Articolo tecnico · jsonW 1.3.11

jsonW: un modello di dati, tre forme senza perdita e i permessi dentro il formato

Wortic Tecnologia · 13 novembre 2026 · testo 1.1, su jsonW 1.3.11

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

INDICE

SOMMARIO

JSON e JSONL sono diventati la lingua comune dei dati, ma sono stati progettati per scambiare messaggi tra programmi, non per conservare e interrogare il volume e la varietà di dati che usano i sistemi di IA. Leggere un campo di un file JSONL obbliga a interpretare il file dall'inizio; denaro, misure e decimali esatti non hanno un tipo; i vettori diventano testo; i permessi e la prova di integrità restano fuori dal dato. Questo articolo descrive jsonW (json Wondrous) 1.3.11, un modello di dati con tre forme che si convertono senza perdita: riga, per messaggi e tempo reale; colonna, il file `.jsw` a blocchi letto tramite `mmap`; e testo, in cui ogni JSON valido è anche jsonW-testo valido. Il formato porta tipi di business, vettori e tensori, diciannove funzioni integrate (ricerca, ranking, manutenzione, grafia simile e segnali di audio, testo e video), viste per permesso e custodia tramite hash. Con 100.000 record sintetici, su una macchina, il file `.jsw` ha occupato 6,6 volte meno del JSONL non compresso e ha letto una colonna circa 2.300 volte più velocemente del nostro lettore JSON. Sulla base reale dei marchi dell'INPI, 55,3 milioni di righe e 12,53 GB di CSV, il `.jsw` è risultato 6,5 volte più piccolo del CSV e 12,8 volte più piccolo dell'XML, e ha contato i 2,3 milioni di record in vigore in 43 ms, più veloce di Parquet letto da DuckDB.

Parole chiave: formato di dati; archiviazione a colonne; JSON; ricerca vettoriale; controllo degli accessi; integrità; scambio di chiavi post-quantistico.

1 Introduzione

JSON [1] è nato per scambiare messaggi tra programmi in un testo che le persone possono leggere. JSON Lines [2] ha portato l'idea nei file: un record per riga. Entrambi sono diventati standard di fatto per log, esportazioni, insiemi di addestramento e risposte di API. Sono semplici, universali e leggibili, e queste qualità spiegano il loro successo. Lavorare con dati per l'IA, però, chiede cose che non sono stati fatti per dare.

1.1 Cosa manca a JSON e JSONL

- **Dimensione.** Tutto è testo. Un vettore di 384 dimensioni diventa migliaia di caratteri per record: i 100.000 record del test della Sezione 5 occupano 519,3 MB in JSONL.
- **Accesso.** Non c'è indice. Per leggere un campo di un record bisogna interpretare il file dall'inizio fino a quel punto; per leggere una colonna, il file intero.
- **Tipi.** La specifica non fissa la precisione dei numeri: la RFC 8259 lascia il limite a ogni implementazione e consiglia di contare solo sulla precisione di un double IEEE 754 [1, 28]. Non c'è un tipo per il denaro con valuta, la misura con unità, il decimale esatto, il codice che non può perdere lo zero iniziale, la data, il vettore o il tensore. Ogni sistema inventa una convenzione, e passare da una convenzione all'altra perde dati.
- **Permessi.** Chi può vedere ogni colonna e ogni riga vive fuori dal dato, in uno strato a parte, e ogni interrogazione deve ricordarsi di applicarlo.
- **Custodia.** Niente nel file prova che non è stato alterato.

In pratica, lo stesso dato passa per una catena: JSON sul cavo, CSV o Parquet per conservare, un database vettoriale per la ricerca semantica e uno strato di permessi per decidere chi vede cosa. Ogni passaggio costa tempo e memoria e, a volte, il dato stesso: il prezzo che diventa testo, il codice fiscale che perde lo zero iniziale, il peso di cui nessuno sa se era in grammi o in chili.

1.2 Contributi

Questo articolo descrive jsonW (json Wondrous) versione 1.3.11. I suoi contributi sono:

1. un modello di dati con tre forme che si convertono senza perdita (riga, colonna e testo), in cui ogni JSON valido è anche jsonW-testo valido (Sezione 2);
2. tipi di business e di IA dichiarati nello schema stesso (Sezione 2.3);
3. diciannove funzioni dentro il file, dalla ricerca e dal ranking ai segnali di audio, testo e video, mostrate con lo strumento pubblico quando questo ha un comando per esse (Sezione 3);
4. permessi per colonna, riga e cella e custodia tramite hash come parte del formato (Sezione 4);
5. misure riproducibili, con metodologia, dati e i limiti della misura (Sezione 5), e un confronto con JSON, JSONL, MessagePack, CBOR, Protobuf, Parquet e Arrow (Sezione 6).

1.3 Cosa questo testo non descrive

Il nucleo di jsonW è scritto in C, senza dipendenze oltre la libreria standard, ed è mantenuto da Wortic Tecnologia; non è open source. Ciò che è pubblico, e ciò che questo testo usa, è la documentazione d'uso, l'ABI in C (`sdk/jsonw.h`), gli SDK, i binari dello strumento e il comportamento osservabile [20]. La disposizione binaria interna, i codec di compressione e l'organizzazione interna di un blocco sono fuori dall'ambito. Ciò che affermiamo sul formato e sullo strumento si può verificare con i binari pubblici; le prove interne (fuzzing, carichi ostili, confronto tra SDK) sono nostre e segnalate come tali. Quando una funzione esiste nel formato ma non ha ancora un comando nello strumento pubblico, lo diciamo.

2 Modello di dati

2.1 Tre forme

jsonW è un unico modello di dati con tre modi di conservare lo stesso valore (Figura 1). Passare da una forma all'altra non perde alcun valore né cambia il tipo di alcun valore.

- **Riga:** binaria, un valore dopo l'altro, per messaggi, eventi e tempo reale. Prende il posto di JSON, MessagePack, CBOR e Protobuf. È la forma che viaggia sul cavo (Sezione 4.5).
- **Colonna:** il file `.jsw` , a blocchi conservati per colonna e letti direttamente dal disco tramite `mmap` , senza copia. Prende il posto di JSONL, CSV e Parquet.
- **Testo:** jsonW-testo, perché le persone leggano e scrivano. Prende il posto di JSON, TOML e YAML. Ogni JSON valido è già jsonW-testo valido; il testo aggiunge notazioni per i tipi che mancano a JSON, come `12.50 BRL` , `250 g` e `t"2026-11-09T01:00:00.000000Z"` .

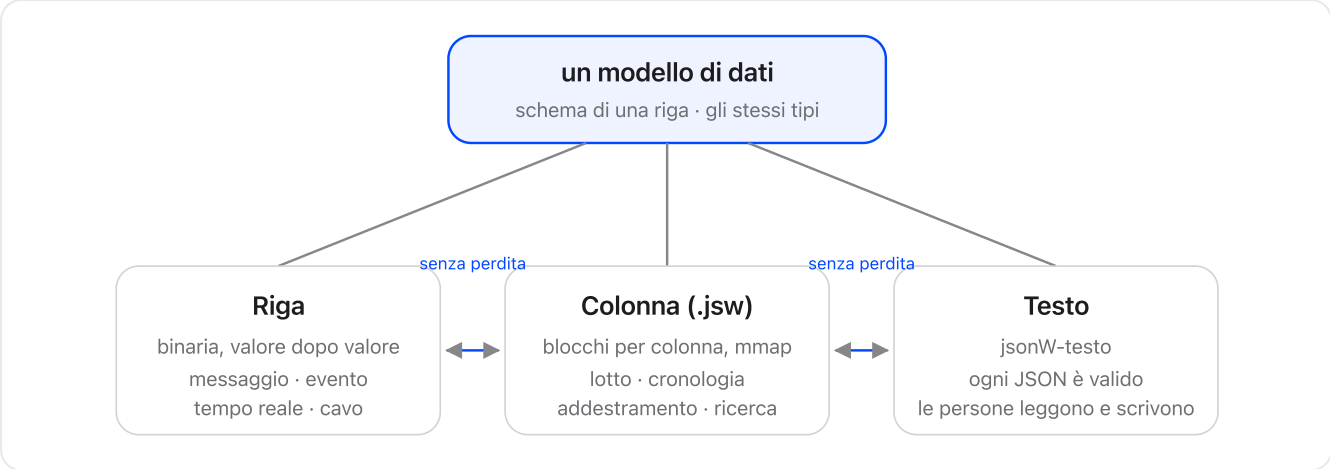


Figura 1. Un modello, tre forme. Schema e tipi sono gli stessi in tutte e tre; convertire dall'una all'altra non perde alcun valore né cambia alcun tipo.

2.2 Schema

Lo schema è una riga di testo con coppie `nome:tipo` separate da virgole. Il punto interrogativo accetta il nullo, e i segni dopo il tipo dichiarano i permessi (Sezione 4.3). I nomi dei tipi e dei comandi sono parole portoghesi: `texto` è testo, `dinheiro` è denaro, `medida` è misura, `codigo` è codice, `tempo` è tempo, `pessoal` è personale.

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

Listato 1. Uno schema. `cpf` (un codice fiscale personale brasiliano) accetta il nullo ed è un dato personale: senza l'ambito `pessoal`, esce nullo.

Lo schema evolve senza rompere i lettori: un campo nuovo è nullo nelle righe vecchie, un nome non cambia mai tipo e il lettore risolve i campi per nome. Così un file vecchio viene letto da codice nuovo, e un file nuovo da codice vecchio [20].

2.3 Tipi

La Tabella 1 elenca i tipi della versione 1.3.11. I tipi di business vengono verificati in scrittura: un ingresso che non rientra nel tipo viene rifiutato con il motivo, invece di essere conservato male.

