Artículo técnico · jsonW 1.3.11

jsonW: un modelo de datos, tres formas sin pérdida y los permisos dentro del formato

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

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

ÍNDICE

RESUMEN

JSON y JSONL se convirtieron en el idioma común de los datos, pero fueron diseñados para intercambiar mensajes entre programas, no para guardar y consultar el volumen y la variedad de datos que usan los sistemas de IA. Leer un campo de un archivo JSONL obliga a interpretar el archivo desde el principio; el dinero, las medidas y los decimales exactos no tienen tipo; los vectores se vuelven texto; los permisos y la prueba de integridad quedan fuera del dato. Este artículo describe jsonW (json Wondrous) 1.3.11, un modelo de datos con tres formas que se convierten sin pérdida: fila, para mensajes y tiempo real; columna, el archivo `.jsw` en bloques leído mediante `mmap` ; y texto, en el que todo JSON válido también es jsonW-texto válido. El formato trae tipos de negocio, vectores y tensores, diecinueve recursos integrados (búsqueda, ranking, mantenimiento, grafía parecida y señales de audio, texto y vídeo), vistas por permiso y custodia por hash. Con 100.000 registros sintéticos, en una máquina, el archivo `.jsw` ocupó 6,6 veces menos que el JSONL sin compresión y leyó una columna unas 2.300 veces más rápido que nuestro propio lector de JSON. Con la base real de marcas del INPI, 55,3 millones de filas y 12,53 GB de CSV, el `.jsw` quedó 6,5 veces más pequeño que el CSV y 12,8 veces más pequeño que el XML, y contó los 2,3 millones de registros vigentes en 43 ms, más rápido que Parquet leído por DuckDB.

Palabras clave: formato de datos; almacenamiento en columnas; JSON; búsqueda vectorial; control de acceso; integridad; intercambio de claves poscuántico.

1 Introducción

JSON [1] nació para intercambiar mensajes entre programas en un texto que la gente puede leer. JSON Lines [2] llevó la idea a los archivos: un registro por línea. Ambos se volvieron estándares de hecho para logs, exportaciones, conjuntos de entrenamiento y respuestas de API. Son simples, universales y legibles, y esas cualidades explican su éxito. Trabajar con datos para IA, sin embargo, pide cosas que no fueron hechos para dar.

1.1 Lo que les falta a JSON y JSONL

- **Tamaño.** Todo es texto. Un vector de 384 dimensiones se convierte en miles de caracteres por registro: los 100.000 registros de la prueba de la Sección 5 ocupan 519,3 MB en JSONL.
- **Acceso.** No hay índice. Para leer un campo de un registro hay que interpretar el archivo desde el principio hasta él; para leer una columna, el archivo entero.
- **Tipos.** La especificación no fija la precisión de los números: la RFC 8259 deja el límite a cada implementación y aconseja contar solo con la precisión de un double IEEE 754 [1, 28]. No hay tipo para dinero con moneda, medida con unidad, decimal exacto, código que no puede perder el cero inicial, fecha, vector ni tensor. Cada sistema inventa una convención, y pasar de una convención a otra pierde datos.
- **Permisos.** Quién puede ver cada columna y cada fila vive fuera del dato, en una capa aparte, y cada consulta tiene que acordarse de aplicarla.
- **Custodia.** Nada en el archivo prueba que no fue alterado.

En la práctica, el mismo dato pasa por una cadena: JSON en el cable, CSV o Parquet para guardar, una base vectorial para la búsqueda semántica y una capa de permisos para decidir quién ve qué. Cada paso cuesta tiempo y memoria y, a veces, el propio dato: el precio que se vuelve texto, el número fiscal que pierde el cero inicial, el peso que nadie sabe si era en gramos o en kilos.

1.2 Contribuciones

Este artículo describe jsonW (json Wondrous) versión 1.3.11. Sus contribuciones son:

1. un modelo de datos con tres formas que se convierten sin pérdida (fila, columna y texto), en el que todo JSON válido también es jsonW-texto válido (Sección 2);
2. tipos de negocio y de IA declarados en el propio esquema (Sección 2.3);
3. diecinueve recursos dentro del archivo, de la búsqueda y el ranking a las señales de audio, texto y vídeo, mostrados con la herramienta pública cuando esta tiene un comando para ellos (Sección 3);
4. permisos por columna, fila y celda y custodia por hash como parte del formato (Sección 4);
5. medidas reproducibles, con metodología, datos y los límites de la medida (Sección 5), y una comparación con JSON, JSONL, MessagePack, CBOR, Protobuf, Parquet y Arrow (Sección 6).

1.3 Lo que este texto no describe

El núcleo de jsonW está escrito en C, sin dependencias más allá de la biblioteca estándar, y lo mantiene Wortic Tecnologia; no es de código abierto. Lo que es público, y lo que este texto usa, es la documentación de uso, la ABI en C (`sdk/jsonw.h`), los SDK, los binarios de la herramienta y el comportamiento observable [20]. La disposición binaria interna, los códecs de compresión y la organización interna de un bloque quedan fuera de alcance. Lo que afirmamos sobre el formato y la herramienta se puede comprobar con los binarios públicos; las pruebas internas (fuzzing, cargas hostiles, comparación entre SDK) son nuestras y están marcadas como tales. Cuando un recurso existe en el formato pero todavía no tiene comando en la herramienta pública, lo decimos.

2 Modelo de datos

2.1 Tres formas

jsonW es un único modelo de datos con tres maneras de guardar el mismo valor (Figura 1). Pasar de una forma a otra no pierde ningún valor ni cambia el tipo de ninguno.

- **Fila:** binaria, un valor tras otro, para mensajes, eventos y tiempo real. Ocupa el lugar de JSON, MessagePack, CBOR y Protobuf. Es la forma que viaja por el cable (Sección 4.5).
- **Columna:** el archivo `.jsw`, en bloques guardados por columna y leídos directamente del disco mediante `mmap`, sin copia. Ocupa el lugar de JSONL, CSV y Parquet.
- **Texto:** jsonW-texto, para que la gente lea y escriba. Ocupa el lugar de JSON, TOML y YAML. Todo JSON válido ya es jsonW-texto válido; el texto añade notaciones para los tipos que le faltan a JSON, como `12.50 BRL`, `250 g` y `t"2026-11-09T01:00:00.000000Z"`.

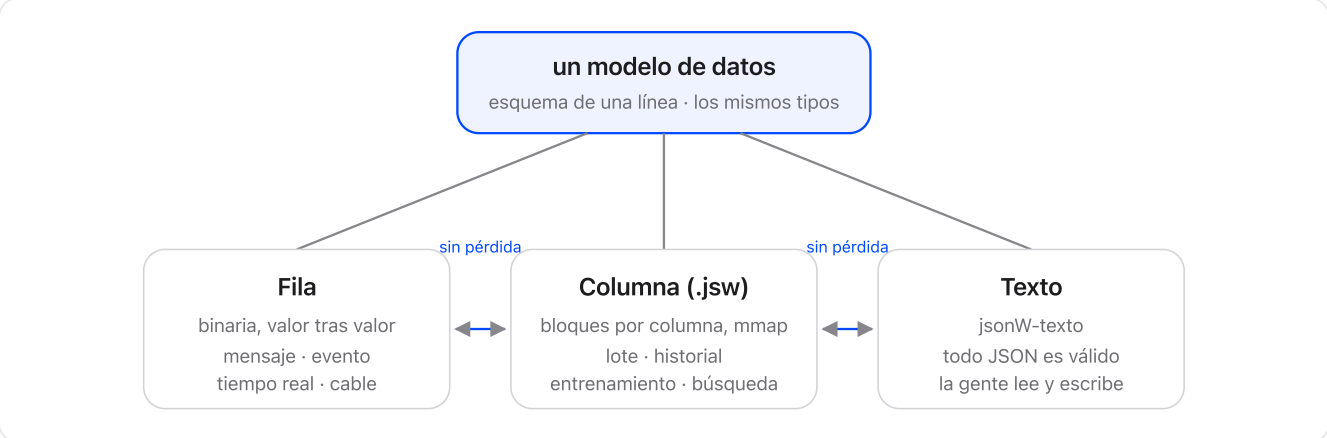


Figura 1. Un modelo, tres formas. El esquema y los tipos son los mismos en las tres; convertir entre ellas no pierde ningún valor ni cambia ningún tipo.

2.2 Esquema

El esquema es una línea de texto con pares `nombre:tipo` separados por comas. El signo de interrogación acepta nulo, y las marcas después del tipo declaran permisos (Sección 4.3). Los nombres de tipos y comandos son palabras en portugués: `texto` es texto, `dinheiro` es dinero, `medida` es medida, `codigo` es código, `tempo` es tiempo, `pessoal` es personal.

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

Listado 1. Un esquema. `cpf` (un número fiscal personal de Brasil) acepta nulo y es un dato personal: sin el alcance `pessoal`, sale nulo.

El esquema evoluciona sin romper a los lectores: un campo nuevo es nulo en las filas antiguas, un nombre nunca cambia de tipo y el lector resuelve los campos por nombre. Así, un archivo antiguo lo lee código nuevo, y un archivo nuevo lo lee código antiguo [20].

2.3 Tipos

La Tabla 1 enumera los tipos de la versión 1.3.11. Los tipos de negocio se comprueban al grabar: una entrada que no encaja en el tipo se rechaza con el motivo, en lugar de guardarse mal.

