Paper técnico · jsonW 1.3.11

jsonW: um modelo de dados, três formas sem perda e a permissão dentro do formato

Wortic Tecnologia · 13 de novembro de 2026 · texto 1.1, sobre o jsonW 1.3.11

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

SUMÁRIO

RESUMO

O JSON e o JSONL viraram a língua comum dos dados, mas foram desenhados para trocar mensagens entre programas, não para guardar e consultar o volume e a variedade de dados que os sistemas de IA usam. Ler um campo de um JSONL exige interpretar o arquivo desde o começo; dinheiro, medida e decimal exato não têm tipo; vetores viram texto; permissão e prova de integridade ficam fora do dado. Este artigo descreve o jsonW (json Wondrous) 1.3.11, um modelo de dados com três formas que se convertem sem perda: linha, para mensagens e tempo real; coluna, o arquivo `.jsw` em blocos lido por `mmap`; e texto, em que todo JSON válido também é jsonW-texto válido. O formato traz tipos de negócio, vetores e tensores, dezenove recursos dentro do arquivo (busca, ranking, manutenção, grafia parecida e sinais de áudio, texto e vídeo), visão por permissão e custódia por hash. Com 100 mil registros sintéticos, numa máquina, o `.jsw` ocupou 6,6 vezes menos que o JSONL sem compressão e leu uma coluna cerca de 2.300 vezes mais rápido que o nosso leitor de JSON. Com a base real de marcas do INPI, 55,3 milhões de linhas e 12,53 GB de CSV, o `.jsw` ficou 6,5 vezes menor que o CSV e 12,8 vezes menor que o XML, e contou os 2,3 milhões de registros em vigor em 43 ms, mais rápido que o Parquet lido pelo DuckDB.

Palavras-chave: formato de dados; armazenamento em colunas; JSON; busca vetorial; controle de acesso; integridade; troca de chaves pós-quântica.

1 Introdução

O JSON [1] nasceu para trocar mensagens entre programas num texto que gente consegue ler. O JSON Lines [2] levou a ideia para arquivos: um registro por linha. Os dois viraram padrão de fato para registros de sistema, exportações, conjuntos de treino e respostas de API. São simples, universais e legíveis, e essas qualidades explicam o sucesso. O trabalho com dados para IA, porém, pede coisas que eles não foram feitos para dar.

1.1 O que falta no JSON e no JSONL

- **Tamanho.** Tudo é texto. Um vetor de 384 dimensões vira milhares de caracteres por registro: os 100 mil registros do teste da Seção 5 ocupam 519,3 MB em JSONL.
- **Acesso.** Não há índice. Para ler um campo de um registro, é preciso interpretar o arquivo do começo até ele; para ler uma coluna, o arquivo inteiro.
- **Tipos.** A especificação não fixa a precisão do número: a RFC 8259 deixa o limite a cargo de cada implementação e recomenda contar só com a precisão do *double* do IEEE 754 [1, 28]. Não há tipo para dinheiro com moeda, medida com unidade, decimal exato, código que não pode perder o zero da frente, data, vetor ou tensor. Cada sistema inventa uma convenção, e a passagem entre convenções perde dado.
- **Permissão.** Quem pode ver qual coluna e qual linha fica fora do dado, numa camada separada, e cada consulta precisa lembrar de aplicá-la.
- **Custódia.** Nada no arquivo prova que ele não foi alterado.

Na prática, o mesmo dado passa por uma cadeia: JSON para trafegar, CSV ou Parquet para guardar, um banco de vetores para a busca por significado e uma camada de permissão para decidir quem vê o quê. Cada passagem custa tempo e memória e, às vezes, o próprio dado: o preço que vira texto, o CPF que perde o zero, o peso que ninguém sabe se era grama ou quilo.

1.2 Contribuições

Este artigo descreve o jsonW (json Wondrous) na versão 1.3.11. As contribuições são:

1. um modelo de dados com três formas que se convertem sem perda (linha, coluna e texto), em que todo JSON válido também é jsonW-texto válido (Seção 2);
2. tipos de negócio e de IA declarados no próprio esquema (Seção 2.3);
3. dezenove recursos dentro do arquivo, da busca e do ranking aos sinais de áudio, texto e vídeo, mostrados com a ferramenta pública quando ela tem comando para eles (Seção 3);
4. permissão por coluna, linha e célula e custódia por hash como parte do formato (Seção 4);
5. medidas reprodutíveis, com metodologia, dados e limites da medida (Seção 5), e uma comparação com JSON, JSONL, MessagePack, CBOR, Protobuf, Parquet e Arrow (Seção 6).

1.3 O que este texto não descreve

O núcleo do jsonW é escrito em C, sem dependência além da biblioteca padrão, e mantido pela Wortic Tecnologia; ele não é aberto. O que é público, e é o que este texto usa, são a documentação de uso, a ABI em C (`sdk/jsonw.h`), os SDKs, os binários da ferramenta e o comportamento observável [20]. O leiaute binário interno, os codecs de compressão e a organização do bloco por dentro ficam fora deste artigo. O que afirmamos sobre o formato e a ferramenta pode ser conferido com os binários públicos; as provas internas (fuzzing, cargas hostis, comparação entre SDKs) são nossas e estão marcadas como tal. Quando um recurso existe no formato mas ainda não tem comando na ferramenta pública, dizemos isso.

2 Modelo de dados

2.1 Três formas

O jsonW é um modelo de dados só, com três formas de guardar o mesmo valor (Figura 1). Passar de uma forma para outra não perde valor e não muda o tipo de nenhum valor.

- **Linha:** binária, um valor depois do outro, para mensagens, eventos e tempo real. Ocupa o lugar do JSON, do MessagePack, do CBOR e do Protobuf. É a forma que corre no fio (Seção 4.5).
- **Coluna:** o arquivo `.jsw` , em blocos guardados por coluna e lidos direto do disco por `mmap` , sem cópia. Ocupa o lugar do JSONL, do CSV e do Parquet.
- **Texto:** o jsonW-texto, para gente ler e escrever. Ocupa o lugar do JSON, do TOML e do YAML. Todo JSON válido já é jsonW-texto válido; o texto acrescenta notações para os tipos que o JSON não tem, como `12.50 BRL` , `250 g` e `t"2026-11-09T01:00:00.000000Z"` .

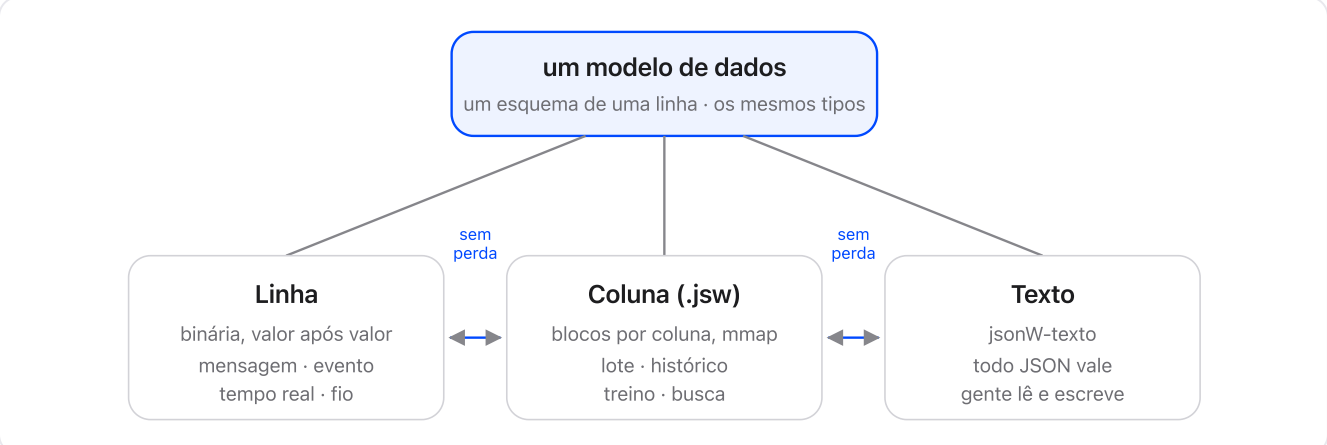


Figura 1. Um modelo, três formas. O esquema e os tipos são os mesmos nas três; a conversão entre elas não perde valor nem muda tipo.

2.2 Esquema

O esquema é uma linha de texto com pares `nome:tipo` separados por vírgula. O ponto de interrogação aceita nulo, e as marcas depois do tipo declaram permissão (Seção 4.3):

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

Listagem 1. Um esquema. `cpf` aceita nulo e é dado pessoal: sem o escopo `pessoal`, sai nulo.

O esquema evolui sem quebrar quem lê: campo novo entra com nulo nas linhas antigas, um nome nunca muda de tipo e o leitor resolve os campos pelo nome. Assim, um arquivo antigo é lido por código novo, e um arquivo novo por código antigo [20].

2.3 Tipos

A Tabela 1 lista os tipos da versão 1.3.11. Os de negócio são conferidos ao gravar: uma entrada que não cabe no tipo é recusada com a razão, em vez de entrar errada.

Tabela 1. Tipos do jsonW 1.3.11 [20].