Tabella 1. Tipi di jsonW 1.3.11 [20].

Tipo	Nello schema	Cosa garantisce
decimale esatto	`decimal[2]`	non passa mai per la virgola mobile: 12.50 resta 12.50
denaro	`dinheiro`, `dinheiro[4]`	valuta in ogni valore, per codice ISO 4217 [27] o per simbolo (`12,50 BRL`, `US$`, `€`, `₿`); somma solo dentro la stessa valuta
misura	`medida`, `medida[kg]`	89 unità in 14 grandezze; converte in scrittura e rifiuta un'altra grandezza
codice	`codigo`	byte esatti: codici fiscali, chiavi, targhe e SKU non vengono mai normalizzati né tagliati
testo	`texto`, `texto+nfc`	Unicode completo, con accenti, emoji e `\0`; `+nfc` normalizza su richiesta [29]
testo ripetuto	`dic`	dizionario per file: ogni testo conservato una volta
tempo	`tempo`	microsecondi, UTC
url	`url`	validata in scrittura (`https://…`, `mailto:`, `tel:`, `urn:`; fino a 8 KiB)
vettore	`vf32[n]` `vf16[n]` `bits[n]` `vq8[n]` `vq4[n]`	matrice contigua nel file, pronta per la ricerca semantica
tensore	`tensor<bf16>[768,768]`	elementi f32, f16, bf16, u8, i8, i16, i32
media	`midia`	hash, tipo, dimensione, durata, larghezza e altezza; jsonW non fissa il codec: il media resta nel suo codec aperto
e altro	`bool` `i64` `f64` `f32` `uuid` `bytes` `ref` `lista<t>` `obj{…}` `mapa<k,v>` `qualquer`	`qualquer` (qualsiasi) conserva JSON aperto; annidamento fino a 1000 livelli

Le 14 grandezze di misura sono lunghezza, massa, volume, energia, tempo, area, dati, velocità, temperatura, adimensionale, potenza, pressione, carbonio e densità. La conversione è esatta tra multipli decimali (12,5 mm = 0,0125 m) e arrotondata alla scala della colonna negli altri casi (1 mi = 1609,344 m, conservato come 1609 m in una colonna `medida[m]`, che non ha decimali; 25 °C = 77 °F).

2.4 Virgola o punto

I lettori accettano entrambi i separatori: `12,50`, `1.234,56 BRL` e `1,234.56 USD`. Quando ci sono entrambi, l'ultimo è il decimale. Con uno solo, quel separatore è il decimale, tranne una virgola seguita da tre cifre in una valuta col punto (`USD 1,234` è mille) o un punto seguito da tre cifre in una valuta con la virgola (`R$ 1.234` è mille). L'uscita canonica usa sempre il punto. E i numeri reali non dipendono mai dalle impostazioni locali del processo: una JVM o un PHP configurati in portoghese non scambiano il separatore.

2.5 Un esempio eseguito

Il Listato 2 mostra il percorso da JSONL a `.jsw` e ritorno, con l'uscita reale dello strumento `wjsw` 1.2.1.

```
# due righe di JSONL, come arrivano da qualsiasi 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"}

# ciò che non rientra nel tipo viene rifiutato, con il 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"}
```

Listato 2. `1 kg` è diventato `1000 g` perché la colonna è `medida[g]`; il litro è volume e la colonna è massa, quindi la riga è stata rifiutata ("misura di un'altra grandezza"; il messaggio mostra l'ingresso vicino all'errore). Il denaro conserva la sua valuta e lo SKU viene conservato byte per byte. Lo strumento scrive i messaggi in portoghese.

3 Le 19 funzioni

Le funzioni qui sotto fanno parte del formato e del nucleo, non di uno strato sopra, e lo strumento pubblico ne espone la maggior parte (Tabella 2). Sono i pezzi che, in altre pile, sono sparsi tra un motore di ricerca, un database vettoriale, un indice e uno strato di permessi. Gli esempi di questa sezione sono stati eseguiti con il binario pubblico di `wjsw` 1.2.1 per Linux x86_64 (il Listato 6B con la 1.3.11), e gli ingressi sono su jsonw.org/paper/exemplos; le uscite lunghe sono state tagliate dove compare `…`.

Tabella 2. Le 19 funzioni e dove compaiono nello strumento e nell'ABI pubblici 1.3.11.

#	Funzione	Cosa fa	Nello strumento
1	Ricerca per termini	vettore sparso con pesi per termine (IDF), il calcolo usato dai motori di ricerca	`avaliar … esparso`
2	Ricerca semantica	coseno su vettori densi `vf32`, `vf16`, `vq8` e `vq4`	`avaliar … cosseno`
3	Griglia multilineare	interpolazione bilineare, trilineare e superiore, direttamente dal file	`avaliar … grade`
4	Punteggio su richiesta	pesi, coseno, sparso o griglia in una richiesta di una riga	`avaliar`
5	I primi k	top-k deterministico con heap: il punteggio più alto prima, a parità vince la riga minore	`avaliar`, `escolher`
6	Zone	minimo e massimo per blocco e colonna: la ricerca salta blocchi interi	`filtrar`, `info`
7	Dizionario	testo ripetuto conservato una volta per file	tipo `dic`
8	Indice per chiave	va diritto alla riga di una chiave, senza scorrere	nel nucleo; senza comando proprio
9	Lessico	automa Aho-Corasick letto dal disco; parola intera, senza distinguere accenti e maiuscole	`reconhecer`
10	Scrittura dal vivo	un processo legge mentre un altro scrive, senza mai vedere mezzo blocco	nel nucleo; senza comando proprio
11	Vivo, cancellato e cronologia	scrivere la stessa chiave modifica; `_apagado: true` cancella; la compattazione riscrive solo ciò che vale	nel nucleo; senza comando proprio
12	Vista per permesso	colonna, riga e cella, secondo la credenziale	`ver`, `wjsw-cgi`, SDK
13	Custodia	crc32c e SHA-256 (o BLAKE3) per blocco, concatenati	`conferir`, `reparar`
14	Lettura remota parziale	con intestazione e piè di pagina, scarica solo i blocchi delle righe lette	`wremoto`
15	Grafia e suono simili	punteggio da 0 a 1 per trigrammi della grafia e di una chiave fonetica, in portoghese del Brasile, inglese americano, spagnolo e cinese	`filtrar … ~~`
16	Segnali dell'audio	tono, pause, intensità, jitter, shimmer, HNR, formanti, MFCC, centroide, velocità di articolazione e respiri	ABI: `jsonw_sinal_audio`
17	Segnali del testo	intensificatori, attenuatori, polarità, assoluti, negazione, voce passiva, pronomi, forma e indizi di ironia, in pt, en, es e zh	ABI: `jsonw_sinal_texto`
18	Video in scene	tagli di scena, le misure e i segnali dell'audio di ogni tratto e un mosaico dei fotogrammi	ABI: `jsonw_video_*`
19	Colonna per blocco e per tenant	dizionario locale per blocco, cifre a larghezza fissa, zona del tenant e confronto a lotti	in scrittura; `filtrar`, `calcular`

3.1 Ricerca e ranking (funzioni da 1 a 5)

Il comando `avaliar` (valutare) dà un punteggio a ogni riga e restituisce le prime k. La richiesta può essere una somma pesata di colonne numeriche, il coseno tra una colonna di vettori e una query, il prodotto tra vettori sparsi o una consultazione di una griglia. La stessa richiesta esiste in JSON, identica nello

strumento e nell'ABI, e tutti gli SDK restituiscono lo stesso risultato, byte per byte [20]. Il calcolo legge solo le colonne della richiesta, in parallelo. Il Listato 3 usa cinque prodotti con una colonna di vettori `vf16[4]` e una colonna di termini `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. punteggio per pesi (il prezzo conta contro, la valutazione a favore) e i primi 3
$ 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. ricerca semantica: coseno con la query
$ 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. ricerca per termini: prodotto tra vettori sparsi (termine 12 e termine 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 stessa richiesta in JSON, come nell'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. i 2 valori più alti di una colonna
$ wjsw escolher prod.jsw nota 2
4.9     3       {id:4,produto:"Bolo de milho",…}
4.8     0       {id:1,produto:"Café especial",…}
```

Listato 3. Uscita: punteggio, posizione della riga e la riga in jsonW-testo, dalla migliore alla peggiore.

Due dettagli dell'uscita meritano attenzione. Primo, il vettore conservato come `vf16` torna come `0.7998046875` dove l'ingresso diceva `0.8`: la mezza precisione conserva circa tre cifre decimali. Quando l'esattezza del vettore conta, la colonna deve essere `vf32`; `vq8` e `vq4` scambiano più precisione con dimensione. Secondo, la ricerca per termini è un prodotto scalare: i pesi IDF [22] sono conservati nel vettore di ogni riga, e la query porta i suoi.

Griglia (funzione 3). Una griglia è una tabella di valori su assi (per esempio, il costo di spedizione per distanza e peso). La richiesta `{"grade":{"campos":["x","y"],"arquivo":"g.jsw"}}` usa le colonne della riga come coordinate e restituisce, come punteggio, il valore interpolato tra i punti vicini della griglia: bilineare con due assi, trilineare con tre, e così via. Nella versione 1.3.11, il formato della tabella della griglia non è ancora nella documentazione pubblica, quindi qui non ne mostriamo l'uscita.

3.2 Organizzazione del file (funzioni da 6 a 8)

Zone. Ogni blocco conserva il minimo e il massimo di ogni colonna. Un filtro che un blocco non può soddisfare fa saltare al lettore il blocco intero. Nel file di 100.000 record della Sezione 5, con 103 blocchi:

```
$ 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
```

Listato 4. Nel file di 100.000 record (id da 0 a 99.999), le zone saltano 102 blocchi su 103 ("pulados pela zona"); nel file dei prodotti del Listato 3, `info` mostra il dizionario della colonna `categoria` e le zone, con i reali in doppia precisione come li stampa lo strumento.

Dizionario. Una colonna `dic` conserva ogni testo distinto una volta per file. Nei 100.000 record del test, le colonne `tipo` e `fonte` sommano 10 testi distinti, e `wjsw info` mostra `dicionario 10 textos`.

Indice per chiave. Il formato ha un indice che va diritto alla riga di una chiave, senza scorrere il file. Nello strumento pubblico 1.3.11 non ha comando né funzione nell'ABI; l'uso visibile delle chiavi è `comparar` (confrontare), che accoppia le righe di due file per una colonna e conta le differenze:

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

Solo in a: 1 · solo in b: 1 · cambiato: 1 · uguali: 0.

3.3 Lessico (funzione 9)

Una tabella di termini, con lo schema `termo:texto,forma:dic,sinal:dic,peso:f32,idioma:dic,origem:dic`, viene compilata, all'apertura, in un automa Aho-Corasick [21] che trova tutti i termini di un testo in un solo passaggio. La corrispondenza è per parola intera, senza distinguere accenti e maiuscole, e restituisce il tratto originale e la somma dei pesi per segnale: un classificatore di testo senza addestramento, utile per lo smistamento e per portare una richiesta allo specialista giusto.

