Article technique · jsonW 1.3.11

jsonW : un modèle de données, trois formes sans perte et les permissions dans le format

Wortic Tecnologia · 13 novembre 2026 · texte 1.1, sur jsonW 1.3.11

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

SOMMAIRE

RÉSUMÉ

JSON et JSONL sont devenus la langue commune des données, mais ils ont été conçus pour échanger des messages entre programmes, pas pour stocker et interroger le volume et la variété de données qu'utilisent les systèmes d'IA. Lire un champ d'un fichier JSONL oblige à analyser le fichier depuis le début ; l'argent, les mesures et les décimaux exacts n'ont pas de type ; les vecteurs deviennent du texte ; les permissions et la preuve d'intégrité restent hors de la donnée. Cet article décrit jsonW (json Wondrous) 1.3.11, un modèle de données à trois formes qui se convertissent sans perte : ligne, pour les messages et le temps réel ; colonne, le fichier `.jsw` en blocs lu par `mmap` ; et texte, dans lequel tout JSON valide est aussi un jsonW-texte valide. Le format apporte des types métier, des vecteurs et des tenseurs, dix-neuf fonctions intégrées (recherche, classement, maintenance, graphie proche et signaux audio, texte et vidéo), des vues par permission et une garde par hachage. Sur 100 000 enregistrements synthétiques, sur une machine, le fichier `.jsw` a occupé 6,6 fois moins que le JSONL non compressé et a lu une colonne environ 2 300 fois plus vite que notre propre lecteur JSON. Sur la base réelle des marques de l'INPI, 55,3 millions de lignes et 12,53 Go de CSV, le `.jsw` est 6,5 fois plus petit que le CSV et 12,8 fois plus petit que le XML, et il a compté les 2,3 millions d'enregistrements en vigueur en 43 ms, plus vite que Parquet lu par DuckDB.

Mots-clés : format de données ; stockage en colonnes ; JSON ; recherche vectorielle ; contrôle d'accès ; intégrité ; échange de clés post-quantique.

1 Introduction

JSON [1] est né pour échanger des messages entre programmes dans un texte que les gens peuvent lire. JSON Lines [2] a porté l'idée aux fichiers : un enregistrement par ligne. Les deux sont devenus des standards de fait pour les journaux, les exports, les jeux d'entraînement et les réponses d'API. Ils sont simples, universels et lisibles, et ces qualités expliquent leur succès. Travailler avec des données pour l'IA demande pourtant des choses qu'ils n'ont pas été faits pour donner.

1.1 Ce qui manque à JSON et JSONL

- **Taille.** Tout est texte. Un vecteur de 384 dimensions devient des milliers de caractères par enregistrement : les 100 000 enregistrements du test de la Section 5 occupent 519,3 Mo en JSONL.
- **Accès.** Il n'y a pas d'index. Pour lire un champ d'un enregistrement, il faut analyser le fichier depuis le début jusqu'à lui ; pour lire une colonne, le fichier entier.
- **Types.** La spécification ne fixe pas la précision des nombres : la RFC 8259 laisse la limite à chaque implémentation et conseille de ne compter que sur la précision d'un double IEEE 754 [1, 28]. Il n'y a pas de type pour l'argent avec sa devise, la mesure avec son unité, le décimal exact, le code qui ne doit pas perdre son zéro initial, la date, le vecteur ni le tenseur. Chaque système invente une convention, et passer d'une convention à l'autre perd des données.
- **Permissions.** Qui peut voir chaque colonne et chaque ligne vit hors de la donnée, dans une couche à part, et chaque requête doit penser à l'appliquer.
- **Garde.** Rien dans le fichier ne prouve qu'il n'a pas été modifié.

En pratique, la même donnée passe par une chaîne : JSON sur le réseau, CSV ou Parquet pour le stockage, une base vectorielle pour la recherche sémantique et une couche de permissions pour décider qui voit quoi. Chaque étape coûte du temps et de la mémoire et, parfois, la donnée elle-même : le prix qui devient du texte, l'identifiant fiscal qui perd son zéro initial, le poids dont personne ne sait s'il était en grammes ou en kilos.

1.2 Contributions

Cet article décrit jsonW (json Wondrous) version 1.3.11. Ses contributions sont :

1. un modèle de données à trois formes qui se convertissent sans perte (ligne, colonne et texte), dans lequel tout JSON valide est aussi un jsonW-texte valide (Section 2) ;
2. des types métier et d'IA déclarés dans le schéma lui-même (Section 2.3) ;
3. dix-neuf fonctions dans le fichier, de la recherche et du classement aux signaux audio, texte et vidéo, montrées avec l'outil public quand il a une commande pour elles (Section 3) ;
4. des permissions par colonne, ligne et cellule et une garde par hachage comme partie du format (Section 4) ;
5. des mesures reproductibles, avec méthodologie, données et les limites de la mesure (Section 5), et une comparaison avec JSON, JSONL, MessagePack, CBOR, Protobuf, Parquet et Arrow (Section 6).

1.3 Ce que ce texte ne décrit pas

Le noyau de jsonW est écrit en C, sans dépendance au-delà de la bibliothèque standard, et il est maintenu par Wortic Tecnologia ; il n'est pas open source. Ce qui est public, et ce que ce texte utilise, c'est la documentation d'usage, l'ABI en C (`sdk/jsonw.h`), les SDK, les binaires de l'outil et le comportement observable [20]. La disposition binaire interne, les codecs de compression et l'organisation interne d'un bloc sont hors du champ. Ce que nous affirmons sur le format et l'outil peut être vérifié avec les binaires publics ; les preuves internes (fuzzing, charges hostiles, comparaison entre SDK) sont les nôtres et signalées comme telles. Quand une fonction existe dans le format mais n'a pas encore de commande dans l'outil public, nous le disons.

2 Modèle de données

2.1 Trois formes

jsonW est un seul modèle de données avec trois manières de stocker la même valeur (Figure 1). Passer d'une forme à l'autre ne perd aucune valeur et ne change le type d'aucune.

- **Ligne :** binaire, une valeur après l'autre, pour les messages, les événements et le temps réel. Elle prend la place de JSON, MessagePack, CBOR et Protobuf. C'est la forme qui circule sur le réseau (Section 4.5).
- **Colonne :** le fichier `.jsw` , en blocs stockés par colonne et lus directement sur le disque par `mmap` , sans copie. Elle prend la place de JSONL, CSV et Parquet.
- **Texte :** le jsonW-texte, pour que les gens lisent et écrivent. Il prend la place de JSON, TOML et YAML. Tout JSON valide est déjà un jsonW-texte valide ; le texte ajoute des notations pour les types qui manquent à JSON, comme `12.50 BRL` , `250 g` et `t"2026-11-09T01:00:00.000000Z"` .

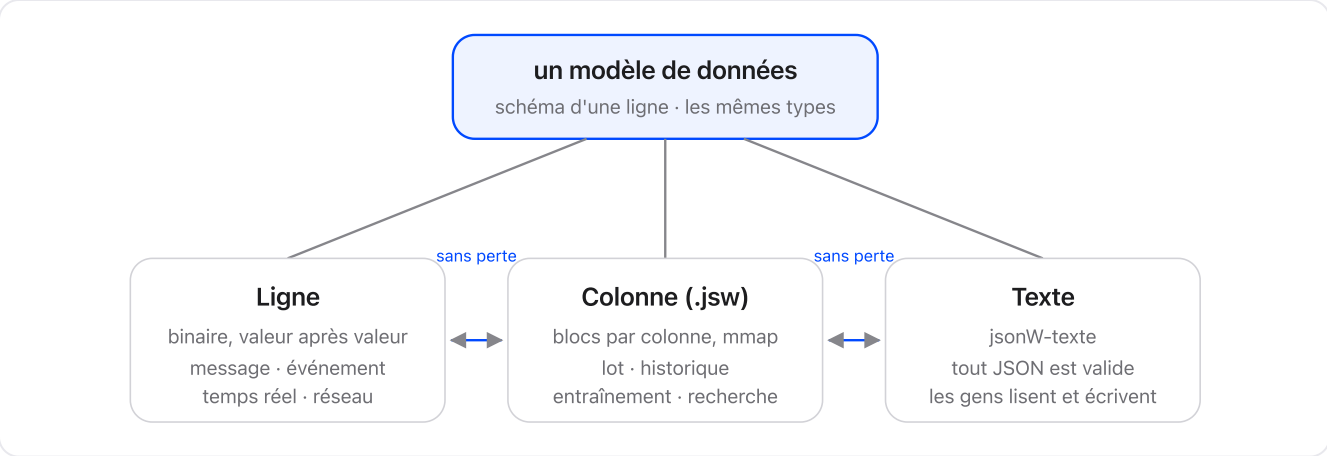


Figure 1. Un modèle, trois formes. Le schéma et les types sont les mêmes dans les trois ; convertir de l'une à l'autre ne perd aucune valeur et ne change aucun type.