Tipo	No esquema	O que garante
decimal exato	`decimal[2]`	nunca passa por ponto flutuante: 12,50 continua 12,50
dinheiro	`dinheiro`, `dinheiro[4]`	moeda em cada valor, pelo código ISO 4217 [27] ou pelo símbolo (`12,50 BRL`, `US$`, `€`, `₿`); soma só na mesma moeda
medida	`medida`, `medida[kg]`	89 unidades em 14 grandezas; converte ao gravar e recusa outra grandeza
código	`codigo`	bytes exatos: CNPJ, chave, placa e SKU nunca são normalizados nem aparados
texto	`texto`, `texto+nfc`	Unicode inteiro, com acento, emoji e `\0`; `+nfc` normaliza quando pedido [29]
texto repetido	`dic`	dicionário por arquivo: cada texto guardado uma vez
tempo	`tempo`	microssegundos, UTC
url	`url`	validada ao gravar (`https://…`, `mailto:`, `tel:`, `urn:`; até 8 KiB)
vetor	`vf32[n]` `vf16[n]` `bits[n]` `vq8[n]` `vq4[n]`	matriz contínua no arquivo, pronta para a busca por significado
tensor	`tensor<bf16>[768,768]`	elementos f32, f16, bf16, u8, i8, i16, i32
mídia	`midia`	hash, tipo, tamanho, duração, largura e altura; o jsonW não fixa o codec: a mídia segue no codec aberto dela
e mais	`bool` `i64` `f64` `f32` `uuid` `bytes` `ref` `lista<t>` `obj{…}` `mapa<k,v>` `qualquer`	`qualquer` guarda JSON aberto; aninhamento até 1000 níveis

As 14 grandezas de medida são comprimento, massa, volume, energia, tempo, área, dados, velocidade, temperatura, adimensional, potência, pressão, carbono e densidade. A conversão é exata entre múltiplos decimais (12,5 mm = 0,0125 m) e arredondada na escala da coluna nas demais (1 mi = 1609,344 m, gravado como 1609 m numa coluna `medida[m]`, que não tem casas decimais; 25 °C = 77 °F).

2.4 Vírgula ou ponto

Quem lê aceita os dois separadores: `12,50`, `1.234,56 BRL` e `1,234.56 USD`. Com os dois presentes, o último é o decimal. Com um só, esse separador é o decimal, exceto vírgula seguida de três dígitos numa moeda que usa ponto (`USD 1,234` é mil) ou ponto seguido de três dígitos numa moeda que usa vírgula (`R$ 1.234` é mil). A saída canônica usa sempre ponto. E o número real nunca depende da configuração regional do processo: uma JVM ou um PHP configurados em português não trocam o separador.

2.5 Um exemplo executado

A Listagem 2 mostra a ida de JSONL para `.jsw` e a volta, com a saída real da ferramenta `wjsw` 1.2.1.

```
# duas linhas de JSONL, como chegam de qualquer sistema
{"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"}

# o que não cabe no tipo é recusado, com a razão
$ wjsw de-texto "…,peso:medida[g],…" leite.jsonl leite.jsw     # peso: "2 L"
valor 1: medida de outra grandeza perto de ","sku":"LEI-0002"}
```

Listagem 2. O `1 kg` virou `1000 g` porque a coluna é `medida[g]`; litro é volume, e a coluna é de massa, então a linha foi recusada (a mensagem mostra o trecho da entrada perto do erro). O dinheiro guarda a moeda, e o SKU é guardado byte a byte.

3 Os 19 recursos

Os recursos abaixo fazem parte do formato e do núcleo, não de uma camada por cima, e a ferramenta pública expõe a maioria deles (Tabela 2). Eles são as peças que, em outras pilhas, ficam espalhadas entre um motor de busca, um banco de vetores, um índice e uma camada de permissão. Os exemplos desta seção foram executados com o binário público `wjsw` 1.2.1 para Linux x86_64 (a Listagem 6B, com o 1.3.11), e as entradas estão em jsonw.org/paper/exemplos; nas saídas longas, as linhas foram cortadas onde aparece `…`.

Tabela 2. Os 19 recursos e onde aparecem na ferramenta e na ABI públicas 1.3.11.

#	Recurso	O que faz	Na ferramenta
1	Busca por termos	vetor esparso com pesos por termo (IDF), o cálculo dos buscadores	`avaliar … esparso`
2	Busca por significado	cosseno sobre vetores densos `vf32`, `vf16`, `vq8` e `vq4`	`avaliar … cosseno`
3	Grade multilinear	interpolação bilinear, trilinear e além, direto do arquivo	`avaliar … grade`
4	Nota por pedido	pesos, cosseno, esparso ou grade num pedido de uma linha	`avaliar`
5	As k melhores	top-k por heap, determinístico: nota maior primeiro, empate pela linha menor	`avaliar`, `escolher`
6	Zonas	mínimo e máximo por bloco e coluna: a busca pula blocos inteiros	`filtrar`, `info`
7	Dicionário	texto repetido guardado uma vez por arquivo	tipo `dic`
8	Índice por chave	vai direto à linha de uma chave, sem varrer	no núcleo; sem comando próprio
9	Léxico	autômato Aho-Corasick lido do disco; palavra inteira, sem diferenciar acento nem maiúsculas	`reconhecer`
10	Gravação viva	um processo lê enquanto outro grava, sem ver bloco pela metade	no núcleo; sem comando próprio
11	Vivo, apagado e histórico	gravar a mesma chave edita; `_apagado: true` exclui; compactar regrava só o que vale	no núcleo; sem comando próprio
12	Visão por permissão	coluna, linha e célula, conforme a credencial	`ver`, `wjsw-cgi`, SDK
13	Custódia	crc32c e SHA-256 (ou BLAKE3) por bloco, em cadeia	`conferir`, `reparar`
14	Leitura remota parcial	com cabeçalho e rodapé, baixa só os blocos das linhas lidas	`wremoto`
15	Grafia e som parecidos	nota de 0 a 1 por trigramas da grafia e de uma chave fonética, em pt-BR, inglês americano, espanhol e chinês	`filtrar … ~~`
16	Sinais do áudio	tom, pausas, intensidade, jitter, shimmer, HNR, formantes, MFCC, centroide, taxa de articulação e respirações	ABI: `jsonw_sinal_audio`
17	Sinais do texto	intensificadores, atenuadores, polaridade, absolutos, negação, voz passiva, pronomes, forma e indícios de ironia, em pt, en, es e zh	ABI: `jsonw_sinal_texto`
18	Vídeo em cenas	cortes de cena, as medidas e os sinais do áudio de cada trecho e um mosaico dos quadros	ABI: `jsonw_video_*`
19	Coluna por bloco e por tenant	dicionário local por bloco, dígitos de largura fixa, zona do tenant e comparação em lotes	na gravação; `filtrar`, `calcular`

3.1 Busca e ranking (recursos 1 a 5)

O comando `avaliar` dá uma nota a cada linha e devolve as k melhores. O pedido pode ser uma soma de pesos sobre colunas numéricas, o cosseno entre uma coluna de vetores e uma consulta, o produto entre vetores esparsos ou a leitura de uma grade. O mesmo pedido existe em JSON, idêntico na ferramenta e na ABI, e cada SDK devolve o mesmo resultado, byte a byte [20]. O cálculo lê só as colunas do pedido, em paralelo. A Listagem 3 usa cinco produtos com uma coluna de vetores `vf16[4]` e uma de termos `mapa<i64,f32>`.

```
$ 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 e 5. nota por pesos (preço pesa contra, nota a favor) e as 3 melhores
$ 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. busca por significado: cosseno com a consulta
$ 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. busca por termos: produto entre vetores esparsos (termo 12 e termo 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],
[40,0.4000000059604645]]}
1.0     4       {id:5,produto:"Suco de uva",…,termos:[[12,0.30000001192092896],[90,1.399999976158142]]}

# o mesmo pedido em JSON, como na ABI
$ 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. as 2 maiores de uma coluna
$ wjsw escolher prod.jsw nota 2
4.9     3       {id:4,produto:"Bolo de milho",…}
4.8     0       {id:1,produto:"Café especial",…}
```

Listagem 3. Saída: nota, posição da linha e a linha em jsonW-texto, da melhor para a pior.

Dois detalhes da saída valem nota. Primeiro, o vetor gravado como `vf16` volta como `0.7998046875` onde a entrada dizia `0.8`: meia precisão tem cerca de três dígitos decimais. Quando a exatidão do vetor importa, a coluna deve ser `vf32`; `vq8` e `vq4` trocam mais precisão por tamanho. Segundo, a busca por termos é um produto escalar: os pesos IDF [22] vêm gravados no próprio vetor de cada linha, e a consulta traz os seus.

Grade (recurso 3). Uma grade é uma tabela de valores sobre eixos (por exemplo, o frete por distância e por peso). O pedido `{"grade":{"campos":["x","y"],"arquivo":"g.jsw"}}` usa colunas da linha como coordenadas e devolve, como nota, o valor interpolado entre os pontos vizinhos da grade: bilinear com dois eixos, trilinear com três, e assim por diante. Na versão 1.3.11, o formato da tabela da grade ainda não está na documentação pública; por isso não mostramos a saída aqui.

3.2 Organização do arquivo (recursos 6 a 8)

Zonas. Cada bloco guarda o mínimo e o máximo de cada coluna. Um filtro que não pode ser satisfeito por um bloco faz o leitor pular o bloco inteiro. No arquivo de 100 mil registros da Seção 5, com 103 blocos:

```
$ 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
```

Listagem 4. No arquivo de 100 mil registros (ids de 0 a 99.999), as zonas pulam 102 de 103 blocos; no arquivo de produtos da Listagem 3, o `info` mostra o dicionário da coluna `categoria` e as zonas, com os reais em precisão dupla, como a ferramenta imprime.