```
$ 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)
```

Listato 5. Una frase in portoghese ("Voglio ANNULLARE l'ordine, oppure dimmi quanto costa il nuovo preventivo"). "CANCELAR" ha trovato "cancelar", e "orcamento" ha trovato "orçamento", perché la corrispondenza ignora accenti e maiuscole. I segnali sono "intenzione di acquisto" e "rischio di perdita".

3.4 Scrittura e cronologia (funzioni 10 e 11)

Un file `.jsw` ammette solo aggiunte. Questo dà due garanzie. La prima è la **scrittura dal vivo**: un processo può leggere il file mentre un altro lo scrive, come un `tail -f` che non vede mai mezzo blocco. La seconda è la **cronologia**: scrivere di nuovo la stessa chiave è una modifica, scrivere la chiave con `_apagado: true` è una cancellazione, e la compattazione riscrive solo ciò che vale in un file nuovo, lasciando la cronologia nell'originale.

Nella versione 1.3.11, la lettura durante la scrittura, la vista "solo ciò che vale" per chiave e la compattazione non hanno comando in `wjsw` né funzione nell'ABI pubblica. Con lo strumento pubblico, la cronologia appare così com'è, riga dopo riga:

```
$ 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}
```

Ciò che l'ABI pubblica garantisce in scrittura è la **pubblicazione atomica**: il file compare, intero, solo in `fechar()` (chiudere). Uno scrittore che esce per un errore a metà chiama `abandonar()` (abbandonare), e la destinazione resta com'era.

3.5 Permessi, custodia e accesso remoto (funzioni da 12 a 14)

La vista per permesso (12) e la custodia (13) sono l'argomento della Sezione 4. La **lettura remota parziale** (14) usa richieste HTTP `Range` [24]: il lettore scarica l'intestazione e il piè di pagina, dove ci sono lo schema, l'indice dei blocchi e le zone, e poi solo i blocchi che gli servono (Figura 4). Abbiamo misurato con `wremoto` 1.2.1 contro un nginx locale che serviva il file di 100.000 record (il `com.jsw` della Sezione 5, con il nome `cartoes.jsw`); con `file://` al posto dell'indirizzo HTTP, `wremoto` dà gli stessi numeri, senza server:

```
$ 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)
```

Listato 6. Per leggere una riga dal mezzo di un file di 79,1 MB, il lettore ne ha scaricato lo 0,98% ("baixado no total"); per il filtro, lo 0,84%, perché le zone hanno scartato gli altri blocchi prima dello scaricamento.

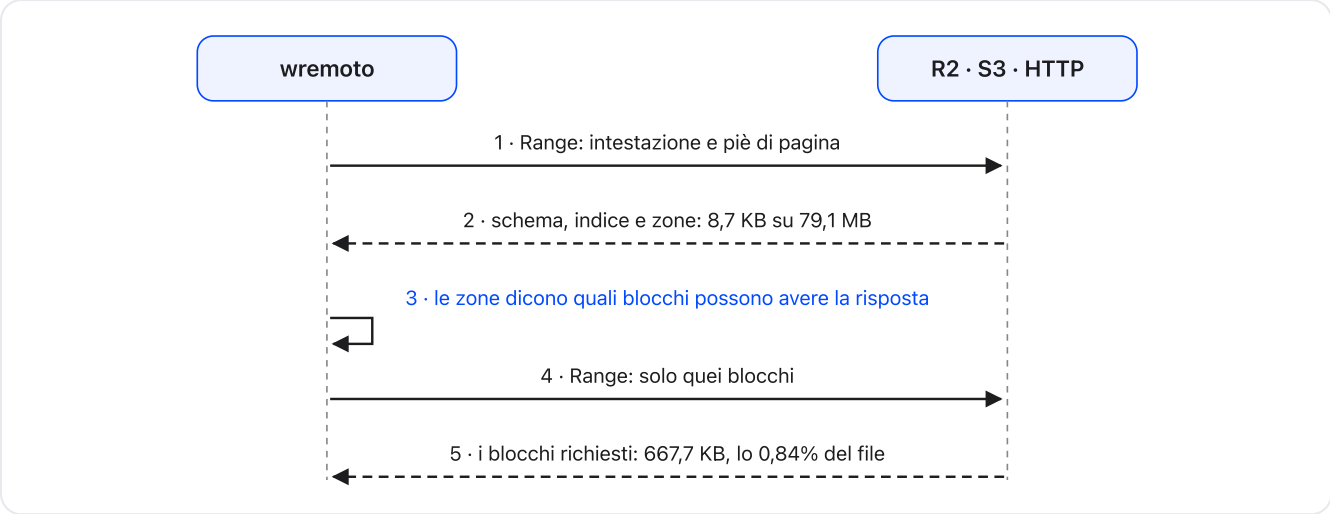


Figura 4. Lettura remota parziale del file di 100.000 record, con i numeri misurati nel Listato 6. Con R2 o S3, ogni richiesta è firmata con AWS SigV4 [25].

3.6 Grafia, segnali e colonna per blocco (funzioni da 15 a 19)

Grafia e suono (funzione 15). L'operatore `~~` del filtro dà a ogni valore un punteggio da 0 a 1: il maggiore di due misure di Dice su trigrammi di caratteri Unicode, una sulla grafia normalizzata (senza accenti né maiuscole) e una su una chiave fonetica della lingua richiesta. C'è una chiave per il portoghese del Brasile, l'inglese americano, lo spagnolo e il cinese scritto in pinyin; negli ideogrammi i trigrammi contano carattere per carattere, e la loro chiave del suono, che dipende da una tabella di letture, non è ancora in questa versione. Il filtro non usa indici: scorre solo la colonna richiesta, in parallelo (Listato 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"}
```

Listato 6B. Grafia e suono simili. "Kokakola" trova "Coca-Cola": la chiave del portoghese porta la "c" dura e la "k" allo stesso suono. `~~en`, `~~es` e `~~zh` cambiano la lingua; `~~pt@0.4` cambia la soglia.

Segnali della comunicazione (funzioni da 16 a 18). Misurano come la persona ha parlato, scritto o è apparsa, e non decidono nulla: chi interpreta è il modello che legge la riga. Nell'audio (PCM a 16 bit), il tono viene dall'autocorrelazione normalizzata, lo stesso metodo di Praat; jitter e shimmer dagli impulsi glottali; le formanti dalla predizione lineare (LPC); e le pause dalla soglia di Otsu sull'intensità, che ricava il taglio dai dati stessi invece che da un numero fisso. Nel testo, il lessico della lingua (pt, en, es o zh, scritto a mano) viene riconosciuto dall'automa della funzione 9, e i conteggi di forma (maiuscole, punteggiatura, emoji, risate) non usano dizionario. Il cinese, che non separa le parole con spazi, viene riconosciuto ideogramma per ideogramma. Nel video, chi chiama decodifica e consegna i fotogrammi e il suono; ogni fotogramma diventa una miniatura con luce, contrasto, colore e forma, e i tagli di scena escono dalla stessa soglia di Otsu sulla differenza tra fotogrammi, solo quando la serie ha due gruppi netti. Ogni misura esce come una riga di jsonW-testo nello schema dichiarato, pronta da scrivere in un `.jsw`.

Colonna per blocco e per tenant (funzione 19). Tre nuovi codec: il dizionario locale di ogni blocco, con e senza codifica entropica, per il testo che si ripete dentro il blocco ma non in tutto il file; e le cifre a larghezza fissa, per codici fatti solo di cifre, come il numero di una domanda. La zona di ogni blocco conserva anche il tenant e, con la vista di una sessione, il blocco di un altro cliente non viene nemmeno

aperto. Il filtro e il calcolo confrontano il codice del dizionario a lotti, con istruzioni vettoriali (SSE2), e valutano il «contiene» sull'intero blocco in una volta. Un file scritto con questi codec non si apre nei lettori 1.2.x. Nella stessa versione, il livello massimo (`--max`) ha guadagnato due codec che scrivono ogni cella e ogni vettore come differenza rispetto alla riga precedente: il prefisso e il suffisso comuni e solo la parte centrale che cambia. L'effetto sulla base dell'INPI è nelle Tabelle 5C e 5D.

4 Sicurezza e custodia

4.1 Custodia

Ogni blocco del file sta in piedi da solo e porta due verifiche: un CRC32C [11], veloce, per i danni accidentali, e un hash crittografico, SHA-256 [12] come predefinito o BLAKE3 [13] come opzione. Gli hash formano una catena che copre il file intero: qualsiasi byte cambiato nei dati, in una zona o nel dizionario fa fallire `conferir` (verificare) (Figura 2).