Tabla 1. Tipos de jsonW 1.3.11 [20].

Tipo	En el esquema	Qué garantiza
decimal exacto	`decimal[2]`	nunca pasa por coma flotante: 12.50 sigue siendo 12.50
dinero	`dinheiro`, `dinheiro[4]`	moneda en cada valor, por código ISO 4217 [27] o por símbolo (`12,50 BRL`, `US$`, `€`, `₿`); solo suma dentro de una misma moneda
medida	`medida`, `medida[kg]`	89 unidades en 14 magnitudes; convierte al grabar y rechaza otra magnitud
código	`codigo`	bytes exactos: números fiscales, claves, matrículas y SKU nunca se normalizan ni se recortan
texto	`texto`, `texto+nfc`	Unicode completo, con acentos, emoji y `\0`; `+nfc` normaliza a pedido [29]
texto repetido	`dic`	diccionario por archivo: cada texto se guarda una vez
tiempo	`tempo`	microsegundos, UTC
url	`url`	validada al grabar (`https://…`, `mailto:`, `tel:`, `urn:`; hasta 8 KiB)
vector	`vf32[n]` `vf16[n]` `bits[n]` `vq8[n]` `vq4[n]`	matriz contigua en el archivo, lista para la búsqueda semántica
tensor	`tensor<bf16>[768,768]`	elementos f32, f16, bf16, u8, i8, i16, i32
medios	`midia`	hash, tipo, tamaño, duración, ancho y alto; jsonW no fija el códec: el medio sigue en su propio códec abierto
y más	`bool` `i64` `f64` `f32` `uuid` `bytes` `ref` `lista<t>` `obj{…}` `mapa<k,v>` `qualquer`	`qualquer` (cualquiera) guarda JSON abierto; anidamiento de hasta 1000 niveles

Las 14 magnitudes de medida son longitud, masa, volumen, energía, tiempo, área, datos, velocidad, temperatura, adimensional, potencia, presión, carbono y densidad. La conversión es exacta entre múltiplos decimales (12,5 mm = 0,0125 m) y se redondea a la escala de la columna en los demás casos (1 mi = 1609,344 m, guardado como 1609 m en una columna `medida[m]`, que no tiene decimales; 25 °C = 77 °F).

2.4 Coma o punto

Los lectores aceptan los dos separadores: `12,50`, `1.234,56 BRL` y `1,234.56 USD`. Cuando aparecen los dos, el último es el decimal. Con uno solo, ese separador es el decimal, salvo una coma seguida de tres dígitos en una moneda de punto (`USD 1,234` es mil) o un punto seguido de tres dígitos en una moneda de coma (`R$ 1.234` es mil). La salida canónica usa siempre el punto. Y los números reales nunca dependen de la configuración regional del proceso: una JVM o un PHP configurados en portugués no cambian el separador.

2.5 Un ejemplo ejecutado

El Listado 2 muestra el camino de JSONL a `.jsw` y de vuelta, con la salida real de la herramienta `wjsw` 1.2.1.

```
# dos líneas de JSONL, como llegan de cualquier 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"}

# lo que no encaja en el tipo se rechaza, con el motivo
$ wjsw de-texto "…,peso:medida[g],…" leite.jsonl leite.jsw     # peso: "2 L"
valor 1: medida de outra grandeza perto de ","sku":"LEI-0002"}
```

Listado 2. `1 kg` se volvió `1000 g` porque la columna es `medida[g]`; el litro es volumen y la columna es masa, así que la fila se rechazó ("medida de otra magnitud"; el mensaje muestra la entrada cerca del error). El dinero conserva su moneda y el SKU se guarda byte a byte. La herramienta escribe sus mensajes en portugués.

3 Los 19 recursos

Los recursos de abajo son parte del formato y del núcleo, no de una capa encima, y la herramienta pública expone la mayoría (Tabla 2). Son las piezas que, en otras pilas, están repartidas entre un motor de búsqueda, una base vectorial, un índice y una capa de permisos. Los ejemplos de esta sección se ejecutaron con el binario público de `wjsw` 1.2.1 para Linux x86_64 (el Listado 6B, con el 1.3.11), y las entradas están en jsonw.org/paper/exemplos; las salidas largas se cortaron donde aparece `…`.

Tabla 2. Los 19 recursos y dónde aparecen en la herramienta y la ABI públicas 1.3.11.

#	Recurso	Qué hace	En la herramienta
1	Búsqueda por términos	vector disperso con pesos por término (IDF), el cálculo que usan los motores de búsqueda	`avaliar … esparso`
2	Búsqueda semántica	coseno sobre vectores densos `vf32`, `vf16`, `vq8` y `vq4`	`avaliar … cosseno`
3	Rejilla multilineal	interpolación bilineal, trilineal y superior, directamente del archivo	`avaliar … grade`
4	Nota por petición	pesos, coseno, disperso o rejilla en una petición de una línea	`avaliar`
5	Los k mejores	top-k determinista con montículo: la nota más alta primero, empates por la fila menor	`avaliar`, `escolher`
6	Zonas	mínimo y máximo por bloque y columna: la búsqueda salta bloques enteros	`filtrar`, `info`
7	Diccionario	texto repetido guardado una vez por archivo	tipo `dic`
8	Índice por clave	va directo a la fila de una clave, sin recorrer	en el núcleo; sin comando propio
9	Léxico	autómata Aho-Corasick leído del disco; palabra completa, sin distinguir acentos ni mayúsculas	`reconhecer`
10	Grabación en vivo	un proceso lee mientras otro graba, sin ver nunca medio bloque	en el núcleo; sin comando propio
11	Vivo, borrado e historial	grabar la misma clave edita; `_apagado: true` borra; la compactación reescribe solo lo vigente	en el núcleo; sin comando propio
12	Vista por permiso	columna, fila y celda, según la credencial	`ver`, `wjsw-cgi`, SDK
13	Custodia	crc32c y SHA-256 (o BLAKE3) por bloque, encadenados	`conferir`, `reparar`
14	Lectura remota parcial	con la cabecera y el pie, descarga solo los bloques de las filas leídas	`wremoto`
15	Grafía y sonido parecidos	nota de 0 a 1 por trigramas de la grafía y de una clave fonética, en portugués de Brasil, inglés estadounidense, español y chino	`filtrar … ~~`
16	Señales del audio	tono, pausas, intensidad, jitter, shimmer, HNR, formantes, MFCC, centroide, tasa de articulación y respiraciones	ABI: `jsonw_sinal_audio`
17	Señales del texto	intensificadores, atenuadores, polaridad, absolutos, negación, voz pasiva, pronombres, forma e indicios de ironía, en pt, en, es y zh	ABI: `jsonw_sinal_texto`
18	Vídeo en escenas	cortes de escena, las medidas y las señales del audio de cada tramo y un mosaico de los fotogramas	ABI: `jsonw_video_*`
19	Columna por bloque y por tenant	diccionario local por bloque, dígitos de ancho fijo, zona del tenant y comparación por lotes	al grabar; `filtrar`, `calcular`

3.1 Búsqueda y ranking (recursos 1 a 5)

El comando `avaliar` (evaluar) da una nota a cada fila y devuelve las k mejores. La petición puede ser una suma ponderada de columnas numéricas, el coseno entre una columna de vectores y una consulta, el producto entre vectores dispersos o una consulta a una rejilla. La misma petición existe en JSON, idéntica en la herramienta y en la ABI, y todos los SDK devuelven el mismo resultado, byte a byte [20]. El cálculo lee solo las columnas de la petición, en paralelo. El Listado 3 usa cinco productos con una columna de vectores `vf16[4]` y una columna de términos `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 y 5. nota por pesos (el precio cuenta en contra, la valoración a favor) y las 3 mejores
$ 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. búsqueda semántica: coseno con la 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. búsqueda por términos: producto entre vectores dispersos (término 12 y término 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]]}

# la misma petición en JSON, como en la 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. los 2 valores más altos de una columna
$ wjsw escolher prod.jsw nota 2
4.9     3       {id:4,produto:"Bolo de milho",…}
4.8     0       {id:1,produto:"Café especial",…}
```

Listado 3. Salida: nota, posición de la fila y la fila en jsonW-texto, de la mejor a la peor.

Dos detalles de la salida merecen atención. Primero, el vector guardado como `vf16` vuelve como `0.7998046875` donde la entrada decía `0.8`: la media precisión guarda unos tres dígitos decimales. Cuando la exactitud del vector importa, la columna debe ser `vf32`; `vq8` y `vq4` cambian más precisión por tamaño. Segundo, la búsqueda por términos es un producto escalar: los pesos IDF [22] se guardan en el vector de cada fila, y la consulta trae los suyos.

Rejilla (recurso 3). Una rejilla es una tabla de valores sobre ejes (por ejemplo, el coste de envío por distancia y peso). La petición `{"grade":{"campos":["x","y"],"arquivo":"g.jsw"}}` usa las columnas de la fila como coordenadas y devuelve, como nota, el valor interpolado entre los puntos vecinos de la rejilla: bilineal con dos ejes, trilineal con tres, y así sucesivamente. En la versión 1.3.11, el formato de la tabla de la rejilla todavía no está en la documentación pública, así que no mostramos su salida aquí.

3.2 Organización del archivo (recursos 6 a 8)

Zonas. Cada bloque guarda el mínimo y el máximo de cada columna. Un filtro que un bloque no puede cumplir hace que el lector salte el bloque entero. En el archivo de 100.000 registros de la Sección 5, con 103 bloques:

```
$ 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
```

Listado 4. En el archivo de 100.000 registros (ids de 0 a 99.999), las zonas saltan 102 de 103 bloques ("pulados pela zona"); en el archivo de productos del Listado 3, `info` muestra el diccionario de la columna `categoria` y las zonas, con los reales en doble precisión tal como los imprime la herramienta.

Diccionario. Una columna `dic` guarda cada texto distinto una vez por archivo. En los 100.000 registros de la prueba, las columnas `tipo` y `fonte` suman 10 textos distintos, y `wjsw info` muestra `dicionario 10 textos`.

Índice por clave. El formato tiene un índice que va directo a la fila de una clave, sin recorrer el archivo. En la herramienta pública 1.3.11 no tiene comando ni función en la ABI; el uso visible de las claves es `comparar`, que empareja las filas de dos archivos por una columna y cuenta las diferencias:

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

Solo en a: 1 · solo en b: 1 · cambió: 1 · iguales: 0.

3.3 Léxico (recurso 9)

Una tabla de términos, con el esquema `termo:texto,forma:dic,sinal:dic,peso:f32,idioma:dic,origem:dic`, se compila, al abrirla, en un autómata Aho-Corasick [21] que encuentra todos los términos de un texto en una sola pasada. La coincidencia es por palabra completa, sin distinguir acentos ni mayúsculas, y devuelve el tramo original y la suma de los pesos por señal: un clasificador de texto sin entrenamiento, útil para la clasificación inicial y para llevar una petición al especialista correcto.

```
$ 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)
```

Listado 5. Una frase en portugués ("Quiero CANCELAR el pedido, o dime cuánto cuesta el presupuesto nuevo"). "CANCELAR" coincidió con "cancelar", y "orcamento" con "orçamento", porque la coincidencia ignora acentos y mayúsculas. Las señales son "intención de compra" y "riesgo de pérdida".

3.4 Escritura e historial (recursos 10 y 11)

Un archivo `.jsw` solo admite anexar. Eso da dos garantías. La primera es la **grabación en vivo**: un proceso puede leer el archivo mientras otro lo graba, como un `tail -f` que nunca ve medio bloque. La segunda es el **historial**: grabar de nuevo la misma clave es una edición, grabar la clave con `_apagado: true` es un borrado, y la compactación reescribe solo lo vigente en un archivo nuevo, dejando el historial en el original.

En la versión 1.3.11, la lectura durante la grabación, la vista "solo lo vigente" por clave y la compactación no tienen comando en `wjsw` ni función en la ABI pública. Con la herramienta pública, el historial aparece tal como es, fila tras fila:

```
$ 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}
```

Lo que la ABI pública sí garantiza al grabar es la **publicación atómica**: el archivo aparece, entero, solo en `fechar()` (cerrar). Un escritor que sale por un error a mitad de camino llama a `abandonar()`, y el destino queda como estaba.

3.5 Permisos, custodia y acceso remoto (recursos 12 a 14)

La vista por permiso (12) y la custodia (13) son el tema de la Sección 4. La **lectura remota parcial** (14) usa peticiones HTTP `Range` [24]: el lector descarga la cabecera y el pie, donde están el esquema, el índice de bloques y las zonas, y después solo los bloques que necesita (Figura 4). Medimos con `wremoto` 1.2.1 contra un nginx local que servía el archivo de 100.000 registros (el `com.jsw` de la Sección 5, con el nombre `cartoes.jsw`); con `file://` en lugar de la dirección HTTP, `wremoto` da los mismos números, sin 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)
```

Listado 6. Para leer una fila del medio de un archivo de 79,1 MB, el lector descargó el 0,98% ("baixado no total"); para el filtro, el 0,84%, porque las zonas descartaron los otros bloques antes de la descarga.

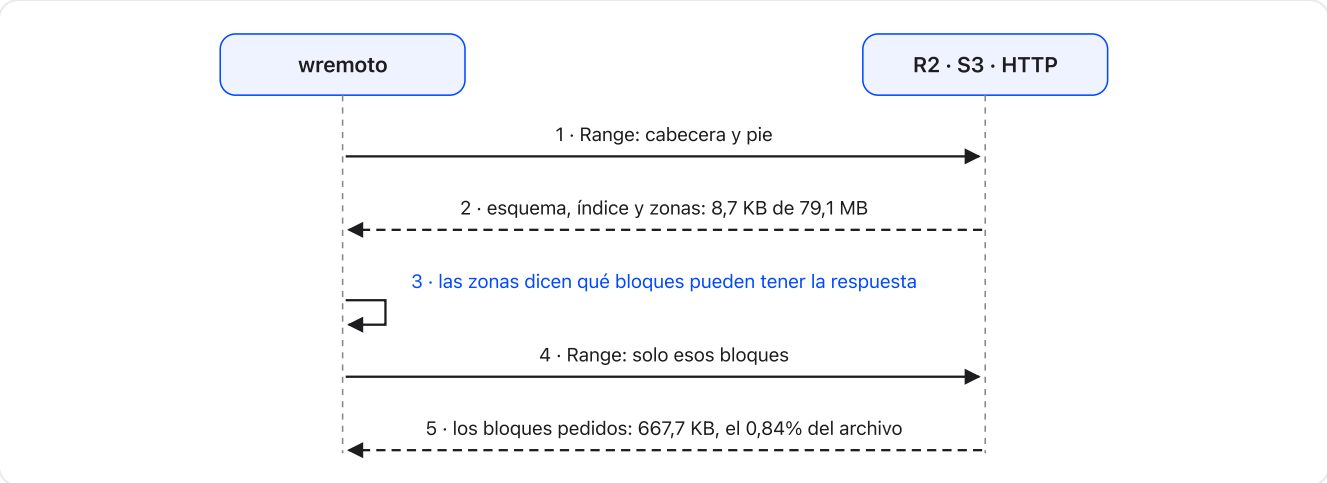


Figura 4. Lectura remota parcial del archivo de 100.000 registros, con los números medidos en el Listado 6. Con R2 o S3, cada petición se firma con AWS SigV4 [25].

3.6 Grafía, señales y columna por bloque (recursos 15 a 19)

Grafía y sonido (recurso 15). El operador `~~` del filtro da a cada valor una nota de 0 a 1: la mayor de dos medidas de Dice sobre trigramas de caracteres Unicode, una en la grafía normalizada (sin acentos ni mayúsculas) y otra en una clave fonética de la lengua pedida. Hay clave para el portugués de Brasil, el inglés estadounidense, el español y el chino escrito en pinyin; en los ideogramas, los trigramas cuentan carácter por carácter, y su clave de sonido, que depende de una tabla de lecturas, todavía no está en esta versión. El filtro no usa índice: recorre solo la columna pedida, en paralelo (Listado 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"}
```

Listado 6B. Grafía y sonido parecidos. "Kokakola" encuentra "Coca-Cola": la clave del portugués lleva la "c" dura y la "k" al mismo sonido. `~~en`, `~~es` y `~~zh` cambian la lengua; `~~pt@0.4` cambia el umbral.

Señales de la comunicación (recursos 16 a 18). Miden cómo habló, escribió o apareció la persona, y no deciden nada: quien interpreta es el modelo que lee la fila. En el audio (PCM de 16 bits), el tono sale de la autocorrelación normalizada, el mismo método de Praat; el jitter y el shimmer, de los pulsos glotales; los formantes, de la predicción lineal (LPC); y las pausas, del umbral de Otsu sobre la intensidad, que saca el corte de los propios datos en lugar de un número fijo. En el texto, el léxico de la lengua (pt, en, es o zh, escrito a mano) se casa con el autómata del recurso 9, y las cuentas de forma (mayúsculas, puntuación, emoji, risas) no usan diccionario. El chino, que no separa las palabras con espacios, se casa ideograma por ideograma. En el vídeo, quien llama decodifica y entrega los fotogramas y el sonido; cada fotograma se vuelve una miniatura con luz, contraste, color y forma, y los cortes de escena salen del mismo umbral de Otsu sobre la diferencia entre fotogramas, solo cuando la serie tiene dos grupos claros. Cada medida sale como una fila de jsonW-texto en el esquema declarado, lista para grabarse en un `.jsw`.

Columna por bloque y por tenant (recurso 19). Tres códecs nuevos: el diccionario local de cada bloque, con y sin codificación de entropía, para el texto que se repite dentro del bloque pero no en todo el archivo; y los dígitos de ancho fijo, para códigos hechos solo de cifras, como el número de un proceso. La zona de cada bloque guarda también el tenant y, con la vista de una sesión, el bloque de otro cliente ni siquiera se abre. El filtro y el cálculo comparan el código del diccionario por lotes, con instrucciones vectoriales (SSE2), y evalúan el «contiene» en el bloque entero de una vez. Un archivo grabado con estos códecs no se abre en los lectores 1.2.x. En la misma versión, el nivel máximo (`--max`) ganó dos códecs que graban cada celda y cada vector como la diferencia con la fila anterior: el prefijo y el sufijo comunes y solo el centro que cambia. El efecto en la base del INPI está en las Tablas 5C y 5D.

4 Seguridad y custodia

4.1 Custodia

Cada bloque del archivo se sostiene solo y lleva dos comprobaciones: un CRC32C [11], rápido, para daños accidentales, y un hash criptográfico, SHA-256 [12] por defecto o BLAKE3 [13] como opción. Los hashes forman una cadena que cubre el archivo entero: cualquier byte cambiado en los datos, en una zona o en el diccionario hace fallar `conferir` (verificar) (Figura 2).



Figura 2. El archivo `.jsw` visto desde fuera, al nivel descrito en la documentación pública. La organización interna de la cabecera, los bloques y el pie no se publica (Sección 1.3).