2.2 Schéma

Le schéma est une ligne de texte avec des paires `nom:type` séparées par des virgules. Le point d'interrogation accepte le nul, et les marques après le type déclarent les permissions (Section 4.3). Les noms des types et des commandes sont des mots portugais : `texto` est texte, `dinheiro` est argent, `medida` est mesure, `codigo` est code, `tempo` est temps, `pessoal` est personnel.

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

Listing 1. Un schéma. `cpf` (un identifiant fiscal personnel brésilien) accepte le nul et est une donnée personnelle : sans la portée `pessoal`, il sort nul.

Le schéma évolue sans casser les lecteurs : un champ nouveau est nul dans les anciennes lignes, un nom ne change jamais de type et le lecteur résout les champs par leur nom. Ainsi un fichier ancien est lu par du code nouveau, et un fichier nouveau par du code ancien [20].

2.3 Types

Le Tableau 1 liste les types de la version 1.3.11. Les types métier sont vérifiés à l'écriture : une entrée qui ne correspond pas au type est refusée avec la raison, au lieu d'être stockée de travers.

Tableau 1. Types de jsonW 1.3.11 [20].

Type	Dans le schéma	Ce qu'il garantit
décimal exact	`decimal[2]`	ne passe jamais par la virgule flottante : 12.50 reste 12.50
argent	`dinheiro`, `dinheiro[4]`	devise dans chaque valeur, par code ISO 4217 [27] ou par symbole (`12,50 BRL`, `US$`, `€`, `₿`) ; ne fait de somme que dans une même devise
mesure	`medida`, `medida[kg]`	89 unités dans 14 grandeurs ; convertit à l'écriture et refuse une autre grandeur
code	`codigo`	octets exacts : identifiants fiscaux, clés, plaques et SKU ne sont jamais normalisés ni rognés
texte	`texto`, `texto+nfc`	Unicode complet, avec accents, emoji et `\0` ; `+nfc` normalise sur demande [29]
texte répété	`dic`	dictionnaire par fichier : chaque texte stocké une fois
temps	`tempo`	microsecondes, UTC
url	`url`	validée à l'écriture (`https://…`, `mailto:`, `tel:`, `urn:` ; jusqu'à 8 Kio)
vecteur	`vf32[n]` `vf16[n]` `bits[n]` `vq8[n]` `vq4[n]`	matrice contiguë dans le fichier, prête pour la recherche sémantique
tenseur	`tensor<bf16>[768,768]`	éléments f32, f16, bf16, u8, i8, i16, i32
média	`midia`	hachage, type, taille, durée, largeur et hauteur ; jsonW ne fixe pas le codec : le média reste dans son propre codec ouvert
et plus	`bool` `i64` `f64` `f32` `uuid` `bytes` `ref` `lista<t>` `obj{…}` `mapa<k,v>` `qualquer`	`qualquer` (n'importe lequel) stocke du JSON ouvert ; imbrication jusqu'à 1000 niveaux

Les 14 grandeurs de mesure sont la longueur, la masse, le volume, l'énergie, le temps, la surface, les données, la vitesse, la température, l'adimensionnel, la puissance, la pression, le carbone et la densité. La conversion est exacte entre multiples décimaux (12,5 mm = 0,0125 m) et arrondie à l'échelle de la colonne pour les autres (1 mi = 1609,344 m, stocké comme 1609 m dans une colonne `medida[m]`, qui n'a pas de décimales ; 25 °C = 77 °F).

2.4 Virgule ou point

Les lecteurs acceptent les deux séparateurs : `12,50`, `1.234,56 BRL` et `1,234.56 USD`. Quand les deux sont présents, le dernier est le décimal. Avec un seul, ce séparateur est le décimal, sauf une virgule suivie de trois chiffres dans une devise à point (`USD 1,234` vaut mille) ou un point suivi de trois chiffres dans une devise à virgule (`R$ 1.234` vaut mille). La sortie canonique utilise toujours le point. Et les nombres réels ne dépendent jamais des paramètres régionaux du processus : une JVM ou un PHP configurés en portugais n'échangent pas le séparateur.

2.5 Un exemple exécuté

Le Listing 2 montre le trajet de JSONL vers `.jsw` et retour, avec la sortie réelle de l'outil `wjsw` 1.2.1.

```
# deux lignes de JSONL, telles qu'elles arrivent de n'importe quel système
{"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"}

# ce qui ne correspond pas au type est refusé, avec la raison
$ wjsw de-texto "…,peso:medida[g],…" leite.jsonl leite.jsw     # peso: "2 L"
valor 1: medida de outra grandeza perto de ","sku":"LEI-0002"}
```

Listing 2. `1 kg` est devenu `1000 g` parce que la colonne est `medida[g]` ; le litre est un volume et la colonne est une masse, donc la ligne a été refusée (« mesure d'une autre grandeur » ; le message montre l'entrée près de l'erreur). L'argent garde sa devise, et le SKU est stocké octet par octet. L'outil écrit ses messages en portugais.

3 Les 19 fonctions

Les fonctions ci-dessous font partie du format et du noyau, pas d'une couche au-dessus, et l'outil public en expose la plupart (Tableau 2). Ce sont les pièces qui, dans d'autres piles, sont réparties entre un moteur de recherche, une base vectorielle, un index et une couche de permissions. Les exemples de cette section ont été exécutés avec le binaire public de `wjsw` 1.2.1 pour Linux x86_64 (le Listing 6B avec la 1.3.11), et les entrées sont sur jsonw.org/paper/exemplos ; les sorties longues ont été coupées là où apparaît `…`.

Tableau 2. Les 19 fonctions et où elles apparaissent dans l'outil et l'ABI publics 1.3.11.

#	Fonction	Ce qu'elle fait	Dans l'outil
1	Recherche par termes	vecteur creux avec poids par terme (IDF), le calcul qu'utilisent les moteurs de recherche	`avaliar … esparso`
2	Recherche sémantique	cosinus sur des vecteurs denses `vf32`, `vf16`, `vq8` et `vq4`	`avaliar … cosseno`
3	Grille multilinéaire	interpolation bilinéaire, trilinéaire et au-delà, directement depuis le fichier	`avaliar … grade`
4	Score à la demande	poids, cosinus, creux ou grille dans une requête d'une ligne	`avaliar`
5	Les k meilleurs	top-k déterministe par tas : le score le plus haut d'abord, égalités départagées par la plus petite ligne	`avaliar`, `escolher`
6	Zones	minimum et maximum par bloc et par colonne : la recherche saute des blocs entiers	`filtrar`, `info`
7	Dictionnaire	texte répété stocké une fois par fichier	type `dic`
8	Index par clé	va directement à la ligne d'une clé, sans parcourir	dans le noyau ; pas de commande propre
9	Lexique	automate Aho-Corasick lu depuis le disque ; mot entier, sans tenir compte des accents ni de la casse	`reconhecer`
10	Écriture en direct	un processus lit pendant qu'un autre écrit, sans jamais voir un demi-bloc	dans le noyau ; pas de commande propre
11	Vivant, supprimé et historique	réécrire la même clé modifie ; `_apagado: true` supprime ; le compactage ne réécrit que ce qui est en vigueur	dans le noyau ; pas de commande propre
12	Vue par permission	colonne, ligne et cellule, selon l'accréditation	`ver`, `wjsw-cgi`, SDK
13	Garde	crc32c et SHA-256 (ou BLAKE3) par bloc, chaînés	`conferir`, `reparar`
14	Lecture distante partielle	avec l'en-tête et le pied, ne télécharge que les blocs des lignes lues	`wremoto`
15	Graphie et son proches	note de 0 à 1 par trigrammes de la graphie et d'une clé phonétique, en portugais du Brésil, anglais américain, espagnol et chinois	`filtrar … ~~`
16	Signaux de l'audio	hauteur, pauses, intensité, jitter, shimmer, HNR, formants, MFCC, centroïde, débit d'articulation et respirations	ABI : `jsonw_sinal_audio`
17	Signaux du texte	intensificateurs, atténuateurs, polarité, absolus, négation, voix passive, pronoms, forme et indices d'ironie, en pt, en, es et zh	ABI : `jsonw_sinal_texto`
18	Vidéo en scènes	coupes de scène, les mesures et les signaux de l'audio de chaque passage et une mosaïque des images	ABI : `jsonw_video_*`
19	Colonne par bloc et par tenant	dictionnaire local par bloc, chiffres à largeur fixe, zone du tenant et comparaison par lots	à l'écriture ; `filtrar`, `calcular`