Dicionário. Uma coluna `dic` guarda cada texto distinto uma vez por arquivo. Nos 100 mil registros do teste, as colunas `tipo` e `fonte` somam 10 textos distintos, e o `wjsw info` mostra `dicionario 10 textos`.

Índice por chave. O formato tem um índice que leva direto à linha de uma chave, sem varrer o arquivo. Na ferramenta pública 1.3.11 ele não tem comando nem função na ABI; o uso visível de chave é o `comparar`, que casa as linhas de dois arquivos por uma coluna e conta as diferenças:

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

3.3 Léxico (recurso 9)

Uma tabela de termos, com o esquema `termo:texto,forma:dic,sinal:dic,peso:f32,idioma:dic,origem:dic`, é compilada, ao abrir, num autômato de Aho-Corasick [21], que encontra todos os termos num texto numa passada só. O casamento é por palavra inteira, sem diferenciar acento nem maiúsculas, e devolve o trecho original e a soma dos pesos por sinal: é um classificador de texto sem treinamento, útil para triagem e para rotear um pedido ao especialista certo.

```
$ 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)
```

Listagem 5. "CANCELAR" casou com "cancelar", e "orcamento" casou com "orçamento", porque o casamento não diferencia acento nem maiúsculas.

3.4 Escrita e histórico (recursos 10 e 11)

O arquivo `.jsw` só anexa. Isso dá duas garantias. A primeira é a **gravação viva**: um processo pode ler o arquivo enquanto outro grava, como um `tail -f` que nunca vê um bloco pela metade. A segunda é o **histórico**: gravar de novo a mesma chave é uma edição, gravar a chave com `_apagado: true` é uma exclusão, e compactar regrava só o que vale num arquivo novo, deixando o histórico no original.

Na versão 1.3.11, a leitura durante a gravação, a visão "só o que vale" por chave e a compactação não têm comando no `wjsw` nem função na ABI pública. Com a ferramenta pública, o histórico aparece como ele é, linha após linha:

```
$ 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}
```

O que a ABI pública garante na escrita é a **publicação atômica**: o arquivo só aparece, inteiro, no `fechar()`. Quem sai por erro no meio chama `abandonar()`, e o destino fica como estava.

3.5 Permissão, custódia e acesso remoto (recursos 12 a 14)

A visão por permissão (12) e a custódia (13) são o assunto da Seção 4. A **leitura remota parcial** (14) usa o pedido HTTP com `Range` [24]: o leitor baixa o cabeçalho e o rodapé, onde estão o esquema, o índice dos blocos e as zonas, e depois só os blocos de que precisa (Figura 4). Medimos com o `wremoto` 1.2.1 contra um nginx local servindo o arquivo de 100 mil registros (o `com.jsw` da Seção 5, com o nome `cartoes.jsw`); com `file://` no lugar do endereço HTTP, o `wremoto` dá os mesmos números, sem servidor:

```
$ 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)
```

Listagem 6. Para ler uma linha do meio de um arquivo de 79,1 MB, o leitor baixou 0,98% dele; para o filtro, 0,84%, porque as zonas descartaram os outros blocos antes do download.

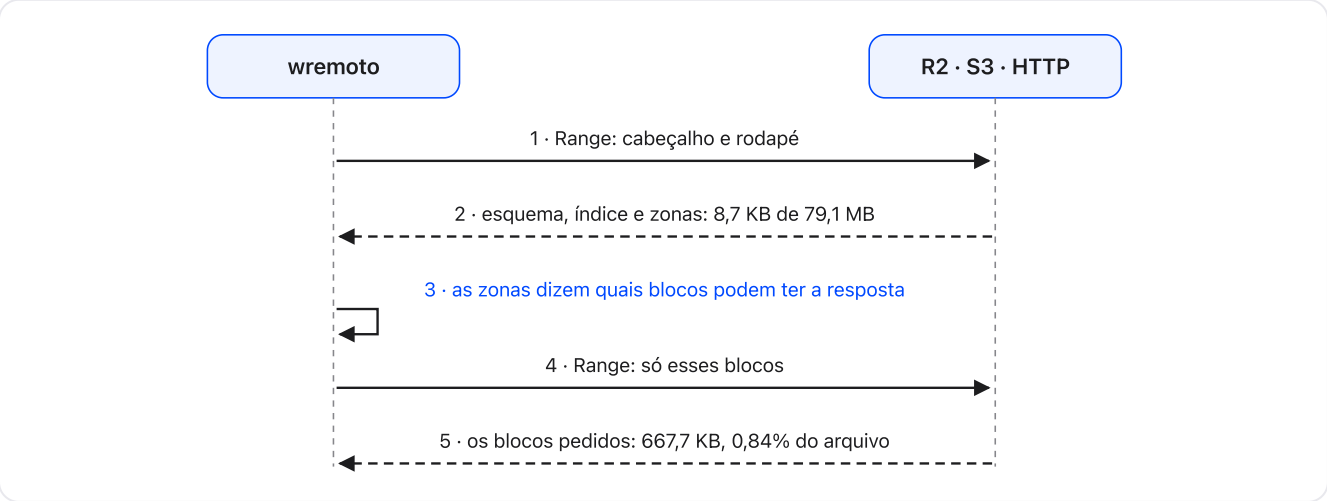


Figura 4. Leitura remota parcial do arquivo de 100 mil registros, com os números medidos na Listagem 6. Com o R2 ou o S3, cada pedido vai assinado em AWS SigV4 [25].

3.6 Grafia, sinais e coluna por bloco (recursos 15 a 19)

Grafia e som (recurso 15). O operador `~~` do filtro dá a cada valor uma nota de 0 a 1: a maior de duas medidas de Dice sobre trigramas de caracteres Unicode, uma na grafia normalizada (sem acento nem maiúsculas) e outra numa chave fonética da língua pedida. Há chave para o português do Brasil, o inglês

americano, o espanhol e o chinês escrito em pinyin; nos ideogramas, os trigramas contam caractere por caractere, e a chave do som deles, que depende de uma tabela de leituras, ainda não está nesta versão. O filtro não usa índice: varre só a coluna pedida, em paralelo (Listagem 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"}
```

Listagem 6B. Grafia e som parecidos. "Kokakola" acha "Coca-Cola": a chave do português leva o "c" duro e o "k" ao mesmo som. `~~en`, `~~es` e `~~zh` trocam a língua; `~~pt@0.4` muda o limiar.

Sinais da comunicação (recursos 16 a 18). Medem como a pessoa falou, escreveu ou apareceu, e não decidem nada: quem interpreta é o modelo que lê a linha. No áudio (PCM de 16 bits), o tom sai pela autocorrelação normalizada, o mesmo método do Praat; jitter e shimmer, pelos pulsos glotais; os formantes, por predição linear (LPC); e as pausas, pelo limiar de Otsu sobre a intensidade, que tira o corte dos próprios dados em vez de um número fixo. No texto, o léxico da língua (pt, en, es ou zh, escrito à mão) é casado pelo autômato do recurso 9, e as contas de forma (caixa alta, pontuação, emoji, risos) não usam dicionário. O chinês, que não separa palavras por espaço, é casado ideograma por ideograma. No vídeo, quem chama decodifica e entrega os quadros e o som; cada quadro vira uma miniatura com luz, contraste, cor e forma, e os cortes de cena saem pelo mesmo limiar de Otsu sobre a diferença entre quadros, só quando a série tem dois grupos claros. Cada medida sai como uma linha de jsonW-texto no esquema declarado, pronta para gravar num `.jsw`.

Coluna por bloco e por tenant (recurso 19). Três codecs novos: o dicionário local de cada bloco, com e sem codificação de entropia, para o texto que se repete dentro do bloco mas não no arquivo inteiro; e os dígitos de largura fixa, para códigos feitos só de algarismos, como o número de um processo. A zona de cada bloco guarda também o tenant, e, com a visão de uma sessão, o bloco de outro cliente nem é aberto. O filtro e o cálculo comparam o código do dicionário em lotes, com instruções vetoriais (SSE2), e avaliam o "contém" no bloco inteiro de uma vez. Um arquivo gravado com esses codecs não abre nos leitores 1.2.x. Na mesma versão, o nível máximo (`--max`) ganhou dois codecs que gravam cada célula e cada vetor como a diferença para a linha anterior: o prefixo e o sufixo comuns e só o miolo que muda. O efeito na base do INPI está nas Tabelas 5C e 5D.

4 Segurança e custódia

4.1 Custódia

Cada bloco do arquivo se sustenta sozinho e carrega duas conferências: um CRC32C [11], rápido, para dano acidental, e um hash criptográfico, SHA-256 [12] por padrão ou BLAKE3 [13] por opção. Os hashes formam uma cadeia que cobre o arquivo inteiro: qualquer byte trocado num dado, numa zona ou no dicionário falha no `conferir` (Figura 2).

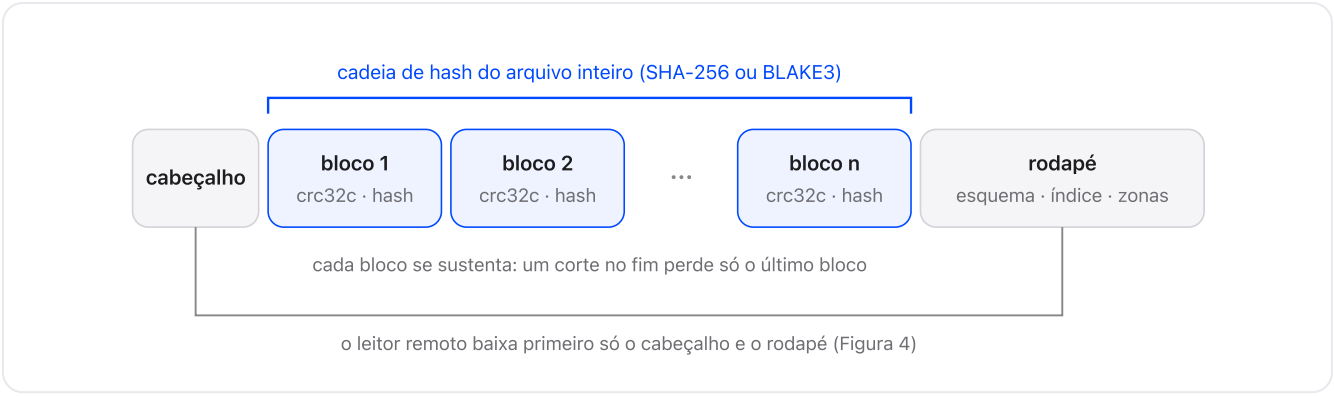


Figura 2. O arquivo `.jsw` visto de fora, no nível descrito na documentação pública. A organização interna do cabeçalho, dos blocos e do rodapé não é publicada (Seção 1.3).