```
$ wjsw conferir vendas.jsw
ok: 2 linhas, 1 blocos, sha256 e crc32c conferidos
# un byte cambiado en el medio del archivo
$ wjsw conferir adulterado.jsw
FALHOU: bloco 0: nao veio ou nao confere
# los últimos 20 bytes cortados, como en una grabación interrumpida
$ 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 en lugar de 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
```

Listado 7. Custodia, adulteración y reparación, con la salida real ("FALHOU" = falló; "sem rodape: rode wjsw reparar" = sin pie: ejecute wjsw reparar).

> La custodia prueba la integridad, no la autoría. Un hash sin clave puede recalcularlo quien reescriba el archivo entero. Para probar que un archivo es el mismo que salió de una fuente, guarde aparte el identificador del archivo que muestra `wjsw info`, o fírmelo con una herramienta aparte, y compárelo.

4.2 Lectura de archivos hostiles

El lector trata todo archivo como hostil: comprueba cada índice y cada límite antes de usarlo, rechaza anidamientos de más de 1000 niveles y se prueba con fuzzing (entradas generadas al azar) en la batería de pruebas. Comillas, comas, barras y cualquier byte dentro de un texto son datos y nunca rompen la lectura.

4.3 Vista por permiso

En jsonW, los permisos se declaran en el esquema y los aplica quien sirve el archivo, antes de responder: lo que la sesión no puede ver nunca sale del servidor (Figura 3, Tabla 3).

Tabla 3. Las capas de la vista por permiso.

Capa	Cómo se declara	Qué ocurre
Columna	`+escopo(financeiro)`, `+pessoal`	sin el alcance, la columna sale nula y no se puede usar ni siquiera en un filtro
Fila	una columna entera llamada `tenant` o `tenant_id`, o marcada `+tenant`	solo las filas del tenant de la credencial; los bloques de otros tenants ni se abren; `tenant:todos` abre todo; el tenant es un entero: con una columna `tenant` de texto, no sale ninguna fila
Celda	las dos juntas	las filas del propio cliente, sin la columna de datos personales

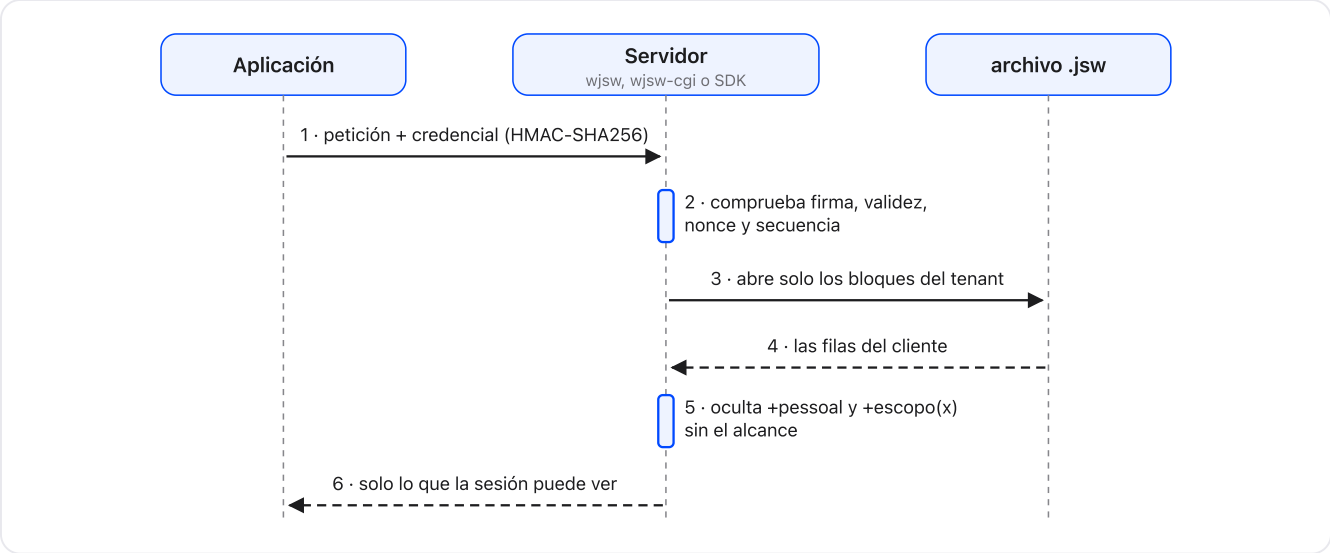


Figura 3. La vista por permiso. Los pasos 2 a 5 ocurren dentro del servidor; el archivo nunca sale entero hacia quien lo pide.

Una clínica con dos unidades muestra las tres capas. La recepción de la unidad 1 ve la agenda de la unidad 1 sin número fiscal y sin importe; el equipo financiero de la unidad 1 ve el importe, todavía sin número fiscal; nadie de la unidad 1 ve la unidad 2:

```
$ wjsw de-texto "paciente:texto,cpf:codigo?+pessoal,valor:dinheiro+escopo(financeiro),data:tempo,tenant:i64"
clin.jsonl clin.jsw
# recepción de la unidad 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}
# financiero de la unidad 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}
```

Listado 8. La misma consulta, dos credenciales, dos respuestas. Los datos del ejemplo son ficticios.

> La vista no es cifrado. Protege lo que sale del servidor; quien tiene el archivo `.jsw` tiene todos los datos, y `para-texto` lo exporta todo, porque es la copia para quien ya es dueño del archivo. Para mostrar datos a un usuario, use `ver`, `wjsw-cgi` o el SDK con `visao()`; para proteger el archivo en reposo, use cifrado de disco o de almacenamiento.

4.4 Credencial

La credencial que acompaña a una petición es ella misma un valor jsonW, con emisor, sujeto, tenant, alcances y validez, firmada con HMAC-SHA256 [18]. La firma se comprueba en tiempo constante, y un nonce con un número de secuencia impide que una petición se repita. Viaja encima de TLS, no en su lugar.

4.5 Transporte poscuántico

La forma de fila viaja en un protocolo sobre QUIC [15] con TLS 1.3 [16] y el intercambio de claves híbrido X25519MLKEM768 [17], que combina la curva X25519 con ML-KEM-768, estandarizado por el NIST en el FIPS 203 [14]. El argumento poscuántico tiene dos partes. El intercambio de claves usa un mecanismo diseñado para resistir a un ordenador cuántico. Y la custodia y las credenciales usan solo hashes y HMAC de 256 bits, que no dependen de la factorización ni del logaritmo discreto; para encontrar una preimagen o la clave del HMAC, el

algoritmo de Grover [19] da una aceleración cuadrática y deja unos 128 bits de seguridad; para encontrar colisiones, la cota cuántica conocida es menor, unas 2^{85} operaciones sobre SHA-256 con el algoritmo de Brassard, Høyer y Tapp [35], que necesita una memoria cuántica enorme.

La última línea de `wjsw medir` (Sección 5) mide la forma de fila: un registro de prueba ocupa 5.157 bytes en JSON y 1.073 bytes como mensaje, 4,8 veces menos, en todas las ejecuciones.

> El paquete público 1.3.11 no incluye un servidor ni un cliente del protocolo.

4.6 Servidores web

El paquete incluye `wjsw-cgi`, que funciona en Apache, nginx, LiteSpeed, Caddy e IIS [20]. Toda petición de `.jsw` pasa por él: un archivo sin columnas marcadas y sin tenant sale sin filtrar, con `Range` y CORS; los demás salen filtrados, con la vista aplicada. El alcance viene del servidor después de autenticar, nunca de una cabecera enviada por el cliente, y pedir una columna oculta mediante `campos=` u `onde=` devuelve 403, así que nada se filtra por deducción. En la versión 1.2.0, la configuración de ejemplo dejaba que el servidor entregara el archivo crudo cuando la petición no tenía parámetros; la 1.2.1 lo corrigió, y quien haya instalado la configuración antigua debe sustituirla [20].

5 Medidas

Esta sección mide jsonW 1.3.11 frente a JSONL y dice, al lado, lo que la medida no prueba. Los números se rehicieron para este texto el 13 de noviembre de 2026, con el binario de la versión publicada, en la misma máquina de la medida del 12 de noviembre.

5.1 Metodología

- **Herramienta:** `wjsw medir` (medir), incluida en el binario público, con 100.000 filas: `wjsw medir <carpeta> 100000`. Binario `bin/linux-x86_64/wjsw` del repositorio público, versión 1.3.11 (prelanzamiento, SHA-256 `cb119d7473e7d786…`).
- **Máquina:** un servidor virtual con un AMD EPYC 9355P, 8 vCPU y 18 GB de memoria, Ubuntu 26.10, núcleo Linux 7.3. Los archivos estaban en un `tmpfs` (memoria), así que la medida excluye el disco. La máquina no estaba aislada de otras cargas.
- **Datos:** 100.000 registros sintéticos, con forma de ficha de una base de conocimiento: `id:i64,tipo:dic,titulo:texto,corpo:texto,fonte:dic,valor:decimal[4],vetor:vf16[384],tags:lista<texto>`. Los textos repiten una plantilla con el número de la fila; `tipo` y `fonte` alternan entre 5 valores; el vector tiene 384 componentes múltiplos de 1/1024, que `vf16` guarda sin pérdida.
- **El lado JSON:** el mismo conjunto escrito como JSONL y leído por el propio lector de texto de jsonW, que interpreta cada línea y convierte los tipos del esquema. Los dos archivos se cargan en memoria antes de que empiece el reloj.
- **Repetición:** 10 ejecuciones seguidas. Damos la mediana de cada tiempo y, para las razones, la mediana y el intervalo sobre las 10. Las salidas brutas se publican con este texto.
- **Comprobación:** en cada ejecución la herramienta comprueba que las dos lecturas contaron 100.000 filas con la misma suma.

5.2 Tiempo

Tabla 4. Tiempo para grabar y leer los 100.000 registros. Tiempos: mediana de 10 ejecuciones. Razón: JSONL ÷ jsonW en cada ejecución, con la mediana y el intervalo sobre las 10. En la fila de 8 hilos, la razón es contra la lectura de JSONL en un hilo. El JSONL lo graba y lo lee el propio código de texto de jsonW.

Operación	JSONL	jsonW	Razón (intervalo)
Grabar	15.211 ms	131 ms	114× (88 a 127)
Leer todo, todos los campos	3.687 ms	78 ms	46× (43 a 51)
Leer todo, 8 hilos	(sin versión paralela)	9,0 ms	419× (365 a 532)
Leer una columna (`valor`)	3.687 ms	1,59 ms	2.310× (1.592 a 2.784)
Ir a la fila N, campo `titulo`	leído desde el principio	0,09 µs (0,05 a 0,16)	sin razón definida

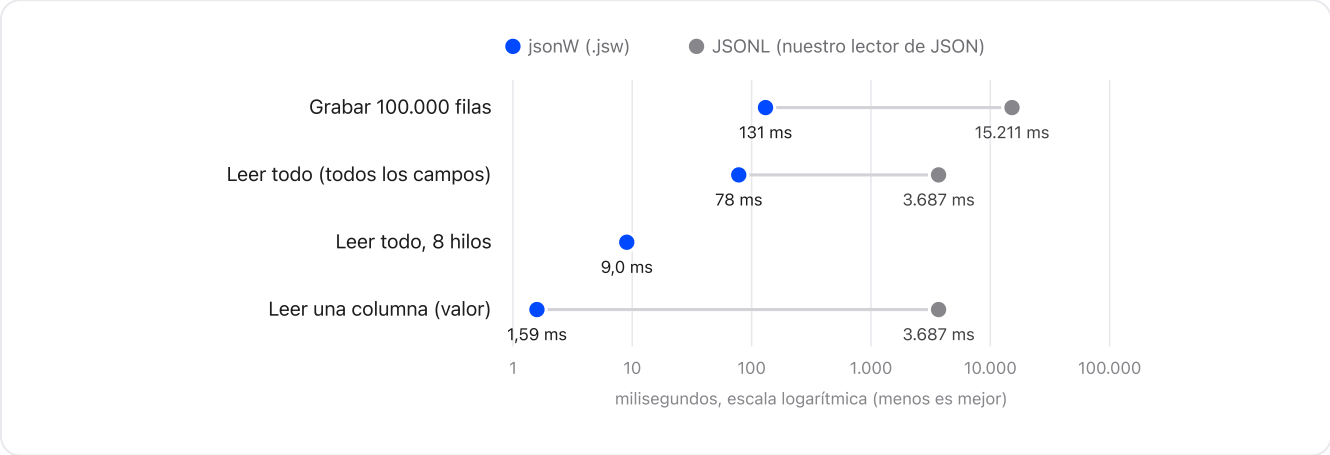


Figura 6. Los tiempos de la Tabla 4 en escala logarítmica (mediana de 10 ejecuciones). La distancia horizontal entre los puntos de cada fila es la razón.

La Figura 6 muestra los mismos tiempos en escala logarítmica. La mayor diferencia está en leer una columna: JSONL obliga a interpretar cada línea entera para sacar un campo, y el archivo en columnas lee solo esa columna. Es una ventaja estructural, la misma que cualquier formato en columnas tiene sobre un formato en filas. Para "ir a la fila N", JSONL no puede saltar: hay que leerlo desde el principio hasta la fila, así que la razón depende de N y no la damos.

5.3 Tamaño

Para comparar el tamaño con los compresores de uso general, regrabamos los mismos 100.000 registros: un generador en `awk`, publicado con este texto, produce el JSONL; `wjsw de-texto` graba el `.jsw`; y el JSONL que devuelve `wjsw para-texto --json` tiene 519.264.563 bytes, el mismo tamaño, en la escala de MB, que `wjsw medir` da para su propio JSONL (519,3 MB); el `.jsw` tiene 79.076.300 bytes (79,1 MB en los dos). El JSONL del generador tiene 200.000 bytes más, porque allí el valor decimal va entre comillas; la Tabla 5 usa el JSONL regrabado por jsonW, y con el archivo del generador las razones cambian en el primer decimal (gzip: 19,7× en lugar de 19,8×). El JSONL se comprimió con gzip 1.14 [32], zstd 1.5.7 [31] y xz 5.8.3 [33] en sus niveles máximos habituales.

Tabla 5. Tamaño de los 100.000 registros, en MB (10^6 bytes), con y sin la columna del vector. Razón = JSONL sin comprimir ÷ el tamaño de esa fila.

Archivo	Con vector	Razón	Sin vector	Razón
JSONL	519,3	1,0×	38,2	1,0×
`.jsw` , vector `vf32`	155,9	3,3×	n/a	n/a
`.jsw` , vector `vf16` , nivel por defecto	79,1	6,6×	2,22	17,2×
`.jsw` , vector `vq8` (con pérdida)	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` , vector `vf16` , `--max`	1,36	381,1×	0,0175	2.185×

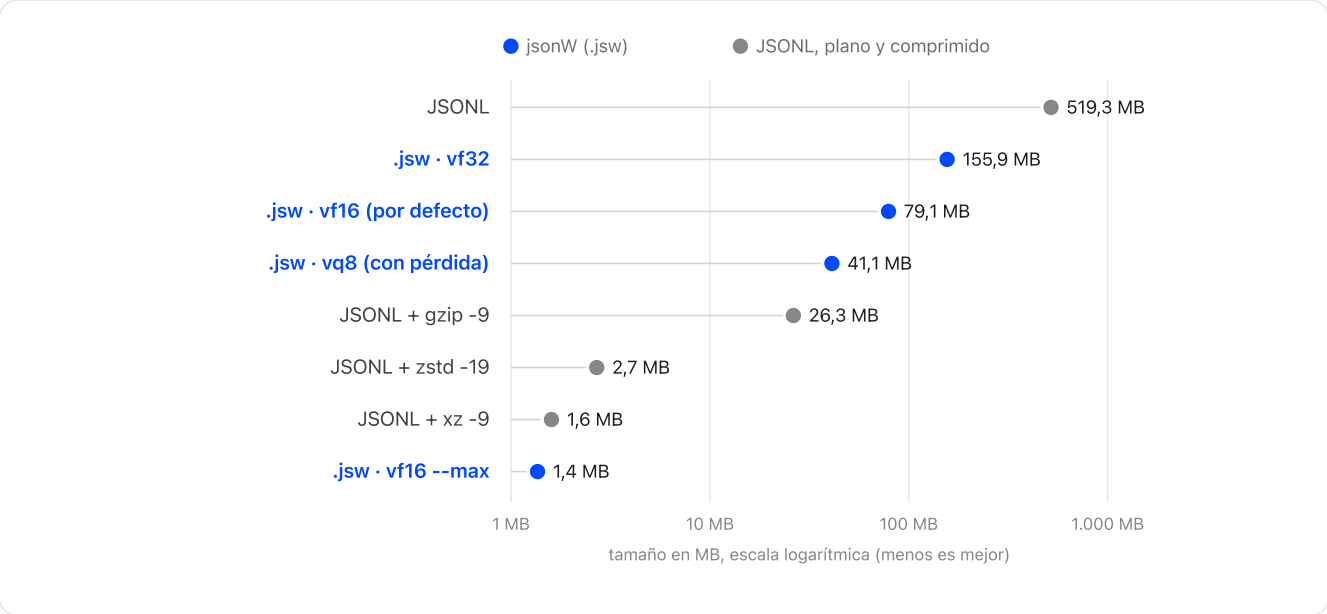


Figura 5. Los tamaños con vector de la Tabla 5, en escala logarítmica.

La Figura 5 muestra los tamaños con vector. Frente al JSONL sin comprimir, que es como el JSONL suele guardarse e intercambiarse, el `.jsw` estándar es 6,6 veces más pequeño, y casi todo su tamaño es el vector: 384 componentes de 2 bytes en 100.000 filas son 76,8 MB. Frente al JSONL comprimido con zstd o xz, el `.jsw` estándar es mayor. En el nivel máximo, la 1.3.11 quedó por debajo de ambos: 1,36 MB con vector, frente a 2,67 MB de zstd y 1,56 MB de xz, y 17,5 kB sin vector, frente a 0,63 y 0,47 MB; en la 1.2.1, el mismo nivel daba 3,75 y 2,19 MB. La ganancia viene de dos códecs nuevos del nivel máximo, que graban cada celda y cada vector como la diferencia con la fila anterior (el prefijo y el sufijo comunes y el centro que cambia), y es grande aquí porque los datos de la prueba son muy repetitivos: los textos se repiten con un cambio de número y el vector sigue un patrón aritmético. La vuelta es sin pérdida: el `para-texto` del archivo en el nivel máximo es igual, byte a byte, al del archivo en el nivel estándar (mismo SHA-256, con y sin vector). En datos reales la ganancia es mucho menor: en la base del INPI, el nivel máximo de la 1.3.11 quedó un 16,5% por debajo del de la 1.2.1 (Tabla 5C). Los vectores de modelos reales también suelen comprimirse poco; no lo medimos aquí, pero, si es así, el tamaño del `.jsw` con vector queda cerca del tamaño bruto del vector.

Lo que un archivo comprimido de uso general no da es acceso. Para leer un campo de un JSONL comprimido con zstd hay que descomprimirlo todo (de 0,13 s a 0,24 s para descomprimir `com-volta.jsonl.zst`, en 3 ejecuciones en la misma máquina, fuera del entorno de la medida) y después interpretar el texto, mientras que el

`.jsw` lee el campo directamente, salta bloques por las zonas, verifica la custodia y se puede leer a distancia por partes.

5.4 Datos reales: CSV, XML, XLSX y Parquet

Las medidas anteriores usan datos generados. Para comparar con los formatos de hoja de cálculo y de intercambio más comunes, usamos datos públicos: el primer millón de filas del archivo `MARCAS_DADOS_BIBLIOGRAFICOS.csv`, de los datos abiertos de marcas del INPI, la oficina brasileña de patentes y marcas (13 columnas: número de la solicitud, fechas, presentación, naturaleza, nombre de la marca, situación). Un programa en C, sin bibliotecas, leyó el CSV siguiendo la RFC 4180 (comillas, comas y saltos de línea dentro del campo) y grabó las mismas filas en cuatro formas: el CSV regrabado; un XML con un elemento por fila y un hijo por columna; un XLSX montado por el propio programa (Office Open XML, celdas de texto en línea, comprimido con `zip -6`); y el `.jsw`, con el esquema escrito a mano (fechas como texto; presentación, naturaleza y situación como diccionario). Un campo vacío queda ausente en XML y en jsonW. XLSX acepta como máximo 1.048.576 filas por hoja, por eso un millón.

La consulta fue la misma en los cuatro: contar los registros en vigor (`codigo_situacao = IPAS023`). Las cuatro formas dieron el mismo resultado, 352.248 registros, y `wjsw conferir` verificó el SHA-256 y el CRC32C de todos los bloques del `.jsw`. Cada consulta se ejecutó 5 veces con el archivo ya en caché; vale la menor. Máquina: servidor de 8 núcleos con Linux 7.3, jsonW 1.2.0, 13 de noviembre de 2026.

Tabla 5B. Un millón de filas reales del INPI en cuatro formatos. Tamaño en MB (10^6 bytes); consulta = contar los registros en vigor, en milisegundos.

Formato	Tamaño	Con zstd -3	Consulta	Cómo se hizo la consulta
XML	502,1	46,0	159	`grep -c` en el archivo entero
CSV	141,1	36,1	97	`grep -c` en el archivo entero
XLSX	74,3	n/a	9.364	`unzip -p` de la hoja + `grep`
`.jsw`	51,2	n/a	119	`wjsw calcular --onde`, solo la columna