Figura 2. Il file `.jsw` visto da fuori, al livello descritto nella documentazione pubblica. L'organizzazione interna di intestazione, blocchi e piè di pagina non è pubblicata (Sezione 1.3).

```
$ wjsw conferir vendas.jsw
ok: 2 linhas, 1 blocos, sha256 e crc32c conferidos
# un byte cambiato a metà del file
$ wjsw conferir adulterado.jsw
FALHOU: bloco 0: nao veio ou nao confere
# gli ultimi 20 byte tagliati, come in una scrittura interrotta
$ 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 al posto di 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
```

Listato 7. Custodia, manomissione e riparazione, con l'uscita reale ("FALHOU" = fallito; "sem rodape: rode wjsw reparar" = senza piè di pagina: esegui wjsw reparar).

> La custodia prova l'integrità, non la paternità. Un hash senza chiave può essere ricalcolato da chi riscrive il file intero. Per provare che un file è lo stesso uscito da una fonte, conservate a parte l'identificatore del file mostrato da `wjsw info` , o firmatelo con uno strumento a parte, e confrontate.

4.2 Lettura di file ostili

Il lettore tratta ogni file come ostile: verifica ogni indice e ogni limite prima dell'uso, rifiuta annidamenti oltre 1000 livelli ed è testato con il fuzzing (ingressi generati a caso) nella batteria di test. Virgolette, virgole, barre e qualsiasi byte dentro un testo sono dati e non rompono mai la lettura.

4.3 Vista per permesso

In jsonW, i permessi sono dichiarati nello schema e applicati da chi serve il file, prima di rispondere: ciò che la sessione non può vedere non esce mai dal server (Figura 3, Tabella 3).

Tabella 3. Gli strati della vista per permesso.

Strato	Come si dichiara	Cosa succede
Colonna	`+escopo(financeiro)`, `+pessoal`	senza l'ambito, la colonna esce nulla e non si può usare nemmeno in un filtro
Riga	una colonna intera chiamata `tenant` o `tenant_id`, o marcata `+tenant`	solo le righe del tenant della credenziale; i blocchi degli altri tenant non vengono nemmeno aperti; `tenant:todos` (tutti) apre tutto; il tenant è un intero: con una colonna `tenant` di testo, non esce nessuna riga
Cella	le due insieme	le righe del cliente stesso, senza la colonna dei dati personali

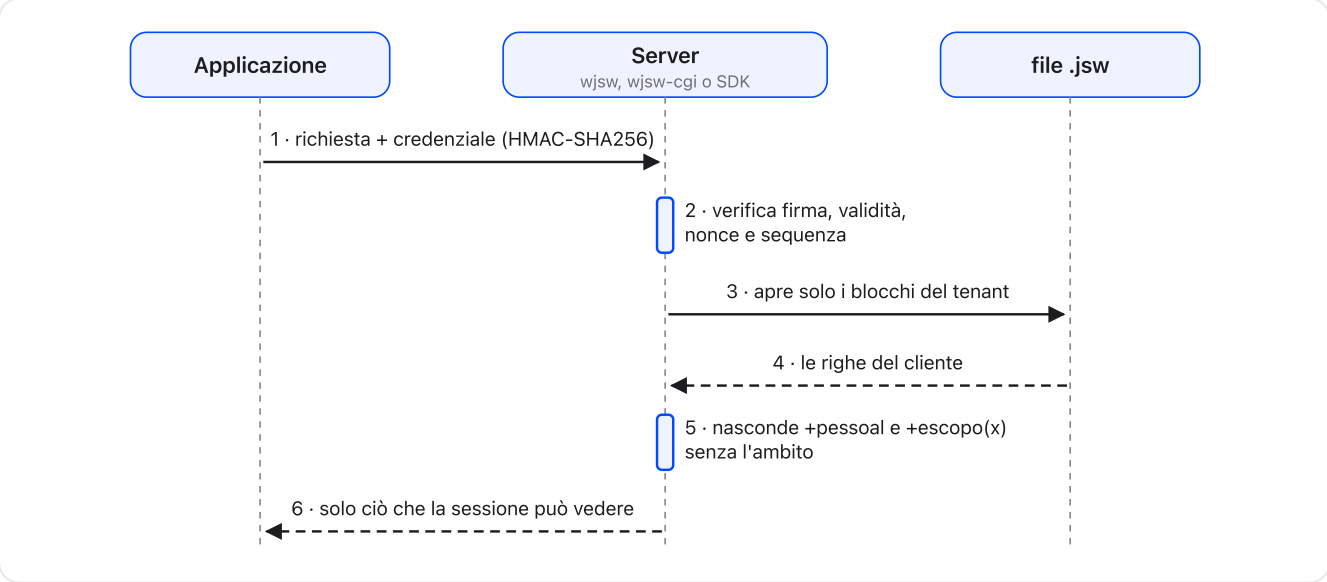


Figura 3. La vista per permesso. I passi da 2 a 5 avvengono dentro il server; il file non esce mai intero verso chi lo chiede.

Una clinica con due sedi mostra i tre strati. L'accettazione della sede 1 vede l'agenda della sede 1 senza codice fiscale e senza importo; l'amministrazione della sede 1 vede l'importo, sempre senza codice fiscale; nessuno della sede 1 vede la sede 2:

```
$ wjsw de-texto "paciente:texto,cpf:codigo?
+pessoal,valor:dinheiro+escopo(financeiro),data:tempo,tenant:i64" clin.jsonl clin.jsw
# accettazione della sede 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}
# amministrazione della sede 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}
```

Listato 8. La stessa interrogazione, due credenziali, due risposte. I dati dell'esempio sono fittizi.

> La vista non è crittografia. Protegge ciò che esce dal server; chi ha il file `.jsw` ha tutti i dati, e `para-texto` esporta tutto, perché è la copia per chi possiede già il file. Per mostrare dati a un utente, usate `ver`, `wjsw-cgi` o l'SDK con `visao()`; per proteggere il file a riposo, usate la crittografia del disco o dell'archiviazione.

4.4 Credenziale

La credenziale che accompagna una richiesta è essa stessa un valore jsonW, con emittente, soggetto, tenant, ambiti e validità, firmata con HMAC-SHA256 [18]. La firma viene verificata in tempo costante, e un nonce con un numero di sequenza impedisce che una richiesta venga ripetuta. Viaggia sopra TLS, non al suo posto.

4.5 Trasporto post-quantistico

La forma a riga viaggia su un protocollo sopra QUIC [15] con TLS 1.3 [16] e lo scambio di chiavi ibrido X25519MLKEM768 [17], che combina la curva X25519 con ML-KEM-768, standardizzato dal NIST nel FIPS 203 [14]. L'argomento post-quantistico ha due parti. Lo scambio di chiavi usa un meccanismo progettato per resistere a un computer quantistico. E la custodia e le credenziali usano solo hash e HMAC a 256 bit, che non dipendono dalla fattorizzazione né dal logaritmo discreto; per trovare una preimmagine o la chiave dell'HMAC, l'algoritmo di Grover [19] dà un'accelerazione quadratica e lascia circa 128 bit di sicurezza; per trovare collisioni, il limite quantistico noto è più basso, circa 2^{85} operazioni su SHA-256 con l'algoritmo di Brassard, Høyer e Tapp [35], che richiede una memoria quantistica enorme.

L'ultima riga di `wjsw medir` (Sezione 5) misura la forma a riga: un record di test occupa 5.157 byte in JSON e 1.073 byte come messaggio, 4,8 volte meno, in tutte le esecuzioni.

> Il pacchetto pubblico 1.3.11 non include né un server né un client del protocollo.

4.6 Server web

Il pacchetto include `wjsw-cgi`, che funziona su Apache, nginx, LiteSpeed, Caddy e IIS [20]. Ogni richiesta di `.jsw` passa da lì: un file senza colonne marcate e senza tenant esce senza filtro, con `Range` e CORS; gli altri escono filtrati, con la vista applicata. L'ambito viene dal server dopo l'autenticazione, mai da un'intestazione inviata dal client, e chiedere una colonna nascosta tramite `campos=` o `onde=` restituisce 403, così nulla trapela per deduzione. Nella versione 1.2.0, la configurazione d'esempio lasciava che il server consegnasse il file grezzo quando la richiesta non aveva parametri; la 1.2.1 l'ha corretto, e chi ha installato la configurazione vecchia deve sostituirla [20].

5 Misure

Questa sezione misura jsonW 1.3.11 rispetto a JSONL e dice, accanto, ciò che la misura non prova. I numeri sono stati rifatti per questo testo il 13 novembre 2026, con il binario della versione pubblicata, sulla stessa macchina della misura del 12 novembre.

5.1 Metodologia

- **Strumento:** `wjsw medir` (misurare), incluso nel binario pubblico, con 100.000 righe: `wjsw medir <cartella> 100000`. Binario `bin/linux-x86_64/wjsw` del repository pubblico, versione 1.3.11 (pre-rilascio, SHA-256 `cb119d7473e7d786…`).
- **Macchina:** un server virtuale con un AMD EPYC 9355P, 8 vCPU e 18 GB di memoria, Ubuntu 26.10, kernel Linux 7.3. I file stavano su un `tmpfs` (memoria), quindi la misura esclude il disco. La macchina non era isolata da altri carichi.
- **Dati:** 100.000 record sintetici, con la forma di una scheda di una base di conoscenza: `id:i64,tipo:dic,titulo:texto,corpo:texto,fonte:dic,valor:decimal[4],vetor:vf16[384],tags:lista<texto>`. I testi ripetono un modello con il numero della riga; `tipo` e `fonte` ruotano tra 5 valori; il vettore ha 384 componenti multipli di 1/1024, che `vf16` conserva senza perdita.
- **Il lato JSON:** lo stesso insieme scritto come JSONL e letto dal lettore di testo di jsonW stesso, che interpreta ogni riga e converte i tipi dello schema. Entrambi i file vengono caricati in memoria prima che parta l'orologio.
- **Ripetizione:** 10 esecuzioni consecutive. Riportiamo la mediana di ogni tempo e, per i rapporti, la mediana e l'intervallo sulle 10. Le uscite grezze sono pubblicate con questo testo.
- **Verifica:** in ogni esecuzione lo strumento verifica che le due letture abbiano contato 100.000 righe con la stessa somma.