3.1 Recherche et classement (fonctions 1 à 5)

La commande `avaliar` (évaluer) donne un score à chaque ligne et renvoie les k meilleures. La requête peut être une somme pondérée de colonnes numériques, le cosinus entre une colonne de vecteurs et une requête, le produit entre vecteurs creux ou une consultation de grille. La même requête existe en JSON, identique dans l'outil et dans l'ABI, et tous les SDK renvoient le même résultat, octet par octet [20]. Le calcul ne lit que les colonnes de la requête, en parallèle. Le Listing 3 utilise cinq produits avec une colonne de vecteurs `vf16[4]` et une colonne de termes `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 et 5. score par poids (le prix compte contre, la note pour) et les 3 meilleurs
$ 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. recherche sémantique : cosinus avec la requête
$ 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. recherche par termes : produit entre vecteurs creux (terme 12 et terme 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 même requête en JSON, comme dans l'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. les 2 plus grandes valeurs d'une colonne
$ wjsw escolher prod.jsw nota 2
4.9     3       {id:4,produto:"Bolo de milho",…}
4.8     0       {id:1,produto:"Café especial",…}
```

Listing 3. Sortie : score, position de la ligne et la ligne en jsonW-texte, de la meilleure à la moins bonne.

Deux détails de la sortie méritent l'attention. D'abord, le vecteur stocké en `vf16` revient comme `0.7998046875` là où l'entrée disait `0.8` : la demi-précision garde environ trois chiffres décimaux. Quand l'exactitude du vecteur compte, la colonne doit être `vf32` ; `vq8` et `vq4` échangent plus de précision contre de la taille. Ensuite, la recherche par termes est un produit scalaire : les poids IDF [22] sont stockés dans le vecteur de chaque ligne, et la requête apporte les siens.

Grille (fonction 3). Une grille est une table de valeurs sur des axes (par exemple, le coût de livraison selon la distance et le poids). La requête `{"grade":{"campos":["x","y"],"arquivo":"g.jsw"}}` utilise les colonnes de la ligne comme coordonnées et renvoie, comme score, la valeur interpolée entre les points voisins de la grille : bilinéaire avec deux axes, trilinéaire avec trois, et ainsi de suite. Dans la version 1.3.11, le format de la table de la grille n'est pas encore dans la documentation publique, nous n'en montrons donc pas la sortie ici.

3.2 Organisation du fichier (fonctions 6 à 8)

Zones. Chaque bloc garde le minimum et le maximum de chaque colonne. Un filtre qu'un bloc ne peut pas satisfaire fait sauter au lecteur le bloc entier. Dans le fichier de 100 000 enregistrements de la Section 5, avec 103 blocs :

```
$ 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
```

Listing 4. Dans le fichier de 100 000 enregistrements (identifiants de 0 à 99 999), les zones sautent 102 blocs sur 103 (« pulados pela zona ») ; dans le fichier des produits du Listing 3, `info` montre le dictionnaire de la colonne `categoria` et les zones, avec les réels en double précision tels que l'outil les affiche.

Dictionnaire. Une colonne `dic` stocke chaque texte distinct une fois par fichier. Dans les 100 000 enregistrements du test, les colonnes `tipo` et `fonte` totalisent 10 textes distincts, et `wjsw info` affiche `dicionario 10 textos`.

Index par clé. Le format a un index qui va directement à la ligne d'une clé, sans parcourir le fichier. Dans l'outil public 1.3.11, il n'a ni commande ni fonction dans l'ABI ; l'usage visible des clés est `comparar` (comparer), qui apparie les lignes de deux fichiers par une colonne et compte les différences :

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

Seulement dans a : 1 · seulement dans b : 1 · modifiée : 1 · identiques : 0.

3.3 Lexique (fonction 9)

Une table de termes, avec le schéma `termo:texto,forma:dic,sinal:dic,peso:f32,idioma:dic,origem:dic`, est compilée, à l'ouverture, en un automate Aho-Corasick [21] qui trouve tous les termes d'un texte en un seul passage. La correspondance se fait par mot entier, sans tenir compte des accents ni de la casse, et renvoie le passage d'origine et la somme des poids par signal : un classifieur de texte sans entraînement, utile pour le tri et pour envoyer une demande au bon spécialiste.

```
$ 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)
```

Listing 5. Une phrase en portugais (« Je veux ANNULER la commande, ou alors dites-moi combien coûte le nouveau devis »). « CANCELAR » a trouvé « cancelar », et « orcamento » a trouvé « orçamento », parce que la correspondance ignore les accents et la casse. Les signaux sont « intention d'achat » et « risque de perte ».

3.4 Écriture et historique (fonctions 10 et 11)

Un fichier `.jsw` ne fait qu'ajouter. Cela donne deux garanties. La première est l'**écriture en direct** : un processus peut lire le fichier pendant qu'un autre l'écrit, comme un `tail -f` qui ne voit jamais un demi-bloc. La seconde est l'**historique** : réécrire la même clé est une modification, écrire la clé avec `_apagado: true` est une suppression, et le compactage ne réécrit que ce qui est en vigueur dans un nouveau fichier, en laissant l'historique dans l'original.

Dans la version 1.3.11, la lecture pendant l'écriture, la vue « seulement ce qui est en vigueur » par clé et le compactage n'ont ni commande dans `wjsw` ni fonction dans l'ABI publique. Avec l'outil public, l'historique apparaît tel quel, ligne après ligne :

```
$ 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}
```

Ce que l'ABI publique garantit bien à l'écriture, c'est la **publication atomique** : le fichier n'apparaît, entier, qu'à `fechar()` (fermer). Un écrivain qui sort sur une erreur en cours de route appelle `abandonar()` (abandonner), et la destination reste telle qu'elle était.

3.5 Permissions, garde et accès distant (fonctions 12 à 14)

La vue par permission (12) et la garde (13) sont le sujet de la Section 4. La **lecture distante partielle** (14) utilise des requêtes HTTP `Range` [24] : le lecteur télécharge l'en-tête et le pied, où se trouvent le schéma, l'index des blocs et les zones, puis seulement les blocs dont il a besoin (Figure 4). Nous avons mesuré avec `wremoto` 1.2.1 contre un nginx local servant le fichier de 100 000 enregistrements (le `com.jsw` de la Section 5, sous le nom `cartoes.jsw`) ; avec `file://` au lieu de l'adresse HTTP, `wremoto` donne les mêmes chiffres, sans serveur :

```
$ 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)
```

Listing 6. Pour lire une ligne au milieu d'un fichier de 79,1 Mo, le lecteur en a téléchargé 0,98 % (« baixado no total ») ; pour le filtre, 0,84 %, parce que les zones ont écarté les autres blocs avant le téléchargement.

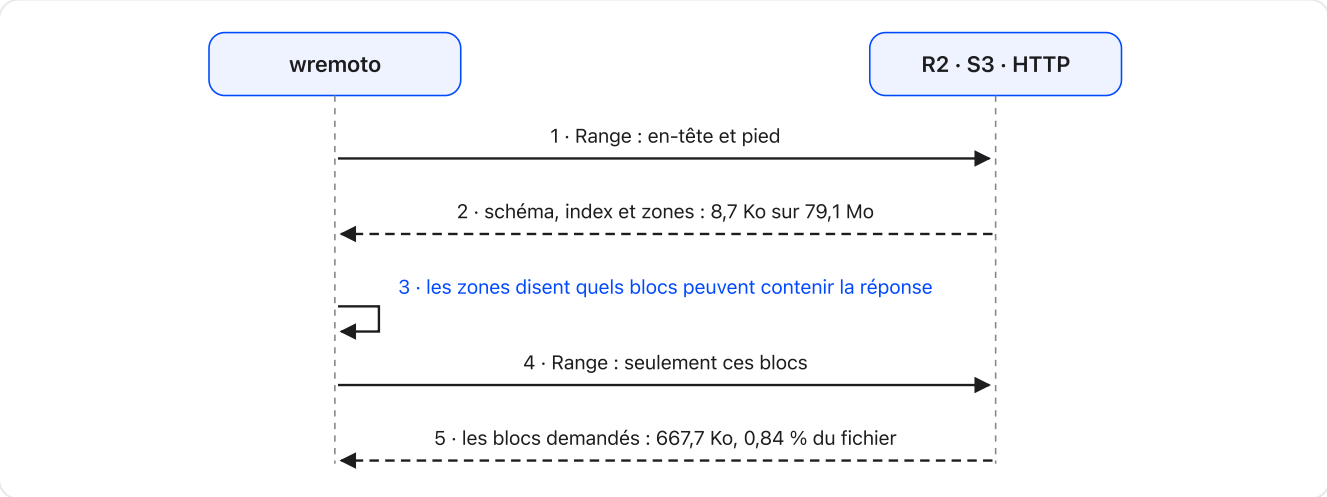


Figure 4. Lecture distante partielle du fichier de 100 000 enregistrements, avec les chiffres mesurés au Listing 6. Avec R2 ou S3, chaque requête est signée avec AWS SigV4 [25].

3.6 Graphie, signaux et colonne par bloc (fonctions 15 à 19)

Graphie et son (fonction 15). L'opérateur `~~` du filtre donne à chaque valeur une note de 0 à 1 : la plus haute de deux mesures de Dice sur des trigrammes de caractères Unicode, l'une sur la graphie normalisée (sans accents ni majuscules), l'autre sur une clé phonétique de la langue demandée. Il existe une clé pour le portugais du Brésil, l'anglais américain, l'espagnol et le chinois écrit en pinyin ; pour les idéogrammes, les trigrammes comptent caractère par caractère, et leur clé de son, qui dépend d'une table de lectures, n'est pas encore dans cette version. Le filtre n'utilise pas d'index : il ne parcourt que la colonne demandée, en parallèle (Listing 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"}
```

Listing 6B. Graphie et son proches. « Kokakola » trouve « Coca-Cola » : la clé du portugais ramène le « c » dur et le « k » au même son. `~~en`, `~~es` et `~~zh` changent la langue ; `~~pt@0.4` change le seuil.