En tamaño, el `.jsw` fue 2,8 veces más pequeño que el CSV, 1,5 veces más pequeño que el XLSX y 9,8 veces más pequeño que el XML. El CSV y el XML comprimidos con zstd son más pequeños que el `.jsw` (36,1 y 46,0 MB contra 51,2 MB), pero, como en la Sección 5.3, hay que descomprimirlos enteros antes de cualquier lectura. En la consulta, el `.jsw` fue 79 veces más rápido que el XLSX, que hay que descomprimir en cada lectura. Frente al CSV y el XML, `grep` fue más rápido o parecido: solo busca un trozo de texto en el archivo entero, sin separar columnas ni tipos, y daría un resultado erróneo si el mismo código apareciera en otra columna. En este conjunto no aparece, y las cuentas coincidieron. Una consulta con la misma garantía en el CSV exige leer y separar cada fila, lo que `awk` hizo en 1.597 ms en la primera ronda de esta medida, sin tratar las comillas.

Lo que ninguno de los tres formatos trae por sí mismo: esquema con tipos (fecha, decimal, dinero, vector), lectura de una columna sin leer las demás, búsqueda por términos y por significado, vista por permiso, historial de filas borradas y custodia por bloque. El XLSX tiene tipos de celda y varias hojas, pero limita el tamaño de la hoja y hay que descomprimirlo para leerlo.

La base entera. La Tabla 5B usa un millón de filas de un archivo. Para ver el formato con todo, convertimos los siete archivos de los datos abiertos de marcas del INPI [38], descargados el 12 de noviembre de 2026, sin cortar nada: 55.278.212 filas y 12,53 GB de CSV, con todas las columnas y el historial completo de resoluciones de cada solicitud (24.978.976 filas solo en `MARCAS_DESPACHOS.csv`). Cada archivo recibió un esquema escrito a mano. Números, fechas y códigos conservaron su propio tipo; el texto, los bytes tal como llegaron. El conversor, en C, comprueba cada valor tipado en el momento: el número y la fecha tienen que volver al mismo texto del CSV, o la

conversión se detiene. Esto detectó dos casos reales en la fuente, que se guardaron tal como venían: una fecha de prioridad en el año 1018 (un error de tecleo en el original) y una fila de la clasificación de Niza sin número de solicitud, entre las 5.333.893 de ese archivo.

El Parquet [3] lo grabó DuckDB 2.0.0 [39] con los mismos tipos y compresión zstd. El XML sigue el diseño de la Tabla 5B. El XLSX quedó fuera: seis de los siete archivos pasan del límite de 1.048.576 filas por hoja, y el de resoluciones tiene casi 24 veces ese límite. Las ocho formas dieron las mismas cuentas en las tres consultas, con una excepción, explicada abajo. `wjsw conferir` verificó el SHA-256 y el CRC32C de todos los bloques de los siete `.jsw`. Cada consulta se ejecutó 5 veces, con el archivo ya en caché, y vale la menor. La grabación del jsonW usó un núcleo. DuckDB, zstd y xz usaron los ocho núcleos, y `wjsw calcular` también. Máquina: la misma de la Tabla 5B, 31 GB de memoria, 13 de noviembre de 2026.

Tabla 5C. La base entera de marcas del INPI (siete archivos, 55,3 millones de filas) en cada formato. Tamaño en MB (10^6 bytes), sumado de los siete archivos; grabar = tiempo para producir el formato a partir del CSV, en segundos.

Formato	Tamaño	Veces más pequeño que el CSV	Grabar	¿Lee una columna sin abrir el resto?
XML	24.804,7	0,5	n/a	no
CSV (original)	12.527,4	1,0	n/a	no
`.jsw` , 1.2.1	2.804,8	4,5	176,7	sí
`.jsw` , 1.3.11	2.579,2	4,9	n/a	sí
`.jsw` , 1.2.1, `--max`	2.321,8	5,4	184,9	sí
XML + zstd -3	2.210,0	5,7	21,4	no, descomprime todo
`.jsw` , 1.3.11, `--max`	1.938,1	6,5	n/a	sí
CSV + zstd -3	1.908,9	6,6	13,2	no, descomprime todo
Parquet (zstd)	1.665,8	7,5	16,6	sí
CSV + xz -6	1.161,0	10,8	544,5	no, descomprime todo

Tabla 5D. Tres consultas por columna en la base entera, en milisegundos (la menor de 5), medidas por quinta vez el 13/11/2026 a las 06:21 UTC, con el `wjsw` 1.3.11 publicado y el 1.2.1, a la misma hora. En vigor: contar `codigo_situacao = IPAS023` en 6,7 millones de procesos. Historial: todos los despachos de un proceso (`numero_inpi = 818602856`) en 25,0 millones. Clase 35: contar `classe_nice = 35` en 5,3 millones. Entre paréntesis, DuckDB con un núcleo. El XML viene de la primera ronda, porque el archivo se borró después de ella (ocupaba 24,8 GB).

Formato y lector	En vigor	Historial	Clase 35
Respuesta correcta	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 leyendo el CSV	375 (1.447)	1.205 (4.523)	592 (1.979)
CSV, `grep -c`	519	6.690	1.278 (errónea: 1.076.929)
CSV + zstd, `zstd -dc | grep -c`	798	3.129	2.786 (errónea)
CSV + xz, `xz -dc | grep -c`	1.050	3.025	2.105 (errónea)
CSV + xz, `xz -dc | grep -c`	971	2.628	1.950 (errónea)
XML, `grep -c` (primera ronda)	3.017	9.925	3.430

Lo que muestra la base entera. En tamaño, el `.jsw` 1.3.11 quedó 4,9 veces más pequeño que el CSV en el nivel por defecto y 6,5 veces en el máximo, y 12,8 veces más pequeño que el XML; el 1.2.1 quedaba 4,5 y 5,4 veces más pequeño que el CSV, y el 1.3.11 en el máximo quedó un 16,5 % más pequeño que el 1.2.1. En la consulta, con ocho núcleos de los dos lados, el 1.3.11 contó los registros vigentes en 43 ms en el nivel máximo y 44 ms en el por defecto, por delante de los 54 ms de Parquet leído por DuckDB; con un núcleo, DuckDB sobre Parquet tardó 76 ms en ese conteo y 521 ms en el historial. En las tres consultas, el `.jsw` fue más rápido que DuckDB leyendo el propio CSV, que `grep`, que el CSV comprimido y que el XML. En el historial de un expediente y en la clase 35, el `.jsw` tardó 207 y 142 ms, y Parquet con ocho núcleos, 129 y 41 ms: es el próximo paso de rendimiento de la Sección 8. En tamaño, Parquet (1.665,8 MB) y el CSV comprimido con xz (1.161,0 MB) ocupan menos que el `.jsw` en el máximo (1.938,1 MB), y el CSV comprimido debe descomprimirse entero antes de cualquier lectura; con zstd, el CSV quedó prácticamente igual al `.jsw` en el máximo (1.908,9 MB). La grabación del `.jsw` 1.2.1, en un núcleo, tardó 177 s; la de Parquet, con ocho, 17 s. Los archivos 1.3.11 se regrabaron a partir del `.jsw` 1.2.1 (`wjsw regravar`), por eso la Tabla 5C no trae su tiempo de grabación.

Las rondas y las correcciones. La Tabla 5D se midió cinco veces, siempre con el Parquet y el CSV a la misma hora. Con el `wjsw` 1.2.0, el `.jsw` tardó 677, 1.494 y 1.114 ms en las tres consultas, de 10 a 31 veces el tiempo del Parquet: el filtro leía todas las columnas de cada fila. Corregido para leer solo la columna consultada, tardó 82, 311 y 209 ms. En la tercera ronda, la evaluación pasó a hacerse por bloque, comparando el código del diccionario y los enteros directamente en la columna y montando la fila solo cuando pasa: 48, 344 y 186 ms. En la cuarta, con el 1.3.11 todavía en preparación, el historial bajó de 355 a 279 ms. En la quinta, con el binario de la versión publicada, que compara el diccionario por lotes con instrucciones vectoriales y evalúa el «contiene» por bloque, el 1.3.11 tardó 44, 254 y 152 ms en el nivel estándar y 43, 207 y 142 ms en el máximo, frente a 54, 353 y 198 ms del 1.2.1 a la misma hora. El historial busca el número del proceso, un código formado solo por cifras que el 1.3.11 graba con ancho fijo (en DADOS_BIBLIOGRAFICOS, solo esa columna bajó de 53,0 a 25,7 MB). La máquina no estaba aislada, y el mismo binario varió hasta un 23% de una ronda a otra (el Parquet, de 41 a 54 ms en el conteo en vigor); por eso la comparación vale dentro de la misma ronda. Un archivo grabado con los

códecs nuevos no se abre en los lectores 1.2.x. Los tiempos de `grep` en el CSV variaron aún más (de 2.102 a 6.690 ms en el historial), porque los 12,5 GB de CSV no caben todos en la caché junto a los otros archivos, y en la quinta ronda las reescrituras justo antes sacaron el CSV de la caché; vale el número de la misma ronda.

La excepción en las cuentas es la de la clase 35. `grep` en el CSV, plano o comprimido, contó 1.076.929, uno más que la respuesta correcta: el trozo `,35,` también aparece en otra columna de una fila. Es el riesgo descrito junto a la Tabla 5B, que en aquel conjunto no ocurrió y aquí sí. Todos los lectores que separan las columnas (jsonW, Parquet, DuckDB leyendo el CSV y el XML con el nombre de la columna en la marca) dieron 1.076.928.

De esta conversión 1:1, el único recurso más allá del tipo y de la columna que entró fue la custodia: cada bloque de los siete archivos tiene CRC32C y SHA-256, verificados arriba. La conversión no marcó permisos ni montó búsqueda, para quedar igual al CSV. El formato permite, en el mismo archivo, ocultar el CPF y el CNPJ de los titulares a quien no tiene el alcance (Sección 4.3), leer solo una parte a distancia y buscar por términos y por significado (Sección 3). El Parquet tiene una suma de verificación opcional por página [36] y cifrado por columna [5], pero no la cadena de hashes del archivo entero ni la vista por alcance; el CSV y el XML no traen nada de esto.

5.5 Limitaciones de la medida

- **El lector de JSON es el nuestro.** El lado JSON lo grabó y lo leyó el propio código de texto de jsonW, que valida y convierte tipos; lee el JSONL a unos 140 MB/s. Los lectores de JSON de punta, como simdjson [23], superan 1 GB/s. Con uno de ellos, las filas "grabar" y "leer todo" de la Tabla 4 se reducirían bastante. Como estimación, no como medida: a 2 GB/s, leer los 519 MB llevaría unos 260 ms, y la razón de "leer una columna" estaría en las centenas, no en los miles.
- **Datos sintéticos y repetitivos,** como se dijo en la Sección 5.3. La razón de tamaño depende mucho de los datos, y más aún en el nivel máximo, que guarda la diferencia entre filas seguidas: mida con los suyos.
- **Una máquina, un sistema, memoria en lugar de disco.** Los tiempos absolutos cambian con el procesador, el almacenamiento y la carga; en las 10 ejecuciones, la mayor razón de cada operación quedó entre un 19% y un 75% por encima de la menor; la de leer una columna es la que más varía, porque el tiempo de jsonW ahí es de 1,6 ms.
- **Parquet medido solo con datos del INPI y un lector.** La comparación de rendimiento con Parquet está en las Tablas 5C y 5D, con DuckDB 2.0.0 como grabador y lector. Otro lector, otra compresión u otro tamaño de grupo de filas cambian los números. Arrow, MessagePack, CBOR y Protobuf siguen sin medida de rendimiento en esta versión del texto; la Sección 6 compara recursos, no velocidad.

5.6 Números publicados antes

Hasta el 12 de noviembre de 2026, el sitio y el repositorio de jsonW citaban "4,9 veces más pequeño que el JSONL" y "3.281 veces más rápido para leer un campo", de la primera medida, del 6 de noviembre, con la versión 1.0 en otra máquina. El texto 1.0 de este artículo, del 12 de noviembre, rehizo la medida con la 1.2.1: 6,6 veces más pequeño y unas 2.500 veces más rápido para leer una columna, con el nivel máximo en 3,75 MB con vector y 2,19 MB sin vector. Rehecha para este texto con la 1.3.11, en la misma máquina, da 6,6 veces más pequeño y unas 2.300 veces más rápido para leer una columna (la diferencia está dentro de la variación entre ejecuciones), y el nivel máximo baja a 1,36 MB y 17,5 kB. Valen los números de este texto, con la fecha, la máquina y la versión al lado.

5.7 Cómo reproducir

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

# Tabla 5: el mismo conjunto en JSONL, el .jsw y los compresores
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*
```