5.2 Tempo

Tabella 4. Tempo per scrivere e leggere i 100.000 record. Tempi: mediana di 10 esecuzioni. Rapporto: JSONL ÷ jsonW in ogni esecuzione, con la mediana e l'intervallo sulle 10. Nella riga a 8 thread, il rapporto è rispetto alla lettura del JSONL in un thread. Il JSONL è scritto e letto dal codice di testo di jsonW stesso.

Operazione	JSONL	jsonW	Rapporto (intervallo)
Scrivere	15.211 ms	131 ms	114× (da 88 a 127)
Leggere tutto, tutti i campi	3.687 ms	78 ms	46× (da 43 a 51)
Leggere tutto, 8 thread	(nessuna versione parallela)	9,0 ms	419× (da 365 a 532)
Leggere una colonna (`valor`)	3.687 ms	1,59 ms	2.310× (da 1.592 a 2.784)
Andare alla riga N, campo `titulo`	letto dall'inizio	0,09 µs (da 0,05 a 0,16)	rapporto non definito

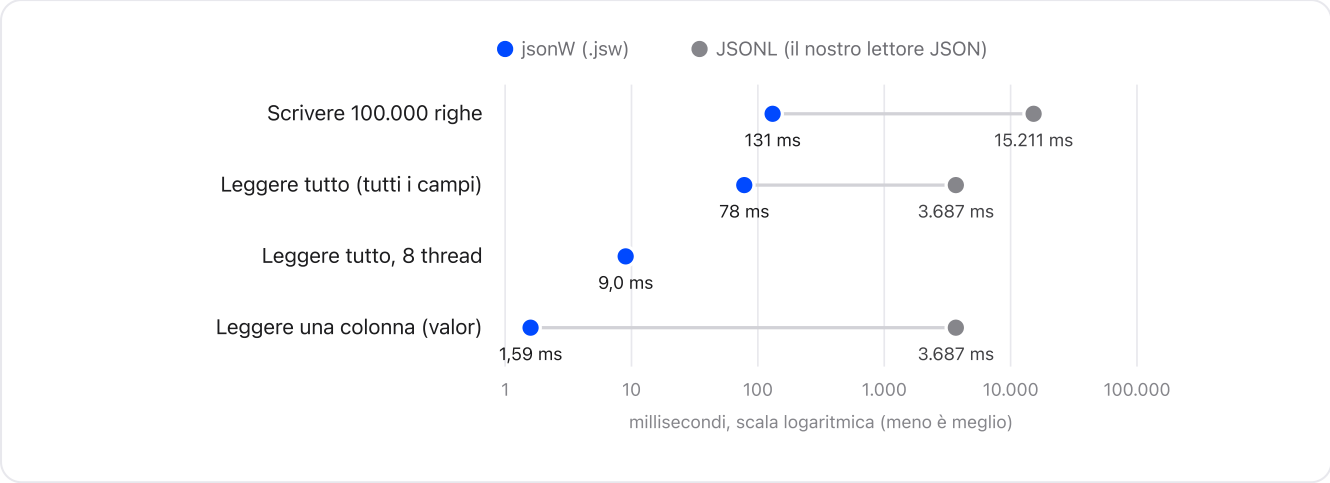


Figura 6. I tempi della Tabella 4 in scala logaritmica (mediana di 10 esecuzioni). La distanza orizzontale tra i punti di ogni riga è il rapporto.

La Figura 6 mostra gli stessi tempi in scala logaritmica. La differenza più grande è nella lettura di una colonna: il JSONL obbliga a interpretare ogni riga intera per estrarre un campo, e il file a colonne legge solo quella colonna. È un vantaggio strutturale, lo stesso che qualsiasi formato a colonne ha rispetto a un formato a righe. Per "andare alla riga N", il JSONL non può saltare: va letto dall'inizio fino alla riga, quindi il rapporto dipende da N e non lo riportiamo.

5.3 Dimensione

Per confrontare la dimensione con i compressori di uso generale, abbiamo riscritto gli stessi 100.000 record: un generatore in `awk`, pubblicato con questo testo, produce il JSONL; `wjsw de-texto` scrive il `.jsw`; e il JSONL restituito da `wjsw para-texto --json` ha 519.264.563 byte, la stessa dimensione, alla scala dei MB, che `wjsw medir` riporta per il proprio JSONL (519,3 MB); il `.jsw` ha 79.076.300 byte (79,1 MB in entrambi). Il JSONL del generatore ha 200.000 byte in più, perché lì il valore decimale è tra virgolette; la Tabella 5 usa il JSONL riscritto da jsonW, e con il file del generatore i rapporti cambiano alla prima cifra decimale (gzip: 19,7× invece di 19,8×). Il JSONL è stato compresso con gzip 1.14 [32], zstd 1.5.7 [31] e xz 5.8.3 [33] ai loro livelli massimi abituali.

Tabella 5. Dimensione dei 100.000 record, in MB (10^6 byte), con e senza la colonna del vettore. Rapporto = JSONL non compresso ÷ la dimensione di quella riga.

File	Con vettore	Rapporto	Senza vettore	Rapporto
JSONL	519,3	1,0×	38,2	1,0×
`.jsw`, vettore `vf32`	155,9	3,3×	n/d	n/d
`.jsw`, vettore `vf16`, livello predefinito	79,1	6,6×	2,22	17,2×
`.jsw`, vettore `vq8` (con perdita)	41,1	12,6×	n/d	n/d
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`, vettore `vf16`, `--max`	1,36	381,1×	0,0175	2.185×

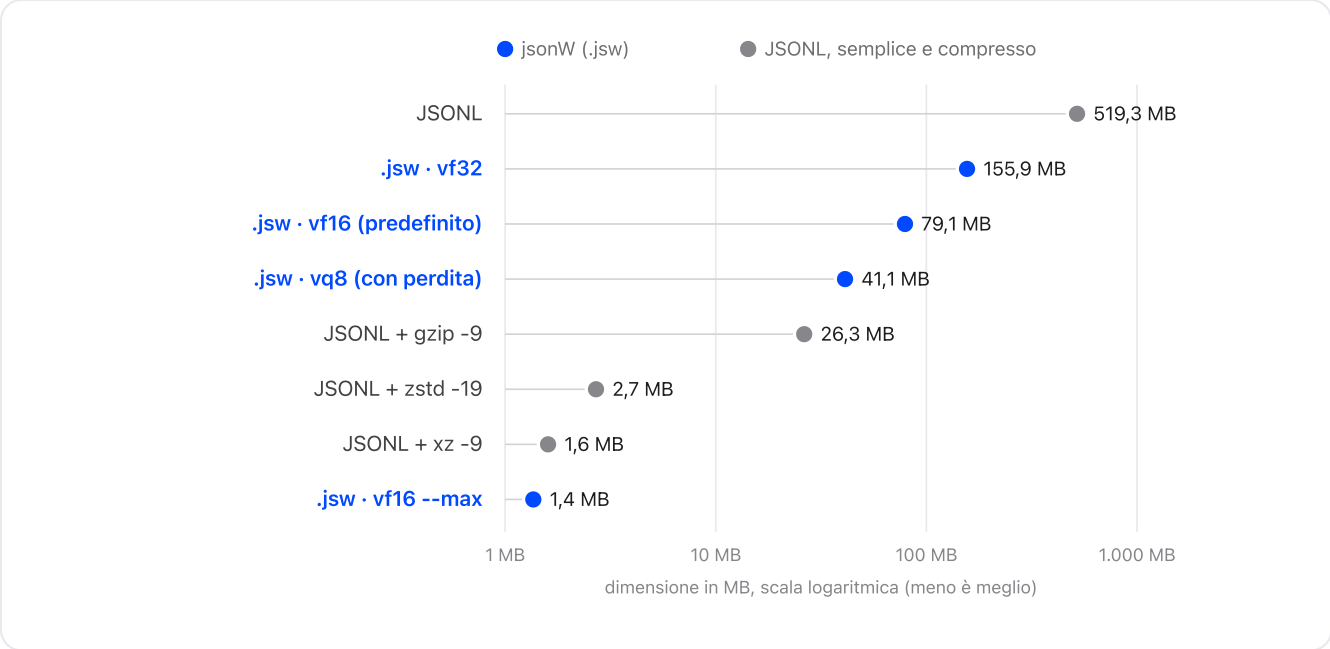


Figura 5. Le dimensioni con vettore della Tabella 5, in scala logaritmica.

