB）和用 xz 压缩的 CSV（1,161.0 MB）比最高级别的 `.jsw`（1,938.1 MB）更小，而压缩后的 CSV 在任何读取前都必须完整解压；用 zstd 压缩的 CSV 与最高级别的 `.jsw` 几乎相同（1,908.9 MB）。用 1 个核心写入 1.2.1 的 `.jsw` 用时 177 秒；用 8 个核心写入 Parquet 用时 17 秒。1.3.11 的文件是从 1.2.1 的 `.jsw` 重写得到的（`wjsw regravar`），因此表 5C 没有给出它们的写入时间。

各轮测量与修正。表 5D 共测量了五次，每次都在同一时间测量 Parquet 和 CSV。使用 `wjsw` 1.2.0 时，`.jsw` 在三个查询上分别用时 677、1,494 和 1,114 ms，是 Parquet 的 10 至 31 倍：过滤器读取了每一行的所有列。修正为只读取被查询的列后，用时 82、311 和 209 ms。第三轮中，评估改为按数据块进行，直接在列中比较字典编码和整数，只有通过过滤的行才组装：48、344 和 186 ms。第四轮时 1.3.11 仍在准备中，历史查询从 355 ms 降到 279 ms。第五轮使用发布版二进制文件，它用向量指令批量比较字典，并按数据块对“包含”求值：1.3.11 在默认级别下用时 44、254 和 152 ms，最高级别下 43、207 和 142 ms，而同一时间的 1.2.1 为 54、353 和 198 ms。历史查询查找的是申请号，这是一个只由数字组成的编码，1.3.11 以定宽方式存储它（在 DADOS_BIBLIOGRAFICOS 中，仅这一列就从 53.0 MB 降到 25.7 MB）。这台机器没有与其他负载隔离，同一个二进制在两轮之间的差异最多达 23%（Parquet 的有效计数从 41 到 54 ms），所以比较只在同一轮内有效。用新编解码器写入的文件无法用 1.2.x 的读取器打开。CSV 上 `grep` 的时间波动更大（历史查询从 2,102 到 6,690 ms），因为 12.5 GB 的 CSV 无法与其他文件一起全部放进缓存，而第五轮之前刚做过的重写把 CSV 挤出了缓存；以同一轮的数字为准。

计数中的例外是第 35 类。在 CSV 上（无论是否压缩）`grep` 数出 1,076,929 条，比正确答案多一条：片段 `,35,` 在某一行的另一列中也出现了。这正是表 5B 旁边描述的风险，那组数据里没有发生，这里发生了。所有能把列分开的读取方（jsonW、Parquet、DuckDB 读取 CSV，以及标记中带列名的 XML）都给出了 1,076,928。

在这次 1:1 转换中，除类型和列之外唯一用上的功能是存证：七个文件的每个数据块都有 CRC32C 和 SHA-256，并已在上文校验。为了与 CSV 保持一致，这次转换没有标记权限，也没有建立搜索。该格式允许在同一个文件中，对没有权限范围的人隐藏权利人的 CPF 和 CNPJ（第 4.3 节），远程只读取其中一部分，以及按词和按语义搜索（第 3 节）。Parquet 有可选的按页校验和 [36] 和按列加密 [5]，但没有覆盖整个文件的哈希链，也没有按权限范围的视图；CSV 和 XML 都不具备这些。

5.5 测量的局限

- **JSON 读取器是我们自己的。**JSON 一侧由 jsonW 自己的文本代码写入和读取，它会校验并转换类型；读取 JSONL 的速度约为 140 MB/s。最先进的 JSON 读取器（如 simdjson [23]）超过 1 GB/s。换用其中之一，表 4 中“写入”和“全部读取”两行会大幅缩小。粗略估算（不是测量）：以 2 GB/s 读取 519 MB 约需 260 ms，“读取一列”的倍数会在几百而不是几千。
- **合成且重复的数据，**如第 5.3 节所述。大小倍数很大程度上取决于数据，在保存相邻行之差的最高级别下更是如此：请用您自己的数据测量。
- **一台机器、一个系统、用内存代替磁盘。**绝对时间随处理器、存储和负载而变化；在 10 次运行中，每项操作的最大倍数比最小倍数高 19% 至 75%；读取一列的波动最大，因为 jsonW 在这里的时间只有 1.6 ms。
- **Parquet 只用 INPI 数据和一种读取器测量。**与 Parquet 的性能比较见表 5C 和 5D，由 DuckDB 2.0.0 负责写入和读取。换用其他读取器、其他压缩方式或其他行组大小，数字都会变化。Arrow、MessagePack、CBOR 和 Protobuf 在本版文本中仍没有性能测量；第 6 节比较的是功能，而不是速度。