Listado 9. Las medidas de este texto en su máquina. `VET=0` y el esquema sin la columna `vetor` dan la columna "sin vector".

6 Comparación con otros formatos

La Tabla 6 compara lo que cada formato hace por sí mismo, sin bibliotecas encima. Los formatos en columnas maduros, Parquet y Arrow, ya tienen buena parte de la ventaja de acceso medida en la Sección 5, y varias cosas que jsonW todavía no tiene.

Tabla 6. Lo que cada formato trae en el propio formato. Fuentes en las referencias indicadas en cada columna.

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Forma	texto; una línea por registro en JSONL	binaria, sin esquema	binaria, esquema compilado (`.proto`)	archivo en columnas	columnas en memoria; archivo y flujo IPC	fila binaria, archivo en columnas y texto
Esquema	no (JSON Schema aparte)	no	sí, fuera del mensaje	sí, en el archivo	sí	sí, una línea, en el archivo
Leer un campo sin el resto	no	no	no	sí, por columna	sí, acceso directo	sí, por fila y campo
Decimal exacto	no	CBOR: sí (fracción decimal, tag 4)	no nativo	sí (DECIMAL)	sí (decimal128 y 256) [37]	sí
Dinero y medida	no	no	dinero con un mensaje común (`google.type.Money`) [10]	no	no	sí, con moneda y unidad
Vector denso	lista de números en texto	lista; CBOR tiene arrays tipados [34]	campo repetido	lista o bytes de tamaño fijo	lista de tamaño fijo	f32, f16, 8 y 4 bits, bits; tensor
Permisos	no	no	no	cifrado de columnas con claves distintas [5]	no	vista por columna, fila y celda, aplicada por quien sirve; sin cifrado
Integridad	no	no	no	CRC opcional por página [36]; cifrado autenticado	no en el formato	crc32c y hash por bloque, encadenados
Búsqueda y ranking en el formato	no	no	no	estadísticas por grupo de filas y página para saltar datos	no	zonas, coseno, disperso, rejilla, top-k, léxico
Lectura remota parcial	no	no	no	sí (pie y Range)	sí, en el formato de archivo	sí
Especificación	abierta (RFC)	abierta (CBOR: RFC)	abierta	abierta (Apache)	abierta (Apache)	uso, texto y ABI públicos; disposición binaria interna
Implementaciones	muchas, independientes	muchas, independientes	varias	varias, independientes	varias, independientes	una (núcleo en C), con SDK en 10 lenguajes

Dónde encaja cada formato. JSON es universal y legible en cualquier lugar. MessagePack, CBOR y Protobuf son maduros y tienen implementaciones en decenas de lenguajes, y Protobuf trae reglas maduras de evolución de esquema. Parquet y Arrow son estándares abiertos, leídos por casi todo motor de análisis, y Parquet tiene

cifrado por columna para el archivo en reposo. jsonW funciona con todos ellos: todo JSON ya es jsonW-texto, `wcon` trae y devuelve tablas de bases de datos con los tipos correctos, y el archivo en reposo puede cifrarse con el disco o el almacenamiento.