La Figura 5 mostra le dimensioni con il vettore. Rispetto al JSONL non compresso, che è come il JSONL viene di solito conservato e scambiato, il `.jsw` standard è 6,6 volte più piccolo, e quasi tutta la sua dimensione è il vettore: 384 componenti da 2 byte in 100.000 righe fanno 76,8 MB. Rispetto al JSONL compresso con zstd o xz, il `.jsw` standard è più grande. Al livello massimo, la 1.3.11 è risultata più piccola di entrambi: 1,36 MB con il vettore, contro 2,67 MB di zstd e 1,56 MB di xz, e 17,5 kB senza vettore, contro 0,63 e 0,47 MB; nella 1.2.1, lo stesso livello dava 3,75 e 2,19 MB. Il guadagno viene da due nuovi codec del livello massimo, che scrivono ogni cella e ogni vettore come differenza rispetto alla riga precedente (il prefisso e il suffisso comuni e la parte centrale che cambia), ed è grande qui perché i dati del test sono molto ripetitivi: i testi si ripetono con un numero diverso e il vettore segue uno schema aritmetico. Il ritorno è senza perdita: il `para-texto` del file al livello massimo è identico, byte per byte, a quello del file al livello standard (stesso SHA-256, con e senza vettore). Sui dati reali il guadagno è molto minore: sulla base dell'INPI, il livello massimo della 1.3.11 è risultato più piccolo del 16,5% rispetto a quello della 1.2.1 (Tabella 5C). Anche i vettori dei modelli reali tendono a comprimersi poco; qui non l'abbiamo misurato, ma se è così, la dimensione del `.jsw` con vettore resta vicina alla dimensione grezza del vettore.

Ciò che un file compresso di uso generale non dà è l'accesso. Per leggere un campo di un JSONL compresso con zstd bisogna decomprimere tutto (da 0,13 s a 0,24 s per decomprimere `com-volta.jsonl.zst`, in 3 esecuzioni sulla stessa macchina, fuori dall'ambiente della misura) e poi interpretare il testo, mentre il `.jsw` legge il campo direttamente, salta blocchi grazie alle zone, verifica la custodia e si può leggere a distanza a pezzi.

5.4 Dati reali: CSV, XML, XLSX e Parquet

Le misure precedenti usano dati generati. Per confrontare con i formati di foglio di calcolo e di scambio più comuni, abbiamo usato dati pubblici: il primo milione di righe del file `MARCAS_DADOS_BIBLIOGRAFICOS.csv`, dai dati aperti sui marchi dell'INPI, l'ufficio brasiliano di brevetti e marchi (13 colonne: numero della domanda, date, presentazione, natura, nome del marchio, stato). Un programma in C, senza librerie, ha letto il CSV seguendo la RFC 4180 (virgolette, virgole e a capo dentro il campo) e ha scritto le stesse righe in quattro forme: il CSV riscritto; un XML con un elemento per riga e un figlio per colonna; un XLSX montato dal programma stesso (Office Open XML, celle di testo in linea, compresso con `zip -6`); e il

`.jsw`, con lo schema scritto a mano (date come testo; presentazione, natura e stato come dizionario). Un campo vuoto è assente in XML e in jsonW. XLSX accetta al massimo 1.048.576 righe per foglio, da cui il milione.

L'interrogazione è stata la stessa in tutti e quattro: contare i registri in vigore (`codigo_situacao = IPAS023`). Le quattro forme hanno dato lo stesso risultato, 352.248 registri, e `wjsw conferir` ha verificato lo SHA-256 e il CRC32C di tutti i blocchi del `.jsw`. Ogni interrogazione è stata eseguita 5 volte con il file già in cache; vale la minore. Macchina: server a 8 core con Linux 7.3, jsonW 1.2.0, 13 novembre 2026.

Tabella 5B. Un milione di righe reali dell'INPI in quattro formati. Dimensione in MB (10^6 byte); interrogazione = contare i registri in vigore, in millisecondi.

Formato	Dimensione	Con zstd -3	Interrogazione	Come è stata fatta l'interrogazione
XML	502,1	46,0	159	`grep -c` sul file intero
CSV	141,1	36,1	97	`grep -c` sul file intero
XLSX	74,3	n/d	9.364	`unzip -p` del foglio + `grep`
`.jsw`	51,2	n/d	119	`wjsw calcular --onde`, solo la colonna

In dimensione, il `.jsw` è stato 2,8 volte più piccolo del CSV, 1,5 volte più piccolo dell'XLSX e 9,8 volte più piccolo dell'XML. CSV e XML compressi con zstd sono più piccoli del `.jsw` (36,1 e 46,0 MB contro 51,2 MB), ma, come nella Sezione 5.3, vanno decompressi per intero prima di qualsiasi lettura. Nell'interrogazione, il `.jsw` è stato 79 volte più veloce dell'XLSX, che va decompresso a ogni lettura. Rispetto a CSV e XML, `grep` è stato più veloce o simile: cerca solo un pezzo di testo nel file intero, senza separare colonne né tipi, e darebbe un risultato sbagliato se lo stesso codice comparisse in un'altra colonna. In questo insieme non compare, e i conteggi coincidono. Un'interrogazione con la stessa garanzia sul CSV richiede di leggere e separare ogni riga, cosa che `awk` ha fatto in 1.597 ms nel primo giro di questa misura, senza gestire le virgolette.

Ciò che nessuno dei tre formati porta da sé: schema con tipi (data, decimale, denaro, vettore), lettura di una colonna senza leggere le altre, ricerca per termini e per significato, vista per permesso, cronologia delle righe cancellate e custodia per blocco. L'XLSX ha tipi di cella e più fogli, ma limita la dimensione del foglio e va decompresso per essere letto.

La base intera. La Tabella 5B usa un milione di righe di un file. Per vedere il formato con tutto, abbiamo convertito i sette file dei dati aperti sui marchi dell'INPI [38], scaricati il 12 novembre 2026, senza tagliare nulla: 55.278.212 righe e 12,53 GB di CSV, con tutte le colonne e la cronologia completa dei provvedimenti di ogni domanda (24.978.976 righe solo in `MARCAS_DESPACHOS.csv`). Ogni file ha ricevuto uno schema scritto a mano. Numeri, date e codici hanno mantenuto il proprio tipo; il testo, i byte così come sono arrivati. Il convertitore, in C, verifica ogni valore tipizzato sul momento: il numero e la data devono tornare allo stesso testo del CSV, altrimenti la conversione si ferma. Questo ha trovato due casi reali nella fonte, conservati così come sono arrivati: una data di priorità nell'anno 1018 (un errore di battitura nell'originale) e una riga della classificazione di Nizza senza numero di domanda, sulle 5.333.893 di quel file.

Il Parquet [3] è stato scritto da DuckDB 2.0.0 [39] con gli stessi tipi e compressione zstd. L'XML segue l'impostazione della Tabella 5B. L'XLSX è rimasto fuori: sei dei sette file superano il limite di 1.048.576 righe per foglio, e quello dei provvedimenti ha quasi 24 volte quel limite. Le otto forme hanno dato gli stessi conteggi nelle tre interrogazioni, con un'eccezione, spiegata più sotto. `wjsw conferir` ha verificato lo SHA-256 e il CRC32C di tutti i blocchi dei sette `.jsw`. Ogni interrogazione è stata eseguita 5 volte, con il

file già in cache, e vale la minore. La scrittura del jsonW ha usato un core. DuckDB, zstd e xz hanno usato gli otto core, e anche `wjsw calcular`. Macchina: la stessa della Tabella 5B, 31 GB di memoria, 13 novembre 2026.

Tabella 5C. La base intera dei marchi dell'INPI (sette file, 55,3 milioni di righe) in ogni formato. Dimensione in MB (10^6 byte), sommata sui sette file; scrivere = tempo per produrre il formato a partire dal CSV, in secondi.

Formato	Dimensione	Volte più piccolo del CSV	Scrivere	Legge una colonna senza aprire il resto?
XML	24.804,7	0,5	n/d	no
CSV (originale)	12.527,4	1,0	n/d	no
`.jsw` , 1.2.1	2.804,8	4,5	176,7	sì
`.jsw` , 1.3.11	2.579,2	4,9	n/d	sì
`.jsw` , 1.2.1, `--max`	2.321,8	5,4	184,9	sì
XML + zstd -3	2.210,0	5,7	21,4	no, decomprime tutto
`.jsw` , 1.3.11, `--max`	1.938,1	6,5	n/d	sì
CSV + zstd -3	1.908,9	6,6	13,2	no, decomprime tutto
Parquet (zstd)	1.665,8	7,5	16,6	sì
CSV + xz -6	1.161,0	10,8	544,5	no, decomprime tutto

Tabella 5D. Tre interrogazioni per colonna sulla base intera, in millisecondi (la minore di 5), misurate per la quinta volta il 13/11/2026 alle 06:21 UTC, con il `wjsw` 1.3.11 pubblicato e l'1.2.1, alla stessa ora. In vigore: contare `codigo_situacao = IPAS023` in 6,7 milioni di domande. Cronologia: tutti gli atti di una domanda (`numero_inpi = 818602856`) in 25,0 milioni. Classe 35: contare `classe_nice = 35` in 5,3 milioni. Tra parentesi, DuckDB con un core. L'XML viene dal primo giro, perché il file è stato cancellato dopo (occupava 24,8 GB).