```
$ wjsw conferir vendas.jsw
ok: 2 linhas, 1 blocos, sha256 e crc32c conferidos
# um byte trocado no meio do arquivo
$ wjsw conferir adulterado.jsw
FALHOU: bloco 0: nao veio ou nao confere
# os últimos 20 bytes cortados, como numa gravação interrompida
$ 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
# BLAKE3 no lugar do 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
```

Listagem 7. Custódia, adulteração e reparo, com a saída real.

> A custódia prova integridade, não autoria. Um hash sem chave pode ser recalculado por quem reescreve o arquivo inteiro. Para provar que um arquivo é o mesmo que saiu de uma fonte, guarde a identificação do arquivo que o `wjsw info` mostra num lugar separado, ou assine-a com uma ferramenta à parte, e compare.

4.2 Leitura de arquivo hostil

O leitor trata todo arquivo como hostil: confere cada índice e cada limite antes de usar, recusa aninhamento acima de 1000 níveis e é testado com *fuzzing* (entradas geradas ao acaso) na bateria de provas. Aspas, vírgulas, barras e qualquer byte dentro de um texto são dados e nunca quebram a leitura.

4.3 Visão por permissão

No jsonW a permissão é declarada no esquema e aplicada por quem serve o arquivo, antes de responder: o que a sessão não pode ver não sai do servidor (Figura 3, Tabela 3).

Tabela 3. As camadas da visão por permissão.

Camada	Como se declara	O que acontece
Coluna	`+escopo(financeiro)`, `+pessoal`	sem o escopo, a coluna sai nula e não pode ser usada nem em filtro
Linha	coluna inteira `tenant` ou `tenant_id`, ou marcada `+tenant`	só as linhas do tenant da credencial; blocos de outros tenants nem são abertos; `tenant:todos` libera; o tenant é um inteiro: com uma coluna `tenant` de texto, nenhuma linha sai
Célula	as duas juntas	a linha do próprio cliente, sem a coluna de dado pessoal

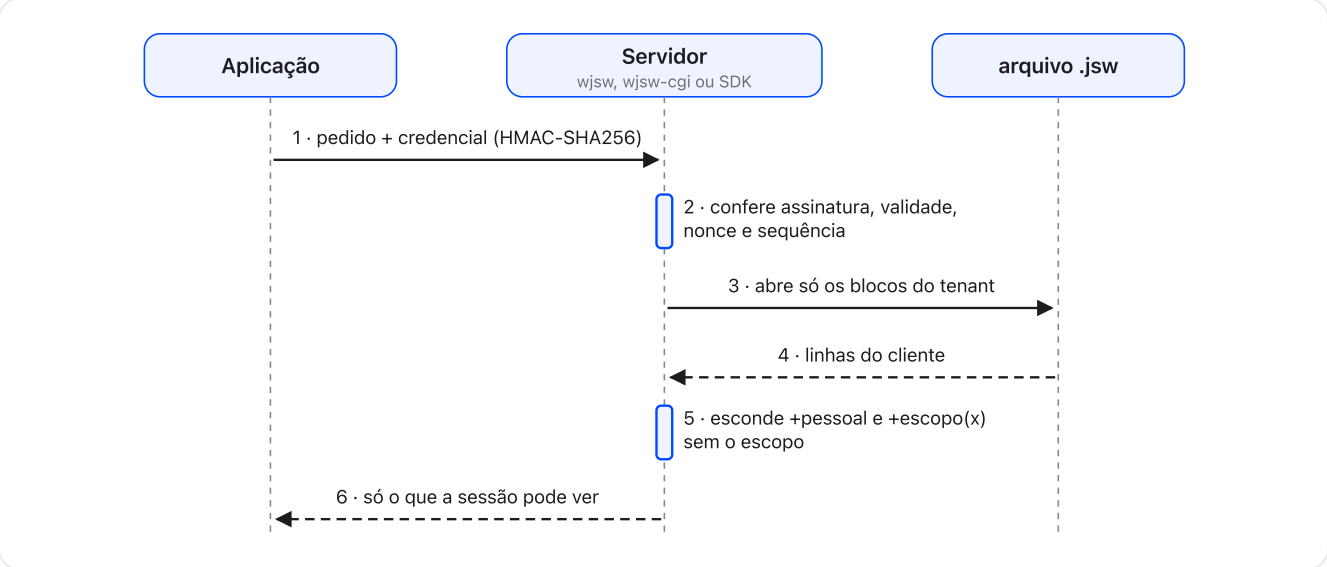


Figura 3. A visão por permissão. Os passos 2 a 5 acontecem dentro do servidor; o arquivo nunca sai inteiro para quem pediu.

O exemplo de uma clínica com duas unidades mostra as três camadas. A recepção da unidade 1 vê a agenda da unidade 1 sem CPF e sem valor; o financeiro da unidade 1 vê o valor, ainda sem CPF; ninguém da unidade 1 vê a unidade 2:

```
$ wjsw de-texto "paciente:texto,cpf:codigo?
+pessoal,valor:dinheiro+escopo(financeiro),data:tempo,tenant:i64" clin.jsonl clin.jsw
# recepção da unidade 1
$ 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}
# financeiro da unidade 1
$ 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}
```

Listagem 8. A mesma consulta, duas credenciais, duas respostas. Os dados do exemplo são fictícios.

> A visão não é criptografia. Ela protege o que sai do servidor; quem tem o arquivo `.jsw` em mãos tem todos os dados, e o `para-texto` exporta tudo, porque é a cópia de quem já é dono do arquivo. Para mostrar dados a um usuário, use o `ver`, o `wjsw-cgi` ou o SDK com `visao()`; para proteger o arquivo em repouso, use a criptografia do disco ou do armazenamento.

4.4 Credencial

A credencial que chega com o pedido é ela mesma um valor jsonW, com emissor, sujeito, tenant, escopos e validade, assinada com HMAC-SHA256 [18]. A assinatura é verificada em tempo constante, e um *nonce* com número de sequência impede que um pedido seja repetido. Ela trafega por cima do TLS, não no lugar dele.

4.5 Transmissão pós-quântica

A forma linha corre num protocolo de fio sobre QUIC [15] com TLS 1.3 [16] e troca de chaves híbrida X25519MLKEM768 [17], que combina a curva X25519 com o ML-KEM-768 padronizado pelo NIST na FIPS 203 [14]. O argumento pós-quântico tem duas partes. A troca de chaves usa um mecanismo feito para resistir a um computador quântico. E a custódia e a credencial usam só hash e HMAC de 256 bits, que não dependem de fatoração nem de logaritmo discreto; para achar uma pré-imagem ou a chave do HMAC, o algoritmo de Grover [19] dá ganho quadrático e deixa cerca de 128 bits de segurança; para achar colisões, o limite quântico conhecido é menor, cerca de 2^{85} operações no SHA-256 pelo algoritmo de Brassard, Høyer e Tapp [35], que pede uma memória quântica enorme.

A última linha do `wjsw medir` (Seção 5) mede a forma linha: um registro do teste ocupa 5.157 bytes em JSON e 1.073 bytes como mensagem, 4,8 vezes menos, em todas as execuções.

> O pacote público 1.3.11 não traz servidor nem cliente do fio.

4.6 Servidores web

O pacote traz o `wjsw-cgi`, que roda em Apache, nginx, LiteSpeed, Caddy e IIS [20]. Todo pedido de `.jsw` passa por ele: um arquivo sem coluna marcada e sem tenant sai sem filtro, com `Range` e CORS; os outros saem filtrados, com a visão obrigatória. O escopo vem do servidor depois de autenticar, nunca de um cabeçalho enviado pelo cliente, e pedir uma coluna escondida por `campos=` ou `onde=` dá 403, para não vazar por inferência. Na versão 1.2.0, a configuração de exemplo deixava o servidor entregar o arquivo cru quando o pedido vinha sem parâmetro; a 1.2.1 corrigiu isso, e quem instalou a configuração antiga deve trocá-la [20].

5 Medidas

Esta seção mede o jsonW 1.3.11 contra o JSONL e diz, junto, o que a medida não prova. Os números foram refeitos para este texto em 13 de novembro de 2026, com o binário do lançamento, na mesma máquina da medida de 12 de novembro.

5.1 Metodologia

- **Ferramenta:** `wjsw medir`, que está no binário público, com 100 mil linhas: `wjsw medir <pasta> 100000`. Binário `bin/linux-x86_64/wjsw` do repositório público, versão 1.3.11 (pré-lançamento, SHA-256 `cb119d7473e7d786…`).
- **Máquina:** servidor virtual com AMD EPYC 9355P, 8 vCPUs e 18 GB de memória, Ubuntu 26.10, kernel Linux 7.3. Os arquivos ficaram num `tmpfs` (memória), então a medida não inclui o disco. A máquina não foi isolada de outras cargas.
- **Dados:** 100 mil registros sintéticos, no formato de um cartão de base de conhecimento: `id:i64,tipo:dic,titulo:texto,corpo:texto,fonte:dic,valor:decimal[4],vetor:vf16[384],tags:lista<texto>`. Os textos repetem um modelo com o número da linha; `tipo` e `fonte` alternam entre 5 valores; o vetor tem 384 componentes múltiplos de 1/1024, que o `vf16` guarda sem perda.
- **O lado JSON:** o mesmo conjunto gravado em JSONL e lido pelo leitor de texto do próprio jsonW, que interpreta cada linha e converte os tipos do esquema. Os dois arquivos são carregados na memória antes

de o relógio começar.

- **Repetição:** 10 execuções seguidas. Reportamos a mediana de cada tempo e, para as razões, a mediana e a faixa das 10. As saídas brutas estão publicadas junto do texto.
- **Conferência:** em toda execução a ferramenta confere que as duas leituras contaram 100 mil linhas com a mesma soma.

5.2 Tempo

Tabela 4. Tempo de gravar e ler os 100 mil registros. Tempos: mediana de 10 execuções. Razão: JSONL ÷ jsonW em cada execução, com a mediana e a faixa das 10. Na linha de 8 threads, a razão é contra a leitura do JSONL em uma thread. O JSONL é gravado e lido pelo código de texto do próprio jsonW.

Operação	JSONL	jsonW	Razão (faixa)
Gravar	15.211 ms	131 ms	114× (88 a 127)
Ler tudo, todos os campos	3.687 ms	78 ms	46× (43 a 51)
Ler tudo, 8 threads	(sem versão paralela)	9,0 ms	419× (365 a 532)
Ler uma coluna (`valor`)	3.687 ms	1,59 ms	2.310× (1.592 a 2.784)
Ir à linha N, campo `titulo`	ler do começo	0,09 µs (0,05 a 0,16)	sem razão definida

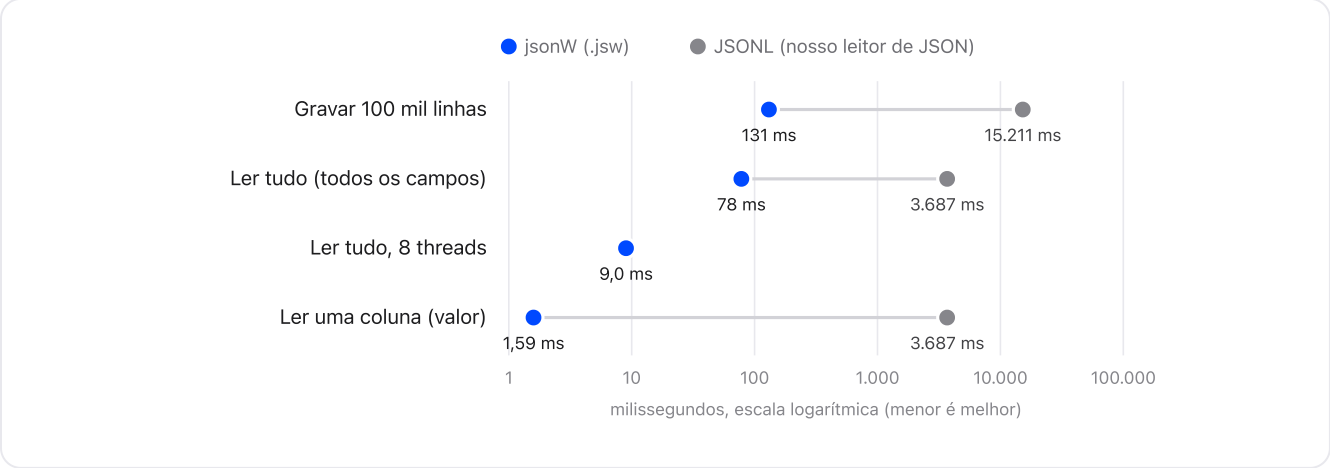


Figura 6. Os tempos da Tabela 4 em escala logarítmica (mediana de 10 execuções). A distância horizontal entre os pontos de cada linha é a razão.

A Figura 6 mostra os mesmos tempos em escala logarítmica. A maior diferença está em ler uma coluna: o JSONL obriga a interpretar cada linha inteira para tirar um campo, e o arquivo em colunas lê só aquela coluna. É uma vantagem estrutural, a mesma que qualquer formato em colunas tem sobre um formato em linhas. Em "ir à linha N", o JSONL não tem como saltar: é preciso ler desde o começo até a linha, então a razão depende de N e não a reportamos.

5.3 Tamanho

Para comparar o tamanho com compressores de uso geral, regravamos os mesmos 100 mil registros: um gerador em `awk`, publicado junto deste texto, produz o JSONL; o `wjsw de-texto` grava o `.jsw`; e o JSONL de volta, pelo `wjsw para-texto --json`, tem 519.264.563 bytes, o mesmo tamanho, na escala de MB, que o `wjsw medir` reporta para o seu JSONL (519,3 MB); o `.jsw` tem 79.076.300 bytes (79,1 MB nos dois). O JSONL do gerador tem 200 mil bytes a mais, porque nele o valor decimal vem entre aspas; a Tabela 5 usa o

JSONL regravado pelo jsonW, e com o do gerador as razões mudam na primeira casa decimal (gzip: 19,7× em vez de 19,8×). O JSONL foi comprimido com gzip 1.14 [32], zstd 1.5.7 [31] e xz 5.8.3 [33] nos níveis máximos usuais.

Tabela 5. Tamanho dos 100 mil registros, em MB (10^6 bytes), com e sem a coluna de vetores. Razão = JSONL sem compressão ÷ o tamanho da linha.

Arquivo	Com vetor	Razão	Sem vetor	Razão
JSONL	519,3	1,0×	38,2	1,0×
`.jsw`, vetor `vf32`	155,9	3,3×	n/a	n/a
`.jsw`, vetor `vf16`, nível padrão	79,1	6,6×	2,22	17,2×
`.jsw`, vetor `vq8` (com perda)	41,1	12,6×	n/a	n/a
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`, vetor `vf16`, `--max`	1,36	381,1×	0,0175	2.185×

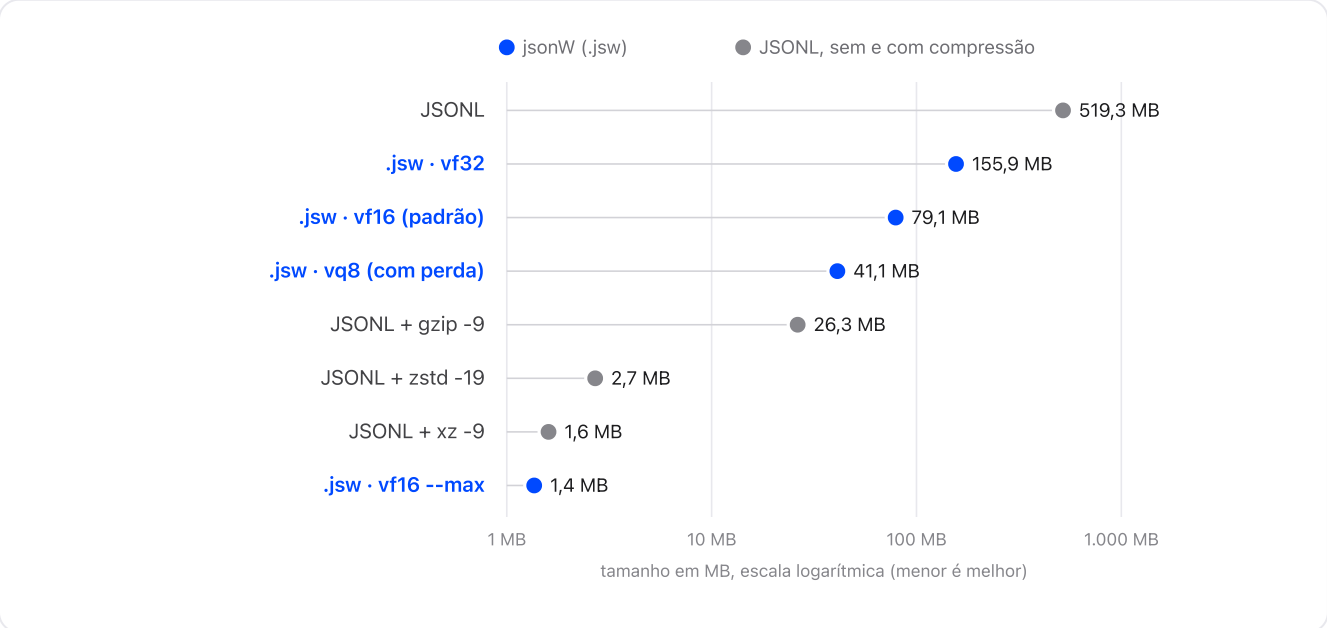


Figura 5. Os tamanhos com vetor da Tabela 5, em escala logarítmica.