Signaux de la communication (fonctions 16 à 18). Ils mesurent comment la personne a parlé, écrit ou est apparue, et ne décident rien : c'est le modèle qui lit la ligne qui interprète. Dans l'audio (PCM 16 bits), la hauteur vient de l'autocorrélation normalisée, la même méthode que Praat ; le jitter et le shimmer, des impulsions glottiques ; les formants, de la prédiction linéaire (LPC) ; et les pauses, du seuil d'Otsu sur l'intensité, qui tire la coupure des données elles-mêmes au lieu d'un nombre fixe. Dans le texte, le lexique de la langue (pt, en, es ou zh, écrit à la main) est reconnu par l'automate de la fonction 9, et les comptes de forme (majuscules, ponctuation, emoji, rires) n'utilisent pas de dictionnaire. Le chinois, qui ne sépare pas les mots par des espaces, est reconnu idéogramme par idéogramme. Dans la vidéo, l'appelant décode et fournit les images et le son ; chaque image devient une vignette avec lumière, contraste, couleur et forme, et les coupes de scène sortent du même seuil d'Otsu sur la différence entre images, seulement quand la série a deux groupes nets. Chaque mesure sort comme une ligne de jsonW-texte dans le schéma déclaré, prête à être écrite dans un `.jsw`.

Colonne par bloc et par tenant (fonction 19). Trois nouveaux codecs : le dictionnaire local de chaque bloc, avec et sans codage entropique, pour le texte qui se répète dans le bloc mais pas dans tout le fichier ; et les chiffres à largeur fixe, pour les codes faits uniquement de chiffres, comme le numéro d'une demande. La zone de chaque bloc garde aussi le tenant et, avec la vue d'une session, le bloc d'un autre client n'est même pas ouvert. Le filtre et le calcul comparent le code du dictionnaire par lots, avec des instructions vectorielles (SSE2), et évaluent le « contient » sur le bloc entier d'un coup. Un fichier écrit avec ces codecs ne s'ouvre pas dans les lecteurs 1.2.x. Dans la même version, le niveau maximal (`--max`) a gagné deux codecs qui écrivent chaque cellule et chaque vecteur comme la différence avec la ligne précédente : le préfixe et le suffixe communs et seulement le milieu qui change. L'effet sur la base de l'INPI est dans les Tableaux 5C et 5D.

4 Sécurité et garde

4.1 Garde

Chaque bloc du fichier tient seul et porte deux contrôles : un CRC32C [11], rapide, pour les dommages accidentels, et un hachage cryptographique, SHA-256 [12] par défaut ou BLAKE3 [13] en option. Les hachages forment une chaîne qui couvre le fichier entier : tout octet modifié dans les données, une zone ou le dictionnaire fait échouer `conferir` (vérifier) (Figure 2).

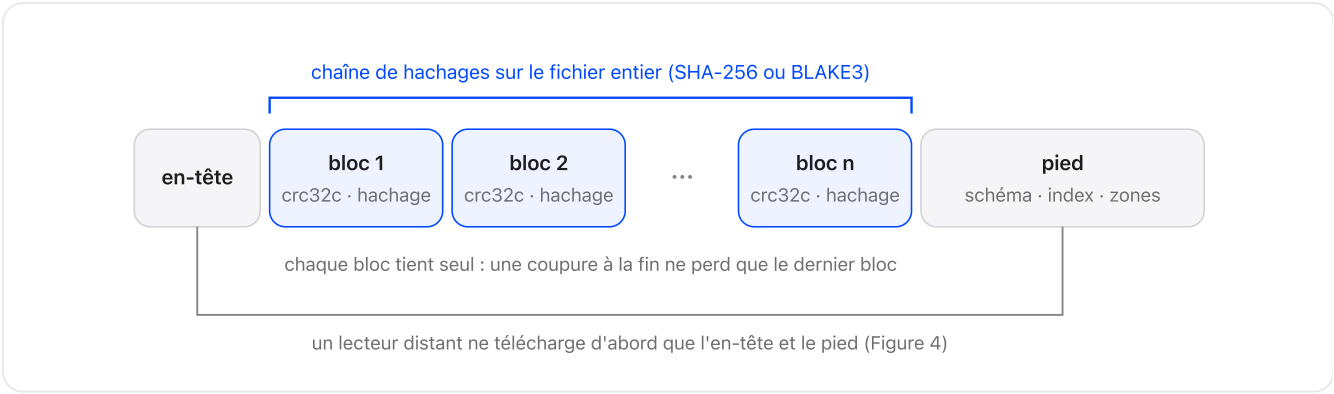


Figure 2. Le fichier `.jsw` vu de l'extérieur, au niveau décrit dans la documentation publique. L'organisation interne de l'en-tête, des blocs et du pied n'est pas publiée (Section 1.3).

```
$ wjsw conferir vendas.jsw
ok: 2 linhas, 1 blocos, sha256 e crc32c conferidos
# un octet modifié au milieu du fichier
$ wjsw conferir adulterado.jsw
FALHOU: bloco 0: nao veio ou nao confere
# les 20 derniers octets coupés, comme lors d'une écriture interrompue
$ 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 au lieu 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
```

Listing 7. Garde, falsification et réparation, avec la sortie réelle (« FALHOU » = échec ; « sem rodape: rode wjsw reparar » = pas de pied : lancez wjsw reparar).

> La garde prouve l'intégrité, pas l'origine. Un hachage sans clé peut être recalculé par quiconque réécrit le fichier entier. Pour prouver qu'un fichier est bien celui qui est sorti d'une source, gardez à part l'identifiant du fichier affiché par `wjsw info`, ou signez-le avec un outil séparé, et comparez.

4.2 Lecture de fichiers hostiles

Le lecteur traite tout fichier comme hostile : il vérifie chaque index et chaque borne avant usage, refuse une imbrication de plus de 1000 niveaux et est testé par fuzzing (entrées générées au hasard) dans la batterie de tests. Les guillemets, virgules, barres et n'importe quel octet dans un texte sont des données et ne cassent jamais la lecture.

4.3 Vue par permission

Dans jsonW, les permissions sont déclarées dans le schéma et appliquées par celui qui sert le fichier, avant de répondre : ce que la session ne peut pas voir ne sort jamais du serveur (Figure 3, Tableau 3).

Tableau 3. Les couches de la vue par permission.

Couche	Comment on la déclare	Ce qui se passe
Colonne	`+escopo(financeiro)`, `+pessoal`	sans la portée, la colonne sort nulle et ne peut même pas servir dans un filtre
Ligne	une colonne entière nommée `tenant` ou `tenant_id`, ou marquée `+tenant`	seulement les lignes du tenant de l'accréditation ; les blocs des autres tenants ne sont même pas ouverts ; `tenant:todos` (tous) ouvre tout ; le tenant est un entier : avec une colonne `tenant` en texte, aucune ligne ne sort
Cellule	les deux ensemble	les lignes du client lui-même, sans la colonne des données personnelles

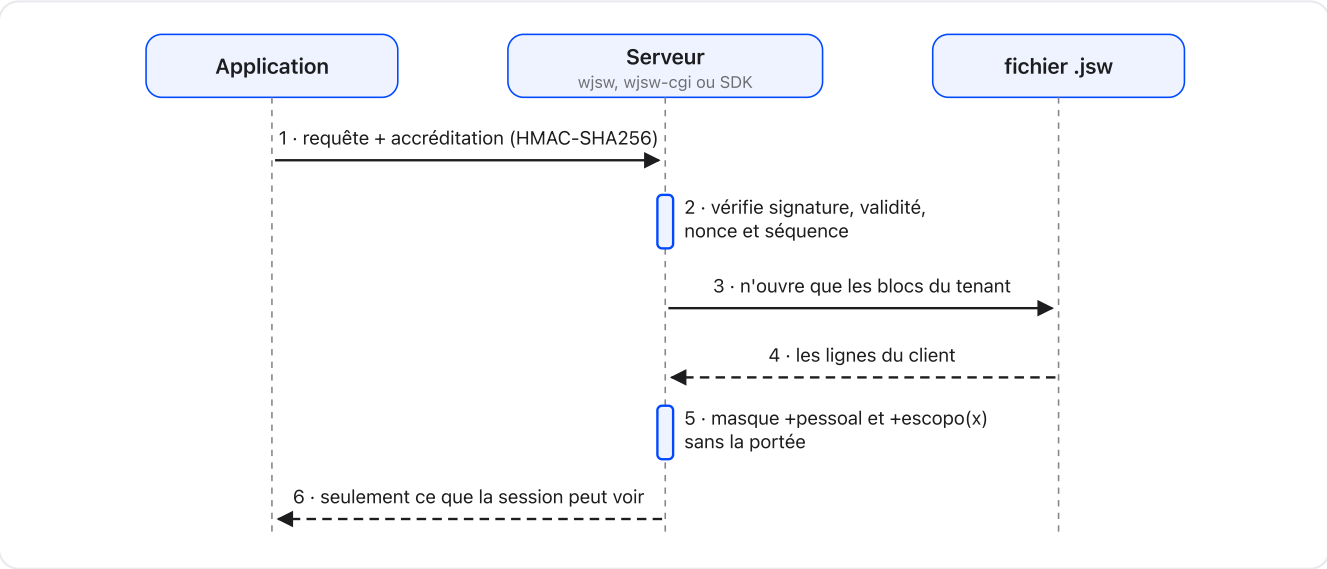


Figure 3. La vue par permission. Les étapes 2 à 5 se passent dans le serveur ; le fichier ne sort jamais entier vers celui qui le demande.