Formato e lettore	In vigore	Cronologia	Classe 35
Risposta corretta	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 che legge il CSV	375 (1.447)	1.205 (4.523)	592 (1.979)
CSV, `grep -c`	519	6.690	1.278 (sbagliata: 1.076.929)
CSV + zstd, `zstd -dc | grep -c`	798	3.129	2.786 (sbagliata)
CSV + xz, `xz -dc | grep -c`	1.050	3.025	2.105 (sbagliata)
CSV + xz, `xz -dc | grep -c`	971	2.628	1.950 (sbagliata)
XML, `grep -c` (primo giro)	3.017	9.925	3.430

Cosa mostra la base intera. Per dimensione, il `.jsw` 1.3.11 è risultato 4,9 volte più piccolo del CSV al livello predefinito e 6,5 volte al massimo, e 12,8 volte più piccolo dell'XML; l'1.2.1 era 4,5 e 5,4 volte più piccolo del CSV, e l'1.3.11 al massimo è risultato il 16,5% più piccolo dell'1.2.1. Nelle query, con otto core da entrambe le parti, l'1.3.11 ha contato i record in vigore in 43 ms al livello massimo e 44 ms al predefinito, davanti ai 54 ms di Parquet letto da DuckDB; con un core, DuckDB su Parquet ha impiegato 76 ms per quel conteggio e 521 ms per lo storico. In tutte e tre le query, il `.jsw` è stato più veloce di DuckDB che legge il CSV stesso, di `grep`, del CSV compresso e dell'XML. Nello storico di una pratica e nella classe 35, il `.jsw` ha impiegato 207 e 142 ms, e Parquet con otto core 129 e 41 ms: è il prossimo passo di prestazioni della Sezione 8. Per dimensione, Parquet (1.665,8 MB) e il CSV compresso con xz (1.161,0 MB) occupano meno del `.jsw` al massimo (1.938,1 MB), e il CSV compresso va decompresso per intero prima di qualsiasi lettura; con zstd, il CSV è risultato praticamente uguale al `.jsw` al massimo (1.908,9 MB). La scrittura del `.jsw` 1.2.1, su un core, ha richiesto 177 s; quella di Parquet, con otto, 17 s. I file 1.3.11 sono stati riscritti a partire dal `.jsw` 1.2.1 (`wjsw regravar`), per questo la Tabella 5C non riporta il loro tempo di scrittura.

I giri e le correzioni. La Tabella 5D è stata misurata cinque volte, sempre con il Parquet e il CSV alla stessa ora. Con il `wjsw` 1.2.0, il `.jsw` ha impiegato 677, 1.494 e 1.114 ms nelle tre interrogazioni, da 10 a 31 volte il tempo del Parquet: il filtro leggeva tutte le colonne di ogni riga. Corretto per leggere solo la colonna richiesta, ha impiegato 82, 311 e 209 ms. Nel terzo giro, la valutazione è passata a farsi per blocco, confrontando il codice del dizionario e gli interi direttamente nella colonna e costruendo la riga solo quando passa: 48, 344 e 186 ms. Nel quarto, con l'1.3.11 ancora in preparazione, la cronologia è scesa da 355 a 279 ms. Nel quinto, con il binario della versione pubblicata, che confronta il dizionario a lotti con istruzioni vettoriali e valuta il «contiene» per blocco, l'1.3.11 ha impiegato 44, 254 e 152 ms al livello predefinito e 43, 207 e 142 ms al massimo, contro 54, 353 e 198 ms dell'1.2.1 alla stessa ora. La cronologia cerca il numero di domanda, un codice fatto solo di cifre che l'1.3.11 scrive a larghezza fissa (in DADOS_BIBLIOGRAFICOS, la sola colonna è scesa da 53,0 a 25,7 MB). La macchina non era isolata, e lo stesso binario è variato fino al 23% da un giro all'altro (il Parquet, da 41 a 54 ms nel conteggio in vigore); per questo il confronto vale all'interno dello stesso giro. Un file scritto con i nuovi codec non si apre nei lettori 1.2.x. I tempi di `grep` sul CSV sono variati ancora di più (da 2.102 a 6.690 ms sulla cronologia), perché i 12,5 GB di CSV non stanno tutti nella cache accanto agli altri file, e nel quinto giro le riscritture appena prima hanno tolto il CSV dalla cache; vale il numero dello stesso giro.

L'eccezione nei conteggi è quella della classe 35. `grep` sul CSV, semplice o compresso, ha contato 1.076.929, uno in più della risposta corretta: il pezzo `,35,` compare anche in un'altra colonna di una riga. È il rischio descritto accanto alla Tabella 5B, che in quell'insieme non si era verificato e qui sì. Tutti i lettori che separano le colonne (jsonW, Parquet, DuckDB che legge il CSV e l'XML con il nome della colonna nel markup) hanno dato 1.076.928.

Di questa conversione 1:1, l'unica funzione oltre al tipo e alla colonna che è entrata è stata la custodia: ogni blocco dei sette file ha CRC32C e SHA-256, verificati sopra. La conversione non ha marcato permessi né montato ricerca, per restare uguale al CSV. Il formato permette, nello stesso file, di nascondere il CPF e il CNPJ dei titolari a chi non ha l'ambito (Sezione 4.3), di leggerne a distanza solo una parte e di cercare per termini e per significato (Sezione 3). Il Parquet ha una somma di controllo opzionale per pagina [36] e la crittografia per colonna [5], ma non la catena di hash del file intero né la vista per ambito; CSV e XML non portano nulla di tutto questo.

5.5 Limiti della misura

- **Il lettore JSON è il nostro.** Il lato JSON è stato scritto e letto dal codice di testo di jsonW stesso, che valida e converte i tipi; legge il JSONL a circa 140 MB/s. I lettori JSON più avanzati, come simdjson [23],

superano 1 GB/s. Con uno di loro, le righe "scrivere" e "leggere tutto" della Tabella 4 si ridurrebbero parecchio. Come stima, non come misura: a 2 GB/s, leggere i 519 MB richiederebbe circa 260 ms, e il rapporto di "leggere una colonna" sarebbe nelle centinaia, non nelle migliaia.

- **Dati sintetici e ripetitivi,** come detto nella Sezione 5.3. Il rapporto di dimensione dipende molto dai dati, e ancora di più al livello massimo, che conserva la differenza tra righe consecutive: misurate con i vostri.
- **Una macchina, un sistema, memoria invece di disco.** I tempi assoluti cambiano con il processore, l'archiviazione e il carico; nelle 10 esecuzioni, il rapporto maggiore di ogni operazione è stato dal 19% al 75% sopra il minore; quello di leggere una colonna è il più variabile, perché il tempo di jsonW lì è di 1,6 ms.
- **Parquet misurato solo con dati dell'INPI e un lettore.** Il confronto di prestazioni con Parquet è nelle Tabelle 5C e 5D, con DuckDB 2.0.0 come scrittore e lettore. Un altro lettore, un'altra compressione o un'altra dimensione del gruppo di righe cambiano i numeri. Arrow, MessagePack, CBOR e Protobuf restano senza misura di prestazioni in questa versione del testo; la Sezione 6 confronta funzioni, non velocità.

5.6 Numeri pubblicati in precedenza

Fino al 12 novembre 2026, il sito e il repository di jsonW citavano "4,9 volte più piccolo del JSONL" e "3.281 volte più veloce a leggere un campo", dalla prima misura, del 6 novembre, con la versione 1.0 su un'altra macchina. Il testo 1.0 di questo articolo, del 12 novembre, ha rifatto la misura con la 1.2.1: 6,6 volte più piccolo e circa 2.500 volte più veloce a leggere una colonna, con il livello massimo a 3,75 MB con vettore e 2,19 MB senza. Rifatta per questo testo con la 1.3.11, sulla stessa macchina, dà 6,6 volte più piccolo e circa 2.300 volte più veloce a leggere una colonna (la differenza è entro la variazione tra esecuzioni), e il livello massimo scende a 1,36 MB e 17,5 kB. Valgono i numeri di questo testo, con la data, la macchina e la versione accanto.

5.7 Come riprodurre

```
git clone https://github.com/developer-wortic/jsonw && cd jsonw
export JSONW_SEM_PING=1                      # senza l'avviso di installazione
W=bin/linux-x86_64/wjsw                       # o bin/macos-arm64/wjsw
for i in $(seq 1 10); do $W medir /tmp 100000; done

# Tabella 5: lo stesso insieme in JSONL, il .jsw e i compressori
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*
```

Listato 9. Le misure di questo testo sulla vostra macchina. `VET=0` e lo schema senza la colonna `vetor` danno la colonna "senza vettore".

6 Confronto con altri formati

La Tabella 6 confronta ciò che ogni formato fa da sé, senza librerie sopra. I formati a colonne maturi, Parquet e Arrow, hanno già buona parte del vantaggio di accesso misurato nella Sezione 5, e varie cose che jsonW non ha ancora.

Tabella 6. Ciò che ogni formato porta nel formato stesso. Fonti nei riferimenti indicati in ogni colonna.

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Forma	testo; una riga per record in JSONL	binaria, senza schema	binaria, schema compilato (`.proto`)	file a colonne	colonne in memoria; file e flusso IPC	riga binaria, file a colonne e testo
Schema	no (JSON Schema a parte)	no	sì, fuori dal messaggio	sì, nel file	sì	sì, una riga, nel file
Leggere un campo senza il resto	no	no	no	sì, per colonna	sì, accesso diretto	sì, per riga e campo
Decimale esatto	no	CBOR: sì (frazione decimale, tag 4)	non nativo	sì (DECIMAL)	sì (decimal128 e 256) [37]	sì
Denaro e misura	no	no	denaro con un messaggio comune (`google.type.Money`) [10]	no	no	sì, con valuta e unità
Vettore denso	lista di numeri in testo	lista; CBOR ha array tipizzati [34]	campo ripetuto	lista o byte di dimensione fissa	lista di dimensione fissa	f32, f16, 8 e 4 bit, bit; tensore
Permessi	no	no	no	crittografia delle colonne con chiavi diverse [5]	no	vista per colonna, riga e cella, applicata da chi serve; senza crittografia
Integrità	no	no	no	CRC opzionale per pagina [36]; crittografia autenticata	non nel formato	crc32c e hash per blocco, concatenati
Ricerca e ranking nel formato	no	no	no	statistiche per gruppo di righe e pagina per saltare dati	no	zone, coseno, sparso, griglia, top-k, lessico
Lettura remota parziale	no	no	no	sì (piè di pagina e Range)	sì, nel formato di file	sì
Specifica	aperta (RFC)	aperta (CBOR: RFC)	aperta	aperta (Apache)	aperta (Apache)	uso, testo e ABI pubblici; disposizione binaria interna

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Implementazioni	molte, indipendenti	molte, indipendenti	diverse	diverse, indipendenti	diverse, indipendenti	una (nucleo in C), con SDK in 10 linguaggi

Dove si colloca ogni formato. JSON è universale e leggibile ovunque. MessagePack, CBOR e Protobuf sono maturi e implementati in decine di linguaggi, e Protobuf ha regole mature di evoluzione dello schema. Parquet e Arrow sono standard aperti, letti da quasi ogni motore di analisi, e Parquet ha la cifratura per colonna per il file a riposo. jsonW lavora con tutti: ogni JSON è già jsonW-testo, `wcon` porta e riporta tabelle dei database con i tipi giusti, e il file a riposo può essere cifrato dal disco o dallo storage.