A Figura 5 mostra os tamanhos com vetor. Contra o JSONL sem compressão, que é como o JSONL costuma ser guardado e trocado, o `.jsw` padrão é 6,6 vezes menor, e quase todo o seu tamanho é o vetor: 384 componentes de 2 bytes em 100 mil linhas são 76,8 MB. Contra o JSONL comprimido com zstd ou xz, o `.jsw` padrão é maior. No nível máximo, a 1.3.11 ficou menor que os dois: 1,36 MB com vetor, contra 2,67 MB do zstd e 1,56 MB do xz, e 17,5 kB sem vetor, contra 0,63 e 0,47 MB; na 1.2.1, o mesmo nível dava 3,75 e 2,19 MB. O ganho vem de dois codecs novos do nível máximo, que gravam cada célula e cada vetor como a diferença para a linha anterior (o prefixo e o sufixo comuns e o miolo que muda), e é grande aqui porque os dados do teste são muito repetitivos: os textos se repetem com uma troca de número e o vetor segue um padrão aritmético. A volta é sem perda: o `para-texto` do arquivo no nível máximo é igual, byte a byte, ao do arquivo no nível padrão (mesmo SHA-256, com e sem vetor). Em dados reais o ganho é bem menor: na base

do INPI, o nível máximo da 1.3.11 ficou 16,5% menor que o da 1.2.1 (Tabela 5C). Vetores de modelos reais também tendem a comprimir pouco; não medimos isso aqui, mas, se for assim, o tamanho do `.jsw` com vetor fica perto do tamanho bruto do vetor.

O que o arquivo comprimido de uso geral não dá é acesso. Para ler um campo do JSONL com zstd, é preciso descomprimir tudo (de 0,13 s a 0,24 s para descomprimir o `com-volta.jsonl.zst`, em 3 execuções na mesma máquina, fora do ambiente da medida) e depois interpretar o texto, enquanto o `.jsw` lê o campo direto, pula blocos pelas zonas, confere a custódia e pode ser lido de longe por partes.

5.4 Dados reais: CSV, XML, XLSX e Parquet

As medidas acima usam dados gerados. Para comparar com os formatos de planilha e de troca mais comuns, usamos dados públicos: o primeiro milhão de linhas do arquivo `MARCAS_DADOS_BIBLIOGRAFICOS.csv`, dos dados abertos de marcas do INPI (13 colunas: número do processo, datas, apresentação, natureza, nome da marca, situação). Um programa em C, sem bibliotecas, leu o CSV seguindo a RFC 4180 (aspas, vírgula e quebra de linha dentro do campo) e gravou as mesmas linhas em quatro formas: o CSV regravado; um XML com um elemento por linha e um filho por coluna; um XLSX montado pelo próprio programa (Office Open XML, células de texto em linha, compactado com `zip -6`); e o `.jsw`, com o esquema escrito à mão (datas como texto, apresentação, natureza e situação como dicionário). Campo vazio fica ausente no XML e no jsonW. O XLSX aceita até 1.048.576 linhas por planilha, por isso um milhão.

A consulta foi a mesma nos quatro: contar os registros em vigor (`codigo_situacao = IPAS023`). As quatro formas deram o mesmo resultado, 352.248 registros, e o `wjsw conferir` conferiu o SHA-256 e o CRC32C de todos os blocos do `.jsw`. Cada consulta rodou 5 vezes, com o arquivo já em cache; vale a menor. Máquina: servidor de 8 núcleos com Linux 7.3, jsonW 1.2.0, 13 de novembro de 2026.

Tabela 5B. Um milhão de linhas reais do INPI em quatro formatos. Tamanho em MB (10^6 bytes); consulta = contar os registros em vigor, em milissegundos.

Formato	Tamanho	Com zstd -3	Consulta	Como a consulta foi feita
XML	502,1	46,0	159	`grep -c` no arquivo inteiro
CSV	141,1	36,1	97	`grep -c` no arquivo inteiro
XLSX	74,3	n/a	9.364	`unzip -p` da planilha + `grep`
`.jsw`	51,2	n/a	119	`wjsw calcular --onde`, só a coluna

Em tamanho, o `.jsw` ficou 2,8 vezes menor que o CSV, 1,5 vez menor que o XLSX e 9,8 vezes menor que o XML. O CSV e o XML comprimidos com zstd ficam menores que o `.jsw` (36,1 e 46,0 MB contra 51,2 MB), mas, como na Seção 5.3, precisam ser descomprimidos inteiros antes de qualquer leitura. Na consulta, o `.jsw` foi 79 vezes mais rápido que o XLSX, que precisa ser descompactado a cada leitura. Contra o CSV e o XML, o `grep` foi mais rápido ou parecido: ele só procura um trecho de texto no arquivo inteiro, sem separar colunas nem tipos, e daria resultado errado se o mesmo código aparecesse em outra coluna. Neste conjunto não aparece, e as contagens bateram. Uma consulta com a mesma garantia no CSV exigiria ler e separar cada linha, o que o `awk` fez em 1.597 ms na primeira rodada desta medida, sem tratar aspas.

O que nenhum dos três formatos traz pelo próprio formato: esquema com tipos (data, decimal, dinheiro, vetor), leitura de uma coluna sem ler as outras, busca por termos e por significado, visão por permissão, histórico de linhas apagadas e custódia por bloco. O XLSX tem tipos de célula e várias planilhas, mas limita o tamanho da planilha e exige descompactar para ler.

A base inteira. A Tabela 5B usa um milhão de linhas de um arquivo. Para ver o formato com tudo, convertemos os sete arquivos dos dados abertos de marcas do INPI [38], baixados em 12 de novembro de 2026, sem cortar nada: 55.278.212 linhas e 12,53 GB de CSV, com todas as colunas e o histórico completo de despachos de cada processo (24.978.976 linhas só no `MARCAS_DESPACHOS.csv`). Cada arquivo ganhou um esquema escrito à mão. Número, data e código ficaram com o próprio tipo; texto, os bytes como vieram. O conversor, em C, confere cada valor tipado na hora: o número e a data precisam voltar ao mesmo texto do CSV, senão a conversão para. Isso pegou dois casos reais da fonte, que foram guardados como vieram: uma data de prioridade no ano 1018 (erro de digitação no original) e uma linha da classificação de Nice sem número de processo, das 5.333.893 do arquivo.

O Parquet [3] foi gravado pelo DuckDB 2.0.0 [39] com os mesmos tipos e compressão zstd. O XML segue o desenho da Tabela 5B. O XLSX ficou de fora: seis dos sete arquivos passam do limite de 1.048.576 linhas por planilha, e o de despachos tem quase 24 vezes esse limite. As oito formas deram as mesmas contagens nas três consultas, com uma exceção, dita abaixo. O `wjsw conferir` conferiu o SHA-256 e o CRC32C de todos os blocos dos sete `.jsw`. Cada consulta rodou 5 vezes, com o arquivo já em cache, e vale a menor. A gravação do jsonW usou um núcleo. O DuckDB, o zstd e o xz usaram os oito núcleos, e o `wjsw calcular` também. Máquina: a mesma da Tabela 5B, 31 GB de memória, 13 de novembro de 2026.

Tabela 5C. A base inteira de marcas do INPI (sete arquivos, 55,3 milhões de linhas) em cada formato. Tamanho em MB (10^6 bytes), somado dos sete arquivos; gravar = tempo para produzir o formato a partir do CSV, em segundos.

Formato	Tamanho	Vezes menor que o CSV	Gravar	Lê uma coluna sem abrir o resto?
XML	24.804,7	0,5	n/a	não
CSV (original)	12.527,4	1,0	n/a	não
`.jsw` , 1.2.1	2.804,8	4,5	176,7	sim
`.jsw` , 1.3.11	2.579,2	4,9	n/a	sim
`.jsw` , 1.2.1, `--max`	2.321,8	5,4	184,9	sim
XML + zstd -3	2.210,0	5,7	21,4	não, descomprime tudo
`.jsw` , 1.3.11, `--max`	1.938,1	6,5	n/a	sim
CSV + zstd -3	1.908,9	6,6	13,2	não, descomprime tudo
Parquet (zstd)	1.665,8	7,5	16,6	sim
CSV + xz -6	1.161,0	10,8	544,5	não, descomprime tudo