5.6 此前发布的数字

到 2026 年 11 月 12 日为止，jsonW 的网站和仓库引用的是“比 JSONL 小 4.9 倍”和“读取一个字段快 3,281 倍”，出自 11 月 6 日在另一台机器上用 1.0 版本做的第一次测量。本文 11 月 12 日的文本 1.0 用 1.2.1 重新测量：小 6.6 倍，读取一列约快 2,500 倍，最高级别带向量 3.75 MB、不带向量 2.19 MB。为本文用 1.3.11 在同一台机器上重新测量，结果是小 6.6 倍，读取一列约快 2,300 倍（差异在多次运行之间的波动范围内），最高级别降到 1.36 MB 和 17.5 kB。以本文的数字为准，日期、机器和版本都写在旁边。

5.7 如何复现

```
git clone https://github.com/developer-wortic/jsonw && cd jsonw
export JSONW_SEM_PING=1                       # 不发送安装通知
W=bin/linux-x86_64/wjsw                        # 或 bin/macos-arm64/wjsw
for i in $(seq 1 10); do $W medir /tmp 100000; done

# 表 5：同一组数据的 JSONL、.jsw 和各压缩工具
curl -O https://jsonw.org/paper/gerar-cartoes.awk
LC_ALL=C awk -v N=100000 -v VET=1 -f gerar-cartoes.awk > com.jsonl
$W de-texto
"id:i64,tipo:dic,titulo:texto,corpo:texto,fonte:dic,valor:decimal[4],vetor:vf16[384],tags:lista<texto
>" com.jsonl com.jsw
$W para-texto com.jsw --json > volta.jsonl
gzip -9 -k volta.jsonl; zstd -19 volta.jsonl; xz -9 -k volta.jsonl; ls -l com.jsw volta.jsonl*
```

代码清单 9. 在您的机器上复现本文的测量。`VET=0` 加上不含 `vetor` 列的模式，得到“不含向量”一列。

6 与其他格式的比较

表 6 比较各格式本身能做什么，不考虑其上的库。成熟的列式格式 Parquet 和 Arrow 已经具备第 5 节测得的大部分访问优势，还有若干 jsonW 尚不具备的东西。

表 6. 各格式本身提供的能力。来源见各列标注的参考文献。

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
形式	文本；JSONL 每行一条记录	二进制，无模式	二进制，编译的模式（`.proto`）	列式文件	内存中的列；IPC 文件和流	二进制行、列式文件和文本
模式	无（JSON Schema 另行定义）	无	有，在消息之外	有，在文件中	有	有，一行，在文件中
只读一个字段	否	否	否	能，按列	能，直接访问	能，按行和字段
精确小数	否	CBOR：有（十进制分数，tag 4）	非原生	有（DECIMAL）	有（decimal128 和 256）[37]	有
货币与计量	否	否	通过通用消息表示货币（`google.type.Money`）[10]	否	否	有，带币种和单位
稠密向量	以文本表示的数字列表	列表；CBOR 有类型化数组 [34]	重复字段	列表或定长字节	定长列表	f32、f16、8 位和 4 位、位；张量
权限	否	否	否	用不同密钥加密各列 [5]	否	按列、行和单元格的视图，由提供方执行；不加密
完整性	否	否	否	可选的按页 CRC [36]；认证加密	格式中没有	每个数据块的 crc32c 和哈希，首尾成链
格式内的搜索与排序	否	否	否	按行组和页的统计信息，用于跳过数据	否	区间、余弦、稀疏、网格、top-k、词库
远程部分读取	否	否	否	能（文件尾和 Range）	能，在文件格式中	能
规范	开放（RFC）	开放（CBOR：RFC）	开放	开放（Apache）	开放（Apache）	用法、文本和 ABI 公开；内部二进制布局不公开

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
实现	许多，彼此独立	许多，彼此独立	若干	若干，彼此独立	若干，彼此独立	一个（C 核心），提供 10 种语言的 SDK