Une clinique à deux sites montre les trois couches. L'accueil du site 1 voit l'agenda du site 1 sans identifiant fiscal et sans montant ; la comptabilité du site 1 voit le montant, toujours sans identifiant fiscal ; personne du site 1 ne voit le site 2 :

```
$ wjsw de-texto "paciente:texto,cpf:codigo?
+pessoal,valor:dinheiro+escopo(financeiro),data:tempo,tenant:i64" clin.jsonl clin.jsw
# accueil du site 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}
# comptabilité du site 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}
```

Listing 8. La même requête, deux accréditations, deux réponses. Les données de l'exemple sont fictives.

> La vue n'est pas un chiffrement. Elle protège ce qui sort du serveur ; qui détient le fichier `.jsw` détient toutes les données, et `para-texto` exporte tout, car c'est la copie pour qui possède déjà le fichier. Pour montrer des données à un utilisateur, utilisez `ver`, `wjsw-cgi` ou le SDK avec `visao()` ; pour protéger le fichier au repos, utilisez le chiffrement du disque ou du stockage.

4.4 Accréditation

L'accréditation qui accompagne une requête est elle-même une valeur jsonW, avec émetteur, sujet, tenant, portées et validité, signée en HMAC-SHA256 [18]. La signature est vérifiée en temps constant, et un nonce avec un numéro de séquence empêche qu'une requête soit rejouée. Elle circule au-dessus de TLS, pas à sa place.

4.5 Transport post-quantique

La forme ligne circule sur un protocole au-dessus de QUIC [15] avec TLS 1.3 [16] et l'échange de clés hybride X25519MLKEM768 [17], qui combine la courbe X25519 avec ML-KEM-768, normalisé par le NIST dans le FIPS 203 [14]. L'argument post-quantique a deux parties. L'échange de clés utilise un mécanisme conçu pour résister à un ordinateur quantique. Et la garde et les accréditations n'utilisent que des hachages et des HMAC de 256 bits, qui ne dépendent ni de la factorisation ni du logarithme discret ; pour trouver une préimage ou la clé du HMAC, l'algorithme de Grover [19] donne une accélération quadratique et laisse environ 128 bits de sécurité ; pour trouver des collisions, la borne quantique connue est plus basse, environ 2^{85} opérations sur SHA-256 avec l'algorithme de Brassard, Høyer et Tapp [35], qui demande une mémoire quantique énorme.

La dernière ligne de `wjsw medir` (Section 5) mesure la forme ligne : un enregistrement de test occupe 5 157 octets en JSON et 1 073 octets en message, 4,8 fois moins, dans toutes les exécutions.

> Le paquet public 1.3.11 ne contient ni serveur ni client du protocole.

4.6 Serveurs web

Le paquet inclut `wjsw-cgi`, qui fonctionne sur Apache, nginx, LiteSpeed, Caddy et IIS [20]. Toute requête de `.jsw` passe par lui : un fichier sans colonnes marquées et sans tenant sort sans filtre, avec `Range` et CORS ; les autres sortent filtrés, avec la vue appliquée. La portée vient du serveur après l'authentification, jamais d'un en-tête envoyé par le client, et demander une colonne masquée par `campos=` ou `onde=` renvoie 403, si bien que rien ne fuit par déduction. Dans la version 1.2.0, la configuration d'exemple laissait le serveur livrer le fichier brut quand la requête n'avait pas de paramètres ; la 1.2.1 l'a corrigé, et qui a installé l'ancienne configuration doit la remplacer [20].

5 Mesures

Cette section mesure jsonW 1.3.11 face au JSONL et dit, à côté, ce que la mesure ne prouve pas. Les chiffres ont été refaits pour ce texte le 13 novembre 2026, avec le binaire de la version publiée, sur la même machine que la mesure du 12 novembre.

5.1 Méthodologie

- **Outil :** `wjsw medir` (mesurer), livré dans le binaire public, avec 100 000 lignes : `wjsw medir <dossier> 100000`. Binaire `bin/linux-x86_64/wjsw` du dépôt public, version 1.3.11 (préversion, SHA-256 `cb119d7473e7d786…`).
- **Machine :** un serveur virtuel avec un AMD EPYC 9355P, 8 vCPU et 18 Go de mémoire, Ubuntu 26.10, noyau Linux 7.3. Les fichiers étaient sur un `tmpfs` (mémoire), la mesure exclut donc le disque. La machine n'était pas isolée d'autres charges.
- **Données :** 100 000 enregistrements synthétiques, en forme de fiche d'une base de connaissances : `id:i64,tipo:dic,titulo:texto,corpo:texto,fonte:dic,valor:decimal[4],vetor:vf16[384],tags:lista<texto>`. Les textes répètent un modèle avec le numéro de ligne ; `tipo` et `fonte` alternent entre 5 valeurs ; le vecteur a 384 composantes multiples de 1/1024, que `vf16` stocke sans perte.

- **Le côté JSON :** le même ensemble écrit en JSONL et lu par le lecteur de texte de jsonW lui-même, qui analyse chaque ligne et convertit les types du schéma. Les deux fichiers sont chargés en mémoire avant le départ du chronomètre.
- **Répétition :** 10 exécutions consécutives. Nous donnons la médiane de chaque temps et, pour les rapports, la médiane et l'intervalle sur les 10. Les sorties brutes sont publiées avec ce texte.
- **Contrôle :** à chaque exécution, l'outil vérifie que les deux lectures ont compté 100 000 lignes avec la même somme.

5.2 Temps

Tableau 4. Temps pour écrire et lire les 100 000 enregistrements. Temps : médiane de 10 exécutions. Rapport : JSONL ÷ jsonW à chaque exécution, avec la médiane et l'intervalle sur les 10. Dans la ligne à 8 threads, le rapport est calculé contre la lecture du JSONL sur un thread. Le JSONL est écrit et lu par le code texte de jsonW lui-même.

Opération	JSONL	jsonW	Rapport (intervalle)
Écrire	15 211 ms	131 ms	114× (88 à 127)
Tout lire, tous les champs	3 687 ms	78 ms	46× (43 à 51)
Tout lire, 8 threads	(pas de version parallèle)	9,0 ms	419× (365 à 532)
Lire une colonne (`valor`)	3 687 ms	1,59 ms	2 310× (1 592 à 2 784)
Aller à la ligne N, champ `titulo`	lu depuis le début	0,09 µs (0,05 à 0,16)	pas de rapport défini

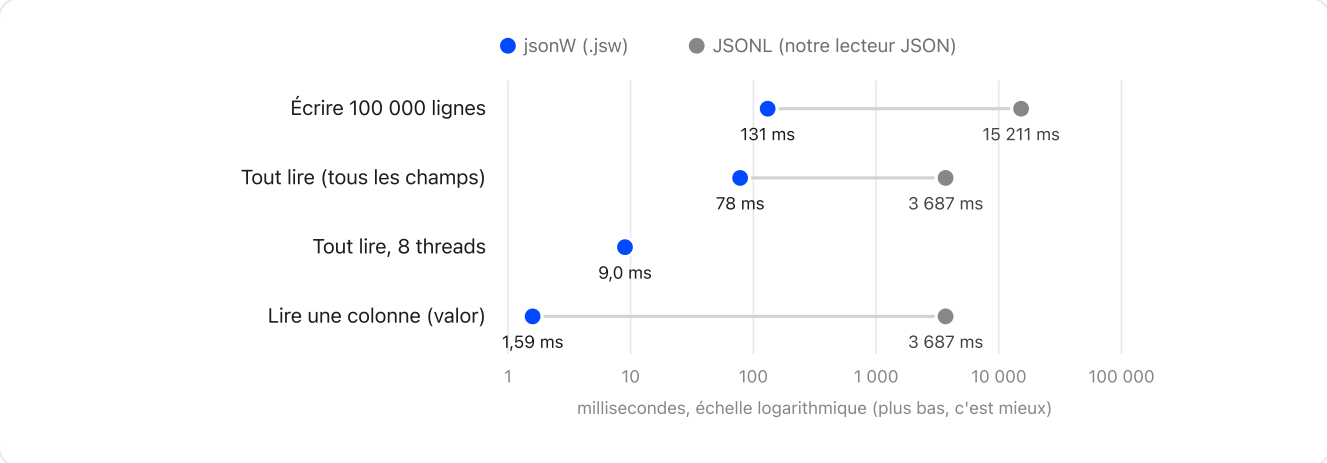


Figure 6. Les temps du Tableau 4 en échelle logarithmique (médiane de 10 exécutions). La distance horizontale entre les points de chaque ligne est le rapport.