Cosa riunisce jsonW. La differenza di jsonW non è una funzione isolata, ma avere, nello stesso modello, la forma di messaggio, la forma di file a colonne e una forma di testo compatibile con JSON; tipi di business verificati in scrittura; permessi per colonna, riga e cella applicati in lettura; e operazioni di ranking, ricerca e classificazione dentro il file. Il prezzo di questa scelta è essere, per ora, un'unica implementazione con il nucleo chiuso (Sezione 8).

7 Usi

7.1 Modelli di IA e ricerca semantica

Una base di conoscenza sta in un `.jsw` con il testo, i metadati e il vettore di ogni brano; la ricerca semantica è un `avaliar … cosseno` che legge solo la colonna del vettore, e il lessico e i vettori sparsi coprono la ricerca per termini. Con i primi k e il lessico, lo stesso file fa da instradatore che sceglie lo specialista: il lessico classifica il testo della richiesta e il ranking sceglie.

7.2 Dati delle imprese

I registri come quelli delle imprese (CNPJ) e dei codici postali (CEP) del Brasile sono dove JSON perde di più: lo zero iniziale sparisce quando il codice diventa numero, il capitale sociale diventa un double e la data perde il fuso orario. In jsonW, ogni cosa ha il suo tipo:

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

Listato 10. Uno schema di registro delle imprese: codici esatti byte per byte, testo ripetuto in un dizionario, denaro con valuta e un tenant per separare i clienti.

7.3 Database

`wcon`, il connettore di database del pacchetto, porta dati da PostgreSQL, MySQL e MariaDB, SQLite, Redis e Valkey e MongoDB a un `.jsw` e ritorno, con lo schema preso dal tipo di ogni colonna: `numeric` diventa decimale esatto, `bigint` resta intero a 64 bit e il `vector` e l'`halfvec` di pgvector [30] diventano `vf32` e `vf16`. I valori viaggiano sempre come dati, con la lunghezza davanti, mai incollati a un comando; il ritorno avviene in una sola transazione. L'andata e ritorno è stata verificata con PostgreSQL 14, 16 e 18 (stesso md5 dai due lati), MariaDB 11.8 (hex di ogni colonna), SQLite, Redis 7 e MongoDB 7, con carico ostile: injection, virgolette, `\0`, accenti, emoji e testo da destra a sinistra [20].

7.4 Server web

Con `wjsw-cgi` (Sezione 4.6), un sito serve un `.jsw` come JSON, NDJSON o il file stesso, con filtro, scelta dei campi e limite nell'indirizzo, sempre con la vista della sessione.

7.5 Archiviazione su R2 e S3

Un `.jsw` può stare in un bucket di Cloudflare R2 o in qualsiasi S3 ed essere interrogato lì, senza scaricare il file intero (Figura 4). Poiché R2 non fa pagare l'uscita dei dati [26], interrogare un pezzo di un file grande costa il pezzo letto più le operazioni, che R2 fa pagare per richiesta (il Listato 6 fa 5 richieste).

8 Limiti e prossimi passi

- **Un'implementazione, nucleo chiuso.** La disposizione binaria non è pubblicata; leggere e scrivere `.jsw` dipende da `libjsonw`. L'ABI stabile e gli SDK riducono il rischio, ma non sostituiscono una specifica aperta con implementazioni indipendenti.
- **Nessun audit indipendente.** Le prove citate (fuzzing, carichi ostili, confronto byte per byte tra SDK) sono nostre.
- **Piattaforme.** Ci sono binari per macOS arm64, Linux x86_64 (glibc 2.34 o più recente) e Windows x86_64. Non ce ne sono ancora per macOS Intel, Linux arm64 né Linux con musl. Su Windows, `wcon` e `wremoto` funzionano tramite WSL.
- **Funzioni senza comando pubblico.** L'indice per chiave, la scrittura dal vivo, la vista "solo ciò che vale" e la compattazione esistono nel nucleo, ma non hanno comando in `wjsw` né funzione nell'ABI 1.3.11; il formato della tabella della griglia non è documentato.
- **Il protocollo non è nel pacchetto.** Il protocollo post-quantistico non è incluso nel pacchetto pubblico 1.3.11.
- **Parole in portoghese.** I tipi e i comandi usano parole portoghesi (`texto`, `dinheiro`, `medida`, `ver`, `conferir`), il che richiede un glossario per chi non legge il portoghese.
- **Database.** MySQL 8 funziona solo con `mysql_native_password`; la prima autenticazione completa di `caching_sha2_password` non esiste ancora.
- **Misure.** I limiti della Sezione 5.5.
- **Interrogazione per colonna: il prossimo passo.** Sulla base intera dell'INPI (Tabella 5D), `wjsw calcular` 1.3.11 conta i record in vigore in 43 ms, davanti a DuckDB che legge Parquet, e dalla prima tornata (1.2.0) quel conteggio è sceso da 677 a 43 ms. Il prossimo passo è portare la stessa lettura per blocco e a lotti allo storico di una pratica e alla classe 35, dove il `.jsw` ha impiegato 207 e 142 ms e DuckDB con otto core 129 e 41 ms. Per dimensione, i prossimi codec (differenza e impacchettamento di bit per interi e date) mirano ai 1.938,1 MB del livello massimo.

Prossimi passi di questo testo e del pacchetto: binari per macOS Intel, Linux arm64 e musl; un'interrogazione che verifichi solo il CRC32C di ogni blocco e lasci lo SHA-256 al `conferir` (il Parquet non verifica alcun hash in lettura, e parte della differenza che resta viene da lì); il codec di dizionario con finestra, per colonne come la classe di Nizza; la codifica per differenze con impacchettamento di bit per interi e date, misurata di nuovo rispetto alle Tabelle 5C e 5D; e il confronto delle prestazioni con Arrow sulla stessa macchina.

9 Disponibilità e licenza

Lo strumento, la libreria, l'ABI, gli SDK, le configurazioni per server web e la documentazione sono su github.com/developer-wortic/jsonw [20], e l'SDK per Node.js è su npm come `@worticTecnologia/jsonw`. L'uso è regolato dalla Licenza di Uso Sostenibile di jsonW (in portoghese), che prevale su questo riassunto: jsonW è di uso libero su tutti i server e le applicazioni necessari, prodotti commerciali compresi, purché sia un componente e non il servizio venduto; senza un accordo scritto con Wortic, nessuno può vendere, ospitare od offrire a terzi jsonW o un servizio la cui funzione principale sia quella di jsonW (conservare, convertire, servire o trasportare dati in jsonW), a pagamento o gratis; le librerie e gli strumenti distribuiti come binari non possono essere decompilati, salvo quanto permesso dalla legge; e la disposizione interna del file, i codec, il protocollo e il nucleo appartengono a Wortic, quindi anche un'implementazione indipendente di essi richiede un accordo scritto. La documentazione pubblica si può leggere, citare e usare liberamente per costruire programmi che usano jsonW.

Per trasparenza: la prima volta che viene eseguito su ogni macchina, lo strumento `wjsw` invia solo la versione e la piattaforma (per esempio `v=1.3.11&so=linux-x86_64`) per contare le installazioni, e non lo ripete in seguito. L'IP arriva al server con la connessione ma non viene conservato, né alcun nome, percorso o dato: il server registra solo il giorno, la piattaforma e la versione. `JSONW_SEM_PING=1` o il file `~/.jsonw/sem-ping` disattivano l'avviso prima della prima esecuzione. La libreria e gli SDK non accedono mai alla rete.

Riferimenti

[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)*, che definisce il 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: repository pubblico, documentazione (docs/JSONW.md), ABI (sdk/jsonw.h) e binari*. 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*, sezione 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` di `PageHeader`). github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift

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

[38] Instituto Nacional da Propriedade Industrial (INPI, Brasile). *Dati aperti sui marchi* (`marcas.zip`, sette file CSV), scaricati il 12/11/2026. dadosabertos.inpi.gov.br/download/marcas/marcas.zip

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

Come citare

Wortic Tecnologia. *jsonW: un modello di dati, tre forme senza perdita e i permessi dentro il formato*. Testo 1.1, su jsonW 1.3.11. 13 novembre 2026. Disponibile su: https://jsonw.org/it/paper.

```
@techreport{jsonw2026it,
  title       = {jsonW: un modello di dati, tre forme senza perdita
                 e i permessi dentro il formato},
  author      = {{Wortic Tecnologia}},
  institution = {Wortic Tecnologia},
  year        = {2026},
  month       = nov,
  note        = {Testo 1.1, su jsonW 1.3.11},
  url         = {https://jsonw.org/it/paper}
}
```

Testo 1.1, pubblicato il 13/11/2026, su jsonW 1.3.11 (il testo 1.0, del 12/11/2026, trattava l'1.2.1). jsonW è json Wondrous; il nucleo in C è mantenuto da Wortic Tecnologia. Correzioni e misure di terzi sono benvenute tramite il repository pubblico. La versione originale è quella portoghese; questa traduzione mantiene gli stessi numeri e termini.