各个格式的位置。JSON 通用，在任何地方都可读。MessagePack、CBOR 和 Protobuf 成熟，在数十种语言中都有实现，Protobuf 有成熟的模式演进规则。Parquet 和 Arrow 是开放标准，几乎所有分析引擎都能读取，Parquet 还有面向静态文件的按列加密。jsonW 可以与它们共同工作：任何 JSON 都已经是 jsonW 文本，`wcon` 可以按正确类型导入和写回数据库表，静态文件可以由磁盘或存储加密。

jsonW 把什么放在了一起。jsonW 的不同不在于某一项单独的功能，而在于在同一个模型里同时具备：消息形式、按列存储的文件形式，以及与 JSON 兼容的文本形式；写入时就检查的业务类型；读取时按列、行和单元格执行的权限；以及在文件内部完成的排序、搜索和分类。这一选择的代价是：目前只有一个实现，而且核心是闭源的（第 8 节）。

7 用途

7.1 AI 模型与语义搜索

一个知识库可以放进一个 `.jsw`，里面是每个段落的文本、元数据和向量；语义搜索就是一次只读向量列的 `avaliar … cosseno`，而词表和稀疏向量负责按词搜索。借助前 k 名和词表，同一个文件还可以充当选择专家的路由器：词表给请求文本分类，排序做出选择。

7.2 企业数据

像巴西的企业登记（CNPJ）和邮政编码（CEP）这样的登记数据，正是 JSON 损失最大的地方：编码变成数字后，开头的零就没了；注册资本变成 double；日期丢失了时区。在 jsonW 里，每样东西都有自己的类型：

```
cnpj:codigo,razao_social:texto,uf:dic,municipio:dic,cep:codigo,capital_social:dinheiro,abertura:tempo
,cnae:dic,tenant:i64
```

代码清单 10. 一个企业登记的模式：逐字节精确的编码、放进字典的重复文本、带币种的金额，以及用来区分客户的 tenant。

7.3 数据库

包里的数据库连接器 `wcon` 能把 PostgreSQL、MySQL 和 MariaDB、SQLite、Redis 和 Valkey 以及 MongoDB 的数据导入 `.jsw`，也能导回去，模式取自每一列的类型：`numeric` 变成精确的十进制数，`bigint` 仍是 64 位整数，pgvector [30] 的 `vector` 和 `halfvec` 变成 `vf32` 和 `vf16`。值始终作为数据传输，前面带着长度，从不拼进命令里；导回在一个事务里完成。往返已在 PostgreSQL 14、16 和 18（两端 md5 相同）、MariaDB 11.8（每列的十六进制）、SQLite、Redis 7 和 MongoDB 7 上验证，使用了恶意负载：注入、引号、`\0`、重音字符、emoji 和从右到左的文本 [20]。

7.4 Web 服务器

借助 `wjsw-cgi`（第 4.6 节），网站可以把一个 `.jsw` 以 JSON、NDJSON 或原样提供出去，地址里可以带过滤条件、字段选择和数量上限，并且始终使用该会话的视图。

7.5 在 R2 和 S3 上存储

一个 `.jsw` 可以放在 Cloudflare R2 或任何 S3 的存储桶里，并直接在那里查询，不必下载整个文件（图 4）。由于 R2 不收数据流出费用 [26]，查询大文件中的一部分，花费的是读取的那一部分加上请求次数，R2 按请求收费（代码清单 6 发出了 5 次请求）。

8 局限与后续工作

- **一个实现，闭源核心。**二进制布局没有公开；读写 `.jsw` 依赖 `libjsonw`。稳定的 ABI 和各个 SDK 降低了风险，但不能代替一份有独立实现的开放规范。
- **没有独立审计。**文中引用的证据（模糊测试、恶意负载、各 SDK 之间的逐字节比较）都是我们自己的。
- **平台。**目前有 macOS arm64、Linux x86_64（glibc 2.34 或更新）和 Windows x86_64 的二进制，还没有 macOS Intel、Linux arm64 和使用 musl 的 Linux 的二进制。在 Windows 上，`wcon` 和 `wremoto` 通过 WSL 运行。
- **没有公开命令的功能。**按键索引、实时写入、“只看当前有效记录”的视图和压缩整理已经在核心里，但在 `wjsw` 里没有命令，在 ABI 1.3.11 里也没有函数；网格表的格式没有文档。
- **协议不在包里。**后量子协议没有包含在 1.3.11 公开包中。
- **葡萄牙语词汇。**类型和命令使用葡萄牙语词汇（`texto`、`dinheiro`、`medida`、`ver`、`conferir`），不读葡萄牙语的人需要一份术语表。
- **数据库。**MySQL 8 只能配合 `mysql_native_password` 使用；`caching_sha2_password` 的首次完整认证还没有实现。
- **测量。**见第 5.5 节列出的局限。
- **按列查询：下一步。**在完整的 INPI 数据集上（表 5D），`wjsw calcular` 1.3.11 统计有效记录用时 43 毫秒，快于 DuckDB 读取 Parquet；从第一轮（1.2.0）到现在，这一计数从 677 毫秒降到 43 毫秒。下一步是把同样的按块、批量读取扩展到单个案卷的历史和第 35 类，在这两项上 `.jsw` 用时 207 和 142 毫秒，8 个核心的 DuckDB 用时 129 和 41 毫秒。在大小上，下一批编码（整数和日期的差分与位打包）的目标是最高级别的 1,938.1 MB。