La Figure 6 montre les mêmes temps en échelle logarithmique. La plus grande différence est dans la lecture d'une colonne : le JSONL oblige à analyser chaque ligne entière pour en extraire un champ, et le fichier en colonnes ne lit que cette colonne. C'est un avantage structurel, le même que tout format en colonnes a sur un format en lignes. Pour « aller à la ligne N », le JSONL ne peut pas sauter : il faut le lire depuis le début jusqu'à la ligne, le rapport dépend donc de N et nous ne le donnons pas.

5.3 Taille

Pour comparer la taille avec les compresseurs généralistes, nous avons réécrit les mêmes 100 000 enregistrements : un générateur en `awk`, publié avec ce texte, produit le JSONL ; `wjsw de-texto` écrit le `.jsw` ; et le JSONL rendu par `wjsw para-texto --json` fait 519 264 563 octets, la même taille, à l'échelle du Mo, que celle que `wjsw medir` indique pour son propre JSONL (519,3 Mo) ; le `.jsw` fait 79 076 300 octets (79,1 Mo dans les deux). Le JSONL du générateur a 200 000 octets de plus, parce que la valeur décimale y

est entre guillemets ; le Tableau 5 utilise le JSONL réécrit par jsonW, et avec le fichier du générateur les rapports changent à la première décimale (gzip : 19,7× au lieu de 19,8×). Le JSONL a été compressé avec gzip 1.14 [32], zstd 1.5.7 [31] et xz 5.8.3 [33] à leurs niveaux maximaux habituels.

Tableau 5. Taille des 100 000 enregistrements, en Mo (10^6 octets), avec et sans la colonne du vecteur. Rapport = JSONL non compressé ÷ la taille de cette ligne.

Fichier	Avec vecteur	Rapport	Sans vecteur	Rapport
JSONL	519,3	1,0×	38,2	1,0×
`.jsw`, vecteur `vf32`	155,9	3,3×	s.o.	s.o.
`.jsw`, vecteur `vf16`, niveau par défaut	79,1	6,6×	2,22	17,2×
`.jsw`, vecteur `vq8` (avec perte)	41,1	12,6×	s.o.	s.o.
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`, vecteur `vf16`, `--max`	1,36	381,1×	0,0175	2 185×

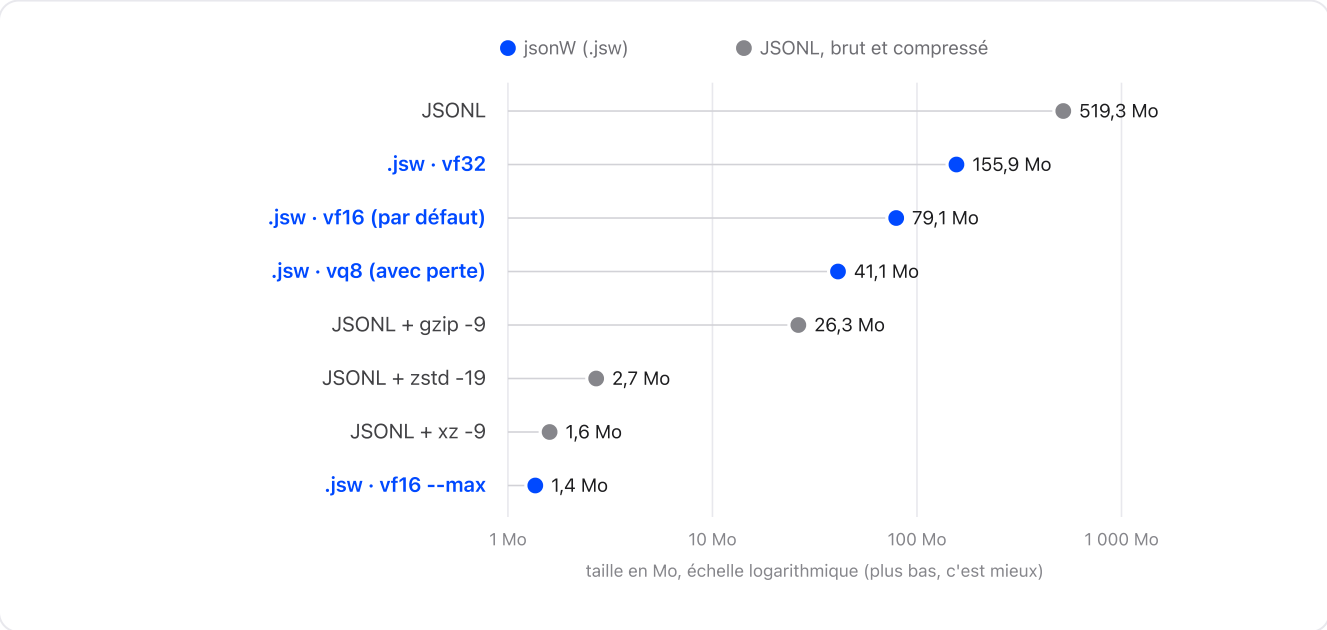


Figure 5. Les tailles avec vecteur du Tableau 5, en échelle logarithmique.

La Figure 5 montre les tailles avec vecteur. Face au JSONL non compressé, qui est la façon habituelle de stocker et d'échanger du JSONL, le `.jsw` par défaut est 6,6 fois plus petit, et presque toute sa taille est le vecteur : 384 composantes de 2 octets sur 100 000 lignes font 76,8 Mo. Face au JSONL compressé avec zstd ou xz, le `.jsw` par défaut est plus gros. Au niveau maximal, la 1.3.11 est passée sous les deux : 1,36 Mo avec vecteur, contre 2,67 Mo pour zstd et 1,56 Mo pour xz, et 17,5 ko sans vecteur, contre 0,63 et 0,47 Mo ; dans la 1.2.1, le même niveau donnait 3,75 et 2,19 Mo. Le gain vient de deux nouveaux codecs du niveau maximal, qui écrivent chaque cellule et chaque vecteur comme la différence avec la ligne précédente (le préfixe et le suffixe communs et le milieu qui change), et il est grand ici parce que les données du test sont très répétitives : les textes se répètent avec un numéro changé et le vecteur suit un motif arithmétique. Le retour est sans perte : le `para-texto` du fichier au niveau maximal est identique, octet par octet, à celui du fichier au niveau par défaut (même SHA-256, avec et sans vecteur). Sur des données réelles, le gain est bien moindre : sur la

base de l'INPI, le niveau maximal de la 1.3.11 est 16,5 % plus petit que celui de la 1.2.1 (Tableau 5C). Les vecteurs de modèles réels se compressent aussi peu en général ; nous ne l'avons pas mesuré ici, mais si c'est le cas, la taille d'un `.jsw` avec vecteurs reste proche de la taille brute des vecteurs.

Ce qu'un fichier compressé généraliste ne donne pas, c'est l'accès. Pour lire un champ d'un JSONL compressé avec zstd, il faut tout décompresser (de 0,13 s à 0,24 s pour décompresser `com-volta.jsonl.zst`, en 3 exécutions sur la même machine, hors de l'environnement de la mesure) puis analyser le texte, alors que le `.jsw` lit le champ directement, saute des blocs grâce aux zones, vérifie la garde et peut être lu à distance par morceaux.

5.4 Données réelles : CSV, XML, XLSX et Parquet

Les mesures ci-dessus utilisent des données générées. Pour comparer avec les formats de tableur et d'échange les plus courants, nous avons utilisé des données publiques : le premier million de lignes du fichier `MARCAS_DADOS_BIBLIOGRAFICOS.csv`, issu des données ouvertes sur les marques de l'INPI, l'office brésilien des brevets et des marques (13 colonnes : numéro de la demande, dates, présentation, nature, nom de la marque, statut). Un programme en C, sans bibliothèque, a lu le CSV selon la RFC 4180 (guillemets, virgules et retours à la ligne dans un champ) et a écrit les mêmes lignes sous quatre formes : le CSV réécrit ; un XML avec un élément par ligne et un enfant par colonne ; un XLSX monté par le programme lui-même (Office Open XML, cellules de texte en ligne, compressé avec `zip -6`) ; et le `.jsw`, avec le schéma écrit à la main (dates en texte ; présentation, nature et statut en dictionnaire). Un champ vide est absent en XML et en jsonW. Le XLSX accepte au plus 1 048 576 lignes par feuille, d'où le million.