Tabela 5D. Três consultas por coluna na base inteira, em milissegundos (o menor de 5), medidas pela quinta vez em 13/11/2026 às 06:21 UTC, com o `wjsw` 1.3.11 do lançamento e o 1.2.1, na mesma hora. Em vigor: contar `codigo_situacao = IPAS023` em 6,7 milhões de processos. Histórico: todos os despachos de um processo (`numero_inpi = 818602856`) em 25,0 milhões. Classe 35: contar `classe_nice = 35` em 5,3 milhões. Entre parênteses, o DuckDB com um núcleo. O XML vem da primeira rodada, porque o arquivo foi apagado depois dela (ocupava 24,8 GB).

Formato e leitor	Em vigor	Histórico	Classe 35
Resposta certa	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 lendo o CSV	375 (1.447)	1.205 (4.523)	592 (1.979)
CSV, `grep -c`	519	6.690	1.278 (errada: 1.076.929)
CSV + zstd, `zstd -dc | grep -c`	798	3.129	2.786 (errada)
CSV + xz, `xz -dc | grep -c`	1.050	3.025	2.105 (errada)
CSV + xz, `xz -dc | grep -c`	971	2.628	1.950 (errada)
XML, `grep -c` (primeira rodada)	3.017	9.925	3.430

O que a base inteira mostra. Em tamanho, o `.jsw` 1.3.11 ficou 4,9 vezes menor que o CSV no nível padrão e 6,5 vezes no máximo, e 12,8 vezes menor que o XML; o 1.2.1 ficava 4,5 e 5,4 vezes menor que o CSV, e o 1.3.11 no máximo ficou 16,5% menor que o 1.2.1. Na consulta, com oito núcleos dos dois lados, o 1.3.11 contou os registros em vigor em 43 ms no nível máximo e 44 ms no padrão, à frente dos 54 ms do Parquet lido pelo DuckDB; com um núcleo, o DuckDB em Parquet levou 76 ms nessa contagem e 521 ms no histórico. Nas três consultas, o `.jsw` foi mais rápido que o DuckDB lendo o próprio CSV, que o `grep`, que o CSV comprimido e que o XML. No histórico de um processo e na classe 35, o `.jsw` levou 207 e 142 ms, e o Parquet com oito núcleos, 129 e 41 ms: é o próximo passo de desempenho da Seção 8. Em tamanho, o Parquet (1.665,8 MB) e o CSV comprimido com xz (1.161,0 MB) ocupam menos que o `.jsw` no máximo (1.938,1 MB), e o CSV comprimido precisa ser descomprimido inteiro antes de qualquer leitura; com o zstd, o CSV ficou praticamente igual ao `.jsw` no máximo (1.908,9 MB). A gravação do `.jsw` 1.2.1, em um núcleo, levou 177 s; a do Parquet, com oito, 17 s. Os arquivos 1.3.11 foram regravados a partir do `.jsw` 1.2.1 (`wjsw regravar`), por isso a Tabela 5C não traz o tempo de gravação deles.

As rodadas e as correções. A Tabela 5D foi medida cinco vezes, sempre com o Parquet e o CSV na mesma hora. Com o `wjsw` 1.2.0, o `.jsw` levou 677, 1.494 e 1.114 ms nas três consultas, de 10 a 31 vezes o tempo do Parquet: o filtro lia todas as colunas de cada linha. Corrigido para ler só a coluna perguntada, levou 82, 311 e 209 ms. Na terceira rodada, a avaliação passou a ser feita por bloco, comparando o código do dicionário e os inteiros direto na coluna e montando a linha só quando ela passa: 48, 344 e 186 ms. Na quarta, com o 1.3.11 ainda em preparação, o histórico caiu de 355 para 279 ms. Na quinta, com o binário do lançamento, que compara o dicionário em lotes com instruções vetoriais e avalia o "contém" por bloco, o 1.3.11 levou 44, 254 e 152 ms no nível padrão e 43, 207 e 142 ms no máximo, contra 54, 353 e 198 ms do 1.2.1 na mesma hora. O histórico procura o número do processo, um código só de algarismos que o 1.3.11 grava com largura fixa (em DADOS_BIBLIOGRAFICOS, só essa coluna caiu de 53,0 para 25,7 MB). A máquina não estava isolada, e o mesmo binário variou até 23% de uma rodada para a outra (o Parquet, de 41 a 54 ms na contagem em vigor);

por isso a comparação vale dentro da mesma rodada. Um arquivo gravado com os codecs novos não abre nos leitores 1.2.x. Os tempos do `grep` no CSV variaram ainda mais (de 2.102 a 6.690 ms no histórico), porque os 12,5 GB de CSV não cabem todos no cache ao lado dos outros arquivos, e na quinta rodada as regravações logo antes tiraram o CSV do cache; vale o número da mesma rodada.

A exceção das contagens é a da classe 35. O `grep` no CSV, puro ou comprimido, contou 1.076.929, um a mais que a resposta certa: o trecho `,35,` também aparece em outra coluna de uma linha. Foi o risco descrito junto da Tabela 5B, que naquele conjunto não aconteceu e aqui aconteceu. Quem lê o arquivo com as colunas separadas (o jsonW, o Parquet, o DuckDB lendo o CSV e o XML com o nome da coluna na marcação) deu 1.076.928.

Desta conversão 1:1, o único recurso além do tipo e da coluna que entrou foi a custódia: cada bloco dos sete arquivos tem CRC32C e SHA-256, conferidos acima. A conversão não marcou permissão nem montou busca, para ficar igual ao CSV. O formato permite, no mesmo arquivo, esconder o CPF e o CNPJ dos titulares de quem não tem o escopo (Seção 4.3), ler só uma parte de longe e buscar por termos e por significado (Seção 3). O Parquet tem uma soma de verificação opcional por página [36] e criptografia por coluna [5], mas não a cadeia de hashes do arquivo inteiro nem a visão por escopo; o CSV e o XML não trazem nada disso.

5.5 Limitações da medida

- **O leitor de JSON é o nosso.** O lado JSON foi gravado e lido pelo código de texto do próprio jsonW, que valida e converte tipos; ele lê o JSONL a cerca de 140 MB/s. Leitores de JSON de ponta, como o simdjson [23], passam de 1 GB/s. Com um deles, as linhas "gravar" e "ler tudo" da Tabela 4 encolheriam bastante. Numa estimativa, não numa medida: a 2 GB/s, ler os 519 MB levaria uns 260 ms, e a razão de "ler uma coluna" ficaria na casa das centenas, não dos milhares.
- **Dados sintéticos e repetitivos,** como dito na Seção 5.3. A razão de tamanho depende muito dos dados, e mais ainda no nível máximo, que guarda a diferença entre linhas seguidas: meça com os seus.
- **Uma máquina, um sistema, memória em vez de disco.** Os tempos absolutos mudam com o processador, o armazenamento e a carga; entre as 10 execuções, a maior razão de cada operação ficou de 19% a 75% acima da menor; a de ler uma coluna é a que mais varia, porque o tempo do jsonW ali é de 1,6 ms.
- **Parquet medido só com dados do INPI e um leitor.** A comparação de desempenho com o Parquet está nas Tabelas 5C e 5D, com o DuckDB 2.0.0 como gravador e leitor. Outro leitor, outra compressão ou outro tamanho de grupo de linhas mudam os números. Arrow, MessagePack, CBOR e Protobuf seguem sem medida de desempenho nesta versão do texto; a Seção 6 compara recursos, não velocidade.

5.6 Números publicados antes

Até 12 de novembro de 2026, o site e o repositório do jsonW citavam "4,9 vezes menor que o JSONL" e "3.281 vezes mais rápido para ler um campo", da primeira medida, de 6 de novembro, com a versão 1.0 em outra máquina. O texto 1.0 deste paper, de 12 de novembro, refez a medida com a 1.2.1: 6,6 vezes menor e cerca de 2.500 vezes mais rápido para ler uma coluna, com o nível máximo em 3,75 MB com vetor e 2,19 MB sem. Refeita para este texto com a 1.3.11, na mesma máquina, dá 6,6 vezes menor e cerca de 2.300 vezes mais rápido para ler uma coluna (a diferença está dentro da variação entre execuções), e o nível máximo cai para 1,36 MB e 17,5 kB. Valem os números deste texto, com a data, a máquina e a versão ao lado.

5.7 Como reproduzir