本文和这个包的**后续工作**：macOS Intel、Linux arm64 和 musl 的二进制；一种只校验每个数据块 CRC32C、把 SHA-256 留给 `conferir` 的查询方式（Parquet 读取时不校验任何哈希，剩下的差距有一部分来自这里）；带窗口的字典编解码器，用于尼斯分类这样的列；对整数和日期做差分编码加位打包，并对照表 5C 和表 5D 重新测量；以及在同一台机器上与 Arrow 做性能比较。

9 获取与许可

工具、库、ABI、各个 SDK、Web 服务器配置和文档都在 github.com/developer-wortic/jsonw [20]，Node.js SDK 在 npm 上，名为 `@worticteconologia/jsonw`。使用受 jsonW 持久使用许可（葡萄牙语）约束，其效力高于本摘要：jsonW 可以在任意数量的服务器和应用上自由使用，包括商业产品，只要它是一个组件，而不是被出售的服务本身；没有与 Wortic 的书面协议，任何人都不能出售、托管或向第三方提供 jsonW，或主要功能就是 jsonW 功能（以 jsonW 存储、转换、提供或传输数据）的服务，无论收费还是免费；以二进制分发的库和工具不得进行逆向工程，法律允许的范围除外；文件的内部布局、编解码器、协议和核心归 Wortic 所有，因此对它们的独立实现也需要书面协议。公开文档可以自由阅读、引用，并用于编写使用 jsonW 的程序。

出于透明：`wjsw` 工具在每台机器上第一次运行时，只发送版本和平台（例如 `v=1.3.11&so=linux-x86_64`），让我们能统计安装数量，之后不再发送。IP 随连接到达服务器，但不会保存，任何名称、路径或数据也都不会保存：服务器只记录日期、平台和版本。在第一次运行前设置 `JSONW_SEM_PING=1` 或创建文件

`~/.jsonw/sem-ping`，即可关闭这次通知。库和各个 SDK 从不访问网络。

参考文献

[1] T. Bray (ed.). *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259, IETF, 2017. rfc-editor.org/rfc/rfc8259

[2] *JSON Lines*. jsonlines.org

[3] Apache Software Foundation. *Apache Parquet: File Format*. parquet.apache.org/docs/file-format

[4] Apache Software Foundation. *Parquet Logical Type Definitions*. github.com/apache/parquet-format/blob/master/LogicalTypes.md

[5] Apache Software Foundation. *Parquet Modular Encryption*. parquet.apache.org/docs/file-format/data-pages/encryption

[6] Apache Software Foundation. *Arrow Columnar Format*. arrow.apache.org/docs/format/Columnar.html

[7] *MessagePack specification*. github.com/msgpack/msgpack/blob/master/spec.md

[8] C. Bormann, P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 8949, IETF, 2020. rfc-editor.org/rfc/rfc8949

[9] Google. *Protocol Buffers: Encoding*. protobuf.dev/programming-guides/encoding

[10] Google. *google/type/money.proto*. github.com/googleapis/googleapis/blob/master/google/type/money.proto

[11] J. Satran et al. *Internet Small Computer Systems Interface (iSCSI)*，其中定义了 CRC32C（Castagnoli）. RFC 3720, IETF, 2004. rfc-editor.org/rfc/rfc3720

[12] NIST. *Secure Hash Standard (SHS)*. FIPS 180-4, 2015. csrc.nist.gov/pubs/fips/180-4/upd1/final

[13] J. O'Connor, J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn. *BLAKE3: one function, fast everywhere*. 2020. github.com/BLAKE3-team/BLAKE3-specs

[14] NIST. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. FIPS 203, 2024. csrc.nist.gov/pubs/fips/203/final

[15] J. Iyengar, M. Thomson (eds.). *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000, IETF, 2021. rfc-editor.org/rfc/rfc9000

[16] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446, IETF, 2018. rfc-editor.org/rfc/rfc8446

[17] K. Kwiatkowski, P. Kampanakis, B. E. Westerbaan, D. Stebila. *Post-Quantum Traditional (PQ/T) Hybrid Key Agreement Mechanisms for TLS 1.3*. RFC 10024, IETF, 2026. rfc-editor.org/rfc/rfc10024