La requête a été la même dans les quatre : compter les enregistrements en vigueur (`codigo_situacao = IPAS023`). Les quatre formes ont donné le même résultat, 352 248 enregistrements, et `wjsw conferir` a vérifié le SHA-256 et le CRC32C de tous les blocs du `.jsw`. Chaque requête a été exécutée 5 fois, avec le fichier déjà en cache ; la plus basse compte. Machine : serveur à 8 cœurs sous Linux 7.3, jsonW 1.2.0, 13 novembre 2026.

Tableau 5B. Un million de lignes réelles de l'INPI en quatre formats. Taille en Mo (10^6 octets) ; requête = compter les enregistrements en vigueur, en millisecondes.

Format	Taille	Avec zstd -3	Requête	Comment la requête a été faite
XML	502,1	46,0	159	`grep -c` sur le fichier entier
CSV	141,1	36,1	97	`grep -c` sur le fichier entier
XLSX	74,3	s.o.	9 364	`unzip -p` de la feuille + `grep`
`.jsw`	51,2	s.o.	119	`wjsw calcular --onde`, la colonne seule

En taille, le `.jsw` a été 2,8 fois plus petit que le CSV, 1,5 fois plus petit que le XLSX et 9,8 fois plus petit que le XML. Le CSV et le XML compressés avec zstd sont plus petits que le `.jsw` (36,1 et 46,0 Mo contre 51,2 Mo), mais, comme à la Section 5.3, ils doivent être décompressés en entier avant toute lecture. Dans la requête, le `.jsw` a été 79 fois plus rapide que le XLSX, qu'il faut décompresser à chaque lecture. Face au CSV et au XML, `grep` a été plus rapide ou proche : il ne cherche qu'un bout de texte dans le fichier entier, sans séparer les colonnes ni les types, et donnerait un résultat faux si le même code apparaissait dans une autre colonne. Dans cet ensemble il n'apparaît pas, et les décomptes ont concordé. Une requête avec la même garantie sur le CSV exige de lire et de découper chaque ligne, ce qu'`awk` a fait en 1 597 ms au premier passage de cette mesure, sans gérer les guillemets.

Ce qu'aucun des trois formats n'apporte par lui-même : schéma typé (date, décimal, argent, vecteur), lecture d'une colonne sans lire les autres, recherche par termes et par sens, vue par permission, historique des lignes supprimées et garde par bloc. Le XLSX a des types de cellule et plusieurs feuilles, mais limite la taille de la feuille et doit être décompressé pour être lu.

La base entière. Le Tableau 5B utilise un million de lignes d'un fichier. Pour voir le format avec tout, nous avons converti les sept fichiers des données ouvertes sur les marques de l'INPI [38], téléchargés le 12 novembre 2026, sans rien couper : 55 278 212 lignes et 12,53 Go de CSV, avec toutes les colonnes et l'historique complet des décisions de chaque demande (24 978 976 lignes dans le seul `MARCAS_DESPACHOS.csv`). Chaque fichier a reçu un schéma écrit à la main. Nombres, dates et codes ont gardé leur propre type ; le texte, les octets tels qu'ils sont arrivés. Le convertisseur, en C, vérifie chaque valeur typée sur le champ : le nombre et la date doivent redonner le même texte que dans le CSV, sinon la conversion s'arrête. Cela a relevé deux cas réels dans la source, gardés tels qu'ils sont arrivés : une date de priorité en l'an 1018 (une faute de frappe dans l'original) et une ligne de la classification de Nice sans numéro de demande, parmi les 5 333 893 de ce fichier.

Le Parquet [3] a été écrit par DuckDB 2.0.0 [39] avec les mêmes types et une compression zstd. Le XML suit la disposition du Tableau 5B. Le XLSX est resté dehors : six des sept fichiers dépassent la limite de 1 048 576 lignes par feuille, et celui des décisions fait presque 24 fois cette limite. Les huit formes ont donné les mêmes décomptes dans les trois requêtes, à une exception près, expliquée plus bas. `wjsw conferir` a vérifié le SHA-256 et le CRC32C de tous les blocs des sept `.jsw`. Chaque requête a été exécutée 5 fois, avec le fichier déjà en cache, et la plus basse compte. L'écriture du jsonW a utilisé un cœur. DuckDB, zstd et xz ont utilisé les huit cœurs, et `wjsw calcular` aussi. Machine : la même que pour le Tableau 5B, 31 Go de mémoire, 13 novembre 2026.

Tableau 5C. La base entière des marques de l'INPI (sept fichiers, 55,3 millions de lignes) dans chaque format. Taille en Mo (10^6 octets), additionnée sur les sept fichiers ; écrire = temps pour produire le format à partir du CSV, en secondes.

Format	Taille	Fois plus petit que le CSV	Écrire	Lit une colonne sans ouvrir le reste ?
XML	24 804,7	0,5	s.o.	non
CSV (original)	12 527,4	1,0	s.o.	non
`.jsw` , 1.2.1	2 804,8	4,5	176,7	oui
`.jsw` , 1.3.11	2 579,2	4,9	s.o.	oui
`.jsw` , 1.2.1, `--max`	2 321,8	5,4	184,9	oui
XML + zstd -3	2 210,0	5,7	21,4	non, décompresse tout
`.jsw` , 1.3.11, `--max`	1 938,1	6,5	s.o.	oui
CSV + zstd -3	1 908,9	6,6	13,2	non, décompresse tout
Parquet (zstd)	1 665,8	7,5	16,6	oui
CSV + xz -6	1 161,0	10,8	544,5	non, décompresse tout

Tableau 5D. Trois requêtes par colonne sur la base entière, en millisecondes (la plus basse de 5), mesurées pour la cinquième fois le 13/11/2026 à 06:21 UTC, avec le `wjsw` 1.3.11 publié et la 1.2.1, à la même heure. En vigueur : compter `codigo_situacao = IPAS023` parmi 6,7 millions de demandes. Historique : toutes les décisions d'une demande (`numero_inpi = 818602856`) parmi 25,0 millions. Classe 35 : compter `classe_nice = 35` parmi 5,3 millions. Entre parenthèses, DuckDB avec un cœur. Le XML vient du premier passage, car le fichier a été effacé ensuite (il occupait 24,8 Go).

Format et lecteur	En vigueur	Historique	Classe 35
Bonne réponse	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 qui lit le CSV	375 (1 447)	1 205 (4 523)	592 (1 979)
CSV, `grep -c`	519	6 690	1 278 (fausse : 1 076 929)
CSV + zstd, `zstd -dc | grep -c`	798	3 129	2 786 (fausse)
CSV + xz, `xz -dc | grep -c`	1 050	3 025	2 105 (fausse)
CSV + xz, `xz -dc | grep -c`	971	2 628	1 950 (fausse)
XML, `grep -c` (premier passage)	3 017	9 925	3 430

Ce que montre la base entière. En taille, le `.jsw` 1.3.11 est 4,9 fois plus petit que le CSV au niveau par défaut et 6,5 fois au maximum, et 12,8 fois plus petit que le XML ; le 1.2.1 était 4,5 et 5,4 fois plus petit que le CSV, et le 1.3.11 au maximum est 16,5 % plus petit que le 1.2.1. Dans les requêtes, avec huit cœurs des deux côtés, le 1.3.11 a compté les enregistrements en vigueur en 43 ms au niveau maximal et 44 ms au niveau par défaut, devant les 54 ms de Parquet lu par DuckDB ; avec un seul cœur, DuckDB sur Parquet a pris 76 ms pour ce comptage et 521 ms pour l'historique. Sur les trois requêtes, le `.jsw` a été plus rapide que DuckDB lisant le CSV lui-même, que `grep`, que le CSV compressé et que le XML. Pour l'historique d'un dossier et pour la classe 35, le `.jsw` a pris 207 et 142 ms, et Parquet avec huit cœurs, 129 et 41 ms : c'est le prochain pas de performance de la section 8. En taille, Parquet (1 665,8 Mo) et le CSV compressé avec xz (1 161,0 Mo) occupent moins que le `.jsw` au maximum (1 938,1 Mo), et le CSV compressé doit être entièrement décompressé avant toute lecture ; avec zstd, le CSV est pratiquement égal au `.jsw` au maximum (1 908,9 Mo). L'écriture du `.jsw` 1.2.1, sur un cœur, a pris 177 s ; celle de Parquet, avec huit, 17 s. Les fichiers 1.3.11 ont été réécrits à partir du `.jsw` 1.2.1 (`wjsw regravar`), c'est pourquoi la table 5C ne donne pas leur temps d'écriture.