Lo que jsonW reúne. La diferencia de jsonW no es un recurso aislado, sino tener, en el mismo modelo, la forma de mensaje, la forma de archivo en columnas y una forma de texto compatible con JSON; tipos de negocio comprobados al grabar; permisos por columna, fila y celda aplicados al leer; y operaciones de ranking, búsqueda y clasificación dentro del archivo. El precio de esa elección es ser, por ahora, una sola implementación con el núcleo cerrado (Sección 8).

7 Usos

7.1 Modelos de IA y búsqueda semántica

Una base de conocimiento cabe en un `.jsw` con el texto, los metadatos y el vector de cada fragmento; la búsqueda semántica es un `avaliar … cosseno` que lee solo la columna del vector, y el léxico y los vectores dispersos cubren la búsqueda por términos. Con los k mejores y el léxico, el mismo archivo sirve de enrutador que elige al especialista: el léxico clasifica el texto de la petición y el ranking elige.

7.2 Datos de empresas

Los registros como los de empresas (CNPJ) y códigos postales (CEP) de Brasil son donde JSON pierde más: el cero inicial desaparece cuando el código se vuelve número, el capital social se vuelve double y la fecha pierde la zona horaria. En jsonW, cada cosa tiene su tipo:

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

Listado 10. Un esquema de registro de empresas: códigos exactos byte a byte, texto repetido en un diccionario, dinero con moneda y un tenant para separar clientes.

7.3 Bases de datos

`wcon`, el conector de bases de datos del paquete, lleva datos de PostgreSQL, MySQL y MariaDB, SQLite, Redis y Valkey y MongoDB a un `.jsw` y de vuelta, con el esquema tomado del tipo de cada columna: `numeric` se vuelve decimal exacto, `bigint` sigue siendo entero de 64 bits y el `vector` y el `halfvec` de pgvector [30] se vuelven `vf32` y `vf16`. Los valores siempre viajan como datos, con el tamaño delante, nunca pegados a un comando; la vuelta se hace en una sola transacción. La ida y vuelta se comprobó con PostgreSQL 14, 16 y 18 (mismo md5 en los dos lados), MariaDB 11.8 (hex de cada columna), SQLite, Redis 7 y MongoDB 7, con carga hostil: inyección, comillas, `\0`, acentos, emoji y texto de derecha a izquierda [20].

7.4 Servidores web

Con `wjsw-cgi` (Sección 4.6), un sitio sirve un `.jsw` como JSON, NDJSON o el propio archivo, con filtro, selección de campos y límite en la dirección, siempre con la vista de la sesión.

7.5 Almacenamiento en R2 y S3

Un `.jsw` puede vivir en un bucket de Cloudflare R2 o en cualquier S3 y consultarse allí, sin descargar el archivo entero (Figura 4). Como R2 no cobra la salida de datos [26], consultar un trozo de un archivo grande cuesta el trozo leído más las operaciones, que R2 cobra por petición (el Listado 6 hace 5 peticiones).

8 Limitaciones y próximos pasos

- **Una implementación, núcleo cerrado.** La disposición binaria no se publica; leer y grabar `.jsw` depende de `libjsonw`. La ABI estable y los SDK reducen el riesgo, pero no sustituyen una especificación abierta con implementaciones independientes.
- **Sin auditoría independiente.** Las pruebas citadas (fuzzing, cargas hostiles, comparación byte a byte entre SDK) son nuestras.
- **Plataformas.** Hay binarios para macOS arm64, Linux x86_64 (glibc 2.34 o más nueva) y Windows x86_64. Todavía no los hay para macOS Intel, Linux arm64 ni Linux con musl. En Windows, `wcon` y `wremoto` funcionan mediante WSL.
- **Recursos sin comando público.** El índice por clave, la grabación en vivo, la vista "solo lo vigente" y la compactación existen en el núcleo, pero no tienen comando en `wjsw` ni función en la ABI 1.3.11; el formato de la tabla de la rejilla no está documentado.
- **El protocolo no está en el paquete.** El protocolo poscuántico no viene en el paquete público 1.3.11.
- **Palabras en portugués.** Los tipos y los comandos usan palabras en portugués (`texto`, `dinheiro`, `medida`, `ver`, `conferir`), lo que pide un glosario para quien no lee portugués.
- **Bases de datos.** MySQL 8 funciona solo con `mysql_native_password`; la primera autenticación completa de `caching_sha2_password` todavía no existe.
- **Medidas.** Las limitaciones de la Sección 5.5.
- **Consulta por columna: el próximo paso.** En la base entera del INPI (Tabla 5D), `wjsw calcular` 1.3.11 cuenta los registros vigentes en 43 ms, por delante de DuckDB leyendo Parquet, y desde la primera ronda (1.2.0) ese conteo bajó de 677 a 43 ms. El próximo paso es llevar la misma lectura por bloque y en lotes al historial de un expediente y a la clase 35, donde el `.jsw` tardó 207 y 142 ms y DuckDB con ocho núcleos, 129 y 41 ms. En tamaño, los próximos códecs (diferencia y empaquetado de bits en enteros y fechas) apuntan a los 1.938,1 MB del nivel máximo.

Próximos pasos de este texto y del paquete: binarios para macOS Intel, Linux arm64 y musl; una consulta que compruebe solo el CRC32C de cada bloque y deje el SHA-256 para el `conferir` (el Parquet no comprueba ningún hash al leer, y parte de la diferencia que queda viene de ahí); el códec de diccionario con ventana, para columnas como la clase de Niza; la codificación por diferencias con empaquetado de bits en enteros y fechas, medida de nuevo frente a las Tablas 5C y 5D; y la comparación de rendimiento con Arrow en la misma máquina.

9 Disponibilidad y licencia

La herramienta, la biblioteca, la ABI, los SDK, las configuraciones de servidor web y la documentación están en github.com/developer-wortic/jsonw [20], y el SDK de Node.js está en npm como `@wortictecnologia/jsonw`. El uso se rige por la Licencia de Uso Sostenible de jsonW (en portugués), que prevalece sobre este resumen: jsonW es de uso libre en tantos servidores y aplicaciones como haga falta, incluidos productos comerciales, siempre que sea un componente y no el servicio que se vende; sin un acuerdo escrito con Wortic, nadie puede vender, alojar ni ofrecer a terceros jsonW ni un servicio cuya función principal sea la de jsonW (guardar, convertir, servir o transportar datos en jsonW), de pago o gratis; las bibliotecas y herramientas distribuidas como binarios no pueden someterse a ingeniería inversa, salvo lo que permita la ley; y la disposición interna del archivo, los códecs, el protocolo y el núcleo pertenecen a Wortic, por lo que una implementación independiente de ellos también requiere un acuerdo escrito. La documentación pública se puede leer, citar y usar libremente para construir programas que usen jsonW.

Por transparencia: la primera vez que se ejecuta en cada máquina, la herramienta `wjsw` envía solo la versión y la plataforma (por ejemplo `v=1.3.11&so=linux-x86_64`) para que podamos contar las instalaciones, y no lo repite después. La IP llega al servidor con la conexión pero no se guarda, ni ningún nombre, ruta o dato: el servidor

registra solo el día, la plataforma y la versión. `JSONW_SEM_PING=1` o el archivo `~/.jsonw/sem-ping` desactivan el aviso antes de la primera ejecución. La biblioteca y los SDK nunca acceden a la red.

Referencias

[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 el 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: repositorio público, documentación (docs/JSONW.md), ABI (sdk/jsonw.h) y binarios*. 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*, sección 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` de `PageHeader`). github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift

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

[38] Instituto Nacional da Propriedade Industrial (INPI, Brasil). *Datos abiertos de marcas* (`marcas.zip`, siete archivos CSV), descargados el 12/11/2026. dadosabertos.inpi.gov.br/download/marcas/marcas.zip

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

Cómo citar

Wortic Tecnologia. *jsonW: un modelo de datos, tres formas sin pérdida y los permisos dentro del formato*. Texto 1.1, sobre jsonW 1.3.11. 13 de noviembre de 2026. Disponible en: https://jsonw.org/es/paper.

```
@techreport{jsonw2026es,
  title       = {jsonW: un modelo de datos, tres formas sin p{\'e}rdida
                 y los permisos dentro del formato},
  author      = {{Wortic Tecnologia}},
  institution = {Wortic Tecnologia},
  year        = {2026},
  month       = nov,
  note        = {Texto 1.1, sobre jsonW 1.3.11},
  url         = {https://jsonw.org/es/paper}
}
```

Texto 1.1, publicado el 13/11/2026, sobre jsonW 1.3.11 (el texto 1.0, del 12/11/2026, trataba de la 1.2.1). jsonW es json Wondrous; el núcleo en C lo mantiene Wortic Tecnologia. Las correcciones y las medidas de terceros son bienvenidas a través del repositorio público. La versión original es la portuguesa; esta traducción mantiene los mismos números y términos.