[18] H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, IETF, 1997. rfc-editor.org/rfc/rfc2104

[19] L. K. Grover. *A fast quantum mechanical algorithm for database search*. STOC 1996. arxiv.org/abs/quant-ph/9605043

[20] Wortic Tecnologia. *jsonW (json Wondrous) 1.3.11: 公开仓库、文档（docs/JSONW.md）、ABI（sdk/jsonw.h）和二进制*. github.com/developer-wortic/jsonw

[21] A. V. Aho, M. J. Corasick. *Efficient string matching: an aid to bibliographic search*. Communications of the ACM 18(6), 1975. doi.org/10.1145/360825.360855

[22] K. Spärck Jones. *A statistical interpretation of term specificity and its application in retrieval*. Journal of Documentation 28(1), 1972. doi.org/10.1108/eb026526

[23] G. Langdale, D. Lemire. *Parsing Gigabytes of JSON per Second*. The VLDB Journal 28, 2019. arxiv.org/abs/1902.08318

[24] R. Fielding, M. Nottingham, J. Reschke (eds.). *HTTP Semantics*, 第 14 节（Range Requests）. RFC 9110, IETF, 2022. rfc-editor.org/rfc/rfc9110

[25] Amazon Web Services. *AWS Signature Version 4 for API requests*. docs.aws.amazon.com/IAM/latest/UserGuide/reference_sigv.html

[26] Cloudflare. *R2 pricing*. developers.cloudflare.com/r2/pricing

[27] ISO. *ISO 4217: Currency codes*. iso.org/iso-4217-currency-codes.html

[28] IEEE. *IEEE Standard for Floating-Point Arithmetic*. IEEE 754-2019. doi.org/10.1109/IEEESTD.2019.8766229

[29] Unicode Consortium. *Unicode Normalization Forms*. UAX #15. unicode.org/reports/tr15

[30] *pgvector: open-source vector similarity search for Postgres*. github.com/pgvector/pgvector

[31] Y. Collet, M. Kucherawy (eds.). *Zstandard Compression and the 'application/zstd' Media Type*. RFC 8878, IETF, 2021. rfc-editor.org/rfc/rfc8878

[32] P. Deutsch. *GZIP file format specification version 4.3*. RFC 1952, IETF, 1996. rfc-editor.org/rfc/rfc1952

[33] *XZ Utils*. tukaani.org/xz

[34] C. Bormann (ed.). *Concise Binary Object Representation (CBOR) Tags for Typed Arrays*. RFC 8746, IETF, 2020. rfc-editor.org/rfc/rfc8746

[35] G. Brassard, P. Høyer, A. Tapp. *Quantum Algorithm for the Collision Problem*. 1997. arxiv.org/abs/quant-ph/9705002

[36] Apache Software Foundation. *parquet.thrift*（`PageHeader` 的 `crc` 字段）. github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift

[37] Apache Software Foundation. *Arrow Schema.fbs*（Decimal 类型，128 位和 256 位宽度）. github.com/apache/arrow/blob/main/format/Schema.fbs

[38] Instituto Nacional da Propriedade Industrial（INPI，巴西）。*商标开放数据*（`marcas.zip`，七个 CSV 文件），2026 年 11 月 12 日下载。 dadosabertos.inpi.gov.br/download/marcas/marcas.zip

[39] DuckDB Foundation. *DuckDB 2.0.0*，CSV 与 Parquet 读取。 duckdb.org/docs

如何引用

Wortic Tecnologia. *jsonW：一个数据模型、三种无损形式，以及格式内置的权限*. 文本 1.1，基于 jsonW 1.3.11。2026 年 11 月 13 日。网址：https://jsonw.org/zh/paper。

```
@techreport{jsonw2026zh,
  title       = {jsonW：一个数据模型、三种无损形式，以及格式内置的权限},
  author      = {{Wortic Tecnologia}},
  institution = {Wortic Tecnologia},
  year        = {2026},
  month       = nov,
  note        = {文本 1.1，基于 jsonW 1.3.11},
  url         = {https://jsonw.org/zh/paper}
}
```

文本 1.1，发布于 2026 年 11 月 13 日，基于 jsonW 1.3.11（2026 年 11 月 12 日的文本 1.0 针对的是 1.2.1）。jsonW 即 json Wondrous；C 语言核心由 Wortic Tecnologia 维护。欢迎通过公开仓库提交更正和第三方测量结果。原文为葡萄牙语；本译文保留相同的数字和术语。