```
git clone https://github.com/developer-wortic/jsonw && cd jsonw
export JSONW_SEM_PING=1                       # sem o aviso de instalação
W=bin/linux-x86_64/wjsw                        # ou bin/macos-arm64/wjsw
for i in $(seq 1 10); do $W medir /tmp 100000; done

# Tabela 5: o mesmo conjunto em JSONL, o .jsw e os compressores
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*
```

Listagem 9. As medidas deste texto, na sua máquina. `VET=0` e o esquema sem a coluna `vetor` dão a coluna "sem vetor".

6 Comparação com outros formatos

A Tabela 6 compara o que cada formato faz pelo próprio formato, sem bibliotecas por cima. Os formatos em colunas mais maduros, Parquet e Arrow, já têm boa parte das vantagens de acesso que medimos na Seção 5, e várias coisas que o jsonW ainda não tem.

Tabela 6. O que cada formato traz no próprio formato. Fontes nas referências indicadas em cada coluna.

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Forma	texto; uma linha por registro no JSONL	binária, sem esquema	binária, esquema compilado (`.proto`)	arquivo em colunas	colunas em memória; arquivo e fluxo IPC	linha binária, arquivo em colunas e texto
Esquema	não (JSON Schema à parte)	não	sim, fora da mensagem	sim, no arquivo	sim	sim, uma linha, no arquivo
Ler um campo sem ler o resto	não	não	não	sim, por coluna	sim, acesso direto	sim, por linha e campo
Decimal exato	não	CBOR: sim (fração decimal, tag 4)	não nativo	sim (DECIMAL)	sim (decimal128 e 256) [37]	sim
Dinheiro e medida	não	não	dinheiro por mensagem comum (`google.type.Money`) [10]	não	não	sim, com moeda e unidade
Vetor denso	lista de números em texto	lista; CBOR tem vetores tipados [34]	campo repetido	lista ou bytes de tamanho fixo	lista de tamanho fixo	f32, f16, 8 e 4 bits, bits; tensor
Permissão	não	não	não	criptografia por coluna, com chaves diferentes [5]	não	visão por coluna, linha e célula, aplicada por quem serve; sem criptografia
Integridade	não	não	não	CRC opcional por página [36]; criptografia autenticada	não no formato	crc32c e hash por bloco, em cadeia
Busca e ranking no formato	não	não	não	estatísticas por grupo e página para pular dados	não	zonas, cosseno, esparso, grade, top-k, léxico
Leitura remota parcial	não	não	não	sim (rodapé e Range)	sim, no formato de arquivo	sim
Especificação	aberta (RFC)	aberta (CBOR: RFC)	aberta	aberta (Apache)	aberta (Apache)	uso, texto e ABI públicos; leiaute binário interno

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Implementações	muitas, independentes	muitas, independentes	várias	várias, independentes	várias, independentes	uma (núcleo em C), com SDKs em 10 linguagens

Onde cada formato se encaixa. JSON é universal e legível em qualquer lugar. MessagePack, CBOR e Protobuf são maduros e implementados em dezenas de linguagens, e o Protobuf traz regras maduras de evolução de esquema. Parquet e Arrow são padrões abertos, lidos por quase todo motor de análise, e o Parquet tem criptografia por coluna para o arquivo em repouso. O jsonW conversa com todos eles: todo JSON já é jsonW-texto, o `wcon` traz e devolve tabelas de bancos com os tipos certos, e o arquivo em repouso pode ser cifrado pelo disco ou pelo armazenamento.

O que o jsonW junta. A diferença do jsonW não está em um recurso isolado, e sim em ter, no mesmo modelo, a forma de mensagem, a de arquivo em colunas e a de texto compatível com JSON; os tipos de negócio conferidos ao gravar; a permissão por coluna, linha e célula aplicada na leitura; e as operações de ranking, busca e classificação dentro do arquivo. O custo dessa escolha é ser, por enquanto, uma implementação só, com o núcleo fechado (Seção 8).

7 Usos

7.1 Modelos de IA e busca por significado

Uma base de conhecimento cabe num `.jsw` com texto, metadados e o vetor de cada trecho; a busca por significado é um `avaliar … cosseno` que lê só a coluna de vetores, e o léxico e o vetor esparso cobrem a busca por termos. Com as k melhores e o léxico, o mesmo arquivo serve de roteador de especialistas: o texto do pedido é classificado pelo léxico, e o ranking escolhe o especialista.

7.2 Dados de empresas

Cadastros como o de CNPJ e o de CEP são o caso em que o JSON mais perde: o zero da frente some quando o código vira número, o capital social vira *double* e a data perde o fuso. No jsonW, cada coisa tem o seu tipo:

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

Listagem 10. Um esquema de cadastro de empresas: código byte a byte, texto repetido em dicionário, dinheiro com moeda e o tenant para separar clientes.

7.3 Bancos de dados

O `wcon`, o conector de bancos do pacote, leva dados do PostgreSQL, do MySQL e MariaDB, do SQLite, do Redis e Valkey e do MongoDB para um `.jsw` e de volta, com o esquema tirado do tipo de cada coluna: `numeric` vira decimal exato, `bigint` continua inteiro de 64 bits e o `vector` e o `halfvec` do pgvector [30] viram `vf32` e `vf16`. O valor viaja sempre como dado, com o tamanho na frente, e nunca montado dentro de um comando; a volta roda numa transação única. A ida e volta foi conferida com o PostgreSQL 14, 16 e 18 (md5 igual nos dois lados), o MariaDB 11.8 (hex de cada coluna), o SQLite, o Redis 7 e o MongoDB 7, com carga hostil: injeção, aspas, `\0`, acentos, emoji e texto da direita para a esquerda [20].

7.4 Servidores web

Com o `wjsw-cgi` (Seção 4.6), um site entrega um `.jsw` como JSON, NDJSON ou o próprio arquivo, com filtro, escolha de campos e limite pelo endereço, sempre com a visão da sessão.

7.5 Armazenamento em R2 e S3

O `.jsw` pode ficar num bucket do Cloudflare R2 ou de qualquer S3 e ser consultado de lá, sem baixar o arquivo inteiro (Figura 4). Como o R2 não cobra a saída de dados [26], consultar um pedaço de um arquivo grande custa o pedaço lido mais as operações, que o R2 cobra por pedido (a Listagem 6 faz 5 pedidos).

8 Limitações e próximos passos

- **Uma implementação, núcleo fechado.** O leiaute binário não é publicado; ler e gravar `.jsw` depende da `libjsonw`. A ABI estável e os SDKs reduzem o risco, mas não substituem uma especificação aberta com implementações independentes.
- **Sem auditoria independente.** As provas citadas (fuzzing, cargas hostis, comparação byte a byte entre SDKs) são nossas.
- **Plataformas.** Há binários para macOS arm64, Linux x86_64 (glibc 2.34 ou mais nova) e Windows x86_64. Ainda não há para macOS Intel, Linux arm64 nem Linux com musl. No Windows, `wcon` e `wremoto` rodam pelo WSL.
- **Recursos sem comando público.** Índice por chave, gravação viva, visão "só o que vale" e compactação existem no núcleo, mas não têm comando no `wjsw` nem função na ABI 1.3.11; o formato da tabela da grade não está documentado.
- **Fio fora do pacote.** O protocolo de fio pós-quântico não vem no pacote público 1.3.11.
- **Palavras em português.** Os tipos e os comandos usam palavras em português (`texto`, `dinheiro`, `medida`, `ver`, `conferir`), o que pede um glossário para quem não lê português.
- **Bancos.** O MySQL 8 entra só com `mysql_native_password`; a primeira autenticação completa do `caching_sha2_password` ainda não existe.
- **Medidas.** As limitações da Seção 5.5.
- **Consulta por coluna: o próximo passo.** Na base inteira do INPI (Tabela 5D), o `wjsw calcular` 1.3.11 conta os registros em vigor em 43 ms, à frente do DuckDB lendo Parquet, e desde a primeira rodada (1.2.0) essa contagem caiu de 677 para 43 ms. O próximo passo é levar a mesma leitura por bloco e em lotes ao histórico de um processo e à classe 35, em que o `.jsw` levou 207 e 142 ms e o DuckDB com oito núcleos, 129 e 41 ms. No tamanho, os próximos codecs (diferença e empacotamento de bits em inteiros e datas) miram os 1.938,1 MB do nível máximo.

Próximos passos deste texto e do pacote: binários para macOS Intel, Linux arm64 e musl; uma consulta que confira só o CRC32C de cada bloco, deixando o SHA-256 para o `conferir` (o Parquet não confere hash na leitura, e parte da diferença que sobra vem daí); o codec de dicionário com janela, para colunas como a classe de Nice; a codificação por diferença com empacotamento de bits em inteiros e datas, medida de novo contra as Tabelas 5C e 5D; e a comparação de desempenho com Arrow na mesma máquina.

9 Disponibilidade e licença

A ferramenta, a biblioteca, a ABI, os SDKs, as configurações de servidor web e a documentação estão em github.com/developer-wortic/jsonw [20], e o SDK de Node.js no npm, como `@worticteconologia/jsonw`. O uso é regido pela Licença de Uso Sustentável do jsonW, que vale no lugar deste resumo: é grátis usar o jsonW em quantos servidores e aplicativos for preciso, inclusive em produtos comerciais, desde que ele seja um

componente e não o próprio serviço vendido; sem acordo por escrito com a Wortic, não se pode vender, hospedar ou oferecer a terceiros o jsonW ou um serviço cuja função principal seja a dele (armazenar, converter, servir ou transportar dados em jsonW), com ou sem cobrança; não se pode fazer engenharia reversa das bibliotecas e ferramentas distribuídas em binário, salvo o que a lei garantir; e o leiaute interno do arquivo, os codecs, o protocolo de fio e o núcleo são da Wortic, de modo que uma implementação independente deles também pede acordo por escrito. A documentação pública pode ser lida, citada e usada livremente para criar programas que usem o jsonW.

Por transparência: na primeira vez que roda em cada máquina, a ferramenta `wjsw` envia só a versão e a plataforma (por exemplo `v=1.3.11&so=linux-x86_64`) para contarmos as instalações, e não repete depois. O IP chega ao servidor com a conexão, mas não é guardado, nem nome, caminho ou dado: o servidor anota só o dia, a plataforma e a versão. `JSONW_SEM_PING=1` ou o arquivo `~/.jsonw/sem-ping` desligam o aviso antes da primeira execução. A biblioteca e os SDKs nunca acessam a rede.

Referências

[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)*, que define o 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: repositório público, documentação (docs/JSONW.md), ABI (sdk/jsonw.h) e binários*. 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*, seção 14 (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* (campo `crc` do `PageHeader`). github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift

[37] Apache Software Foundation. *Arrow Schema.fbs* (tipo Decimal, larguras de 128 e 256 bits). github.com/apache/arrow/blob/main/format/Schema.fbs

[38] Instituto Nacional da Propriedade Industrial (INPI). *Dados abertos de marcas* (`marcas.zip`, sete arquivos CSV), baixado em 12/11/2026. dadosabertos.inpi.gov.br/download/marcas/marcas.zip

[39] DuckDB Foundation. *DuckDB 2.0.0*, leitura de CSV e Parquet. duckdb.org/docs

Como citar

Wortic Tecnologia. *jsonW: um modelo de dados, três formas sem perda e a permissão dentro do formato*. Texto 1.1, sobre o jsonW 1.3.11. 13 nov. 2026. Disponível em: https://jsonw.org/paper.

```
@techreport{jsonw2026,
  title       = {jsonW: um modelo de dados, tr{\^e}s formas sem perda
                 e a permiss{\~a}o dentro do formato},
  author      = {{Wortic Tecnologia}},
  institution = {Wortic Tecnologia},
  year        = {2026},
  month       = nov,
  note        = {Texto 1.1, sobre o jsonW 1.3.11},
  url         = {https://jsonw.org/paper}
}
```

Texto 1.1, publicado em 13/11/2026, sobre o jsonW 1.3.11 (o texto 1.0, de 12/11/2026, tratava do 1.2.1). jsonW é o json Wondrous; o núcleo em C é mantido pela Wortic Tecnologia. Correções e medidas de terceiros são bem-vindas pelo repositório público.