Les passages et les corrections. Le Tableau 5D a été mesuré cinq fois, toujours avec le Parquet et le CSV à la même heure. Avec le `wjsw` 1.2.0, le `.jsw` a pris 677, 1 494 et 1 114 ms sur les trois requêtes, de 10 à 31 fois le temps du Parquet : le filtre lisait toutes les colonnes de chaque ligne. Corrigé pour ne lire que la colonne demandée, il a pris 82, 311 et 209 ms. Au troisième passage, l'évaluation s'est faite par bloc, en comparant le code du dictionnaire et les entiers directement dans la colonne et en ne construisant la ligne que lorsqu'elle passe : 48, 344 et 186 ms. Au quatrième, avec la 1.3.11 encore en préparation, l'historique est passé de 355 à 279 ms. Au cinquième, avec le binaire de la version publiée, qui compare le dictionnaire par lots avec des instructions vectorielles et évalue le « contient » par bloc, la 1.3.11 a pris 44, 254 et 152 ms par défaut et 43, 207 et 142 ms au maximum, contre 54, 353 et 198 ms pour la 1.2.1 à la même heure. L'historique cherche le numéro de demande, un code fait uniquement de chiffres que la 1.3.11 écrit à largeur fixe (dans

DADOS_BIBLIOGRAFICOS, cette seule colonne est passée de 53,0 à 25,7 Mo). La machine n'était pas isolée, et le même binaire a varié jusqu'à 23 % d'un passage à l'autre (le Parquet, de 41 à 54 ms pour le comptage en vigueur) ; la comparaison vaut donc au sein d'un même passage. Un fichier écrit avec les nouveaux codecs ne s'ouvre pas dans les lecteurs 1.2.x. Les temps de `grep` sur le CSV ont varié encore plus (de 2 102 à 6 690 ms sur l'historique), car les 12,5 Go de CSV ne tiennent pas tous dans le cache à côté des autres fichiers, et au cinquième passage les réécritures juste avant ont sorti le CSV du cache ; c'est le chiffre du même passage qui compte.

L'exception dans les décomptes est celle de la classe 35. `grep` sur le CSV, brut ou compressé, a compté 1 076 929, un de plus que la bonne réponse : le bout de texte `,35,` apparaît aussi dans une autre colonne d'une ligne. C'est le risque décrit à côté du Tableau 5B, qui ne s'était pas produit dans cet ensemble et s'est produit ici. Tous les lecteurs qui séparent les colonnes (jsonW, Parquet, DuckDB lisant le CSV et le XML avec le nom de la colonne dans le balisage) ont donné 1 076 928.

De cette conversion 1:1, la seule fonction au-delà du type et de la colonne qui est entrée est la garde : chaque bloc des sept fichiers a un CRC32C et un SHA-256, vérifiés plus haut. La conversion n'a marqué aucune permission ni monté de recherche, pour rester identique au CSV. Le format permet, dans le même fichier, de cacher le CPF et le CNPJ des titulaires à qui n'a pas la portée (Section 4.3), de n'en lire à distance qu'une partie et de chercher par termes et par sens (Section 3). Le Parquet a une somme de contrôle optionnelle par page [36] et un chiffrement par colonne [5], mais pas la chaîne de hachages du fichier entier ni la vue par portée ; le CSV et le XML n'apportent rien de tout cela.

5.5 Limites de la mesure

- **Le lecteur JSON est le nôtre.** Le côté JSON a été écrit et lu par le code texte de jsonW lui-même, qui valide et convertit les types ; il lit le JSONL à environ 140 Mo/s. Les meilleurs lecteurs JSON, comme simdjson [23], dépassent 1 Go/s. Avec l'un d'eux, les lignes « écrire » et « tout lire » du Tableau 4 se réduiraient beaucoup. En estimation, pas en mesure : à 2 Go/s, lire les 519 Mo prendrait environ 260 ms, et le rapport de « lire une colonne » serait dans les centaines, pas dans les milliers.
- **Données synthétiques et répétitives,** comme dit à la Section 5.3. Le rapport de taille dépend beaucoup des données, et plus encore au niveau maximal, qui garde la différence entre lignes successives : mesurez avec les vôtres.
- **Une machine, un système, la mémoire au lieu du disque.** Les temps absolus changent avec le processeur, le stockage et la charge ; sur les 10 exécutions, le plus grand rapport de chaque opération a été de 19 % à 75 % au-dessus du plus petit ; celui de la lecture d'une colonne varie le plus, parce que le temps de jsonW y est de 1,6 ms.
- **Parquet mesuré seulement avec les données de l'INPI et un lecteur.** La comparaison de performances avec Parquet est dans les Tableaux 5C et 5D, avec DuckDB 2.0.0 comme écrivain et lecteur. Un autre lecteur, une autre compression ou une autre taille de groupe de lignes changent les chiffres. Arrow, MessagePack, CBOR et Protobuf restent sans mesure de performances dans cette version du texte ; la Section 6 compare des fonctions, pas des vitesses.

5.6 Chiffres publiés auparavant

Jusqu'au 12 novembre 2026, le site et le dépôt de jsonW citaient « 4,9 fois plus petit que le JSONL » et « 3 281 fois plus rapide pour lire un champ », issus de la première mesure, du 6 novembre, avec la version 1.0 sur une autre machine. Le texte 1.0 de cet article, du 12 novembre, a refait la mesure avec la 1.2.1 : 6,6 fois plus petit et environ 2 500 fois plus rapide pour lire une colonne, avec le niveau maximal à 3,75 Mo avec vecteur et 2,19 Mo sans. Refaite pour ce texte avec la 1.3.11, sur la même machine, elle donne 6,6 fois plus petit et

environ 2 300 fois plus rapide pour lire une colonne (l'écart reste dans la variation entre exécutions), et le niveau maximal tombe à 1,36 Mo et 17,5 ko. Ce sont les chiffres de ce texte qui valent, avec la date, la machine et la version à côté.

5.7 Comment reproduire

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

# Tableau 5 : le même ensemble en JSONL, le .jsw et les compresseurs
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*
```

Listing 9. Les mesures de ce texte sur votre machine. `VET=0` et le schéma sans la colonne `vetor` donnent la colonne « sans vecteur ».

6 Comparaison avec d'autres formats

Le Tableau 6 compare ce que chaque format fait par lui-même, sans bibliothèque par-dessus. Les formats en colonnes mûrs, Parquet et Arrow, ont déjà une bonne part de l'avantage d'accès mesuré à la Section 5, et plusieurs choses que jsonW n'a pas encore.

Tableau 6. Ce que chaque format apporte dans le format lui-même. Sources dans les références indiquées dans chaque

colonne.

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Forme	texte ; une ligne par enregistrement en JSONL	binaire, sans schéma	binaire, schéma compilé (`.proto`)	fichier en colonnes	colonnes en mémoire ; fichier et flux IPC	ligne binaire, fichier en colonnes et texte
Schéma	non (JSON Schema à part)	non	oui, hors du message	oui, dans le fichier	oui	oui, une ligne, dans le fichier
Lire un champ sans le reste	non	non	non	oui, par colonne	oui, accès direct	oui, par ligne et champ
Décimal exact	non	CBOR : oui (fraction décimale, tag 4)	pas natif	oui (DECIMAL)	oui (decimal128 et 256) [37]	oui
Argent et mesure	non	non	argent via un message commun (`google.type.Money`) [10]	non	non	oui, avec devise et unité
Vecteur dense	liste de nombres en texte	liste ; CBOR a des tableaux typés [34]	champ répété	liste ou octets de taille fixe	liste de taille fixe	f32, f16, 8 et 4 bits, bits ; tenseur
Permissions	non	non	non	chiffrement des colonnes avec des clés différentes [5]	non	vue par colonne, ligne et cellule, appliquée par qui sert ; sans chiffrement
Intégrité	non	non	non	CRC optionnel par page [36] ; chiffrement authentifié	pas dans le format	crc32c et hachage par bloc, chaînés
Recherche et classement dans le format	non	non	non	statistiques par groupe de lignes et par page pour sauter des données	non	zones, cosinus, creux, grille, top-k, lexique
Lecture distante partielle	non	non	non	oui (pied et Range)	oui, dans le format de fichier	oui
Spécification	ouverte (RFC)	ouverte (CBOR : RFC)	ouverte	ouverte (Apache)	ouverte (Apache)	usage, texte et ABI publics ; disposition binaire interne

	JSON · JSONL [1,2]	MessagePack · CBOR [7,8]	Protobuf [9]	Parquet [3,4,5]	Arrow [6]	jsonW [20]
Implémentations	nombreuses, indépendantes	nombreuses, indépendantes	plusieurs	plusieurs, indépendantes	plusieurs, indépendantes	une (noyau en C), avec des SDK dans 10 langages

La place de chaque format. JSON est universel et lisible partout. MessagePack, CBOR et Protobuf sont mûrs et implémentés dans des dizaines de langages, et Protobuf a des règles mûres d'évolution du schéma. Parquet et Arrow sont des standards ouverts, lus par presque tous les moteurs d'analyse, et Parquet a un chiffrement par colonne pour le fichier au repos. jsonW travaille avec tous : tout JSON est déjà du jsonW-texte, `wcon` importe et renvoie les tables des bases de données avec les bons types, et le fichier au repos peut être chiffré par le disque ou le stockage.

Ce que jsonW réunit. La différence de jsonW n'est pas une fonction isolée, mais le fait d'avoir, dans le même modèle, la forme message, la forme fichier en colonnes et une forme texte compatible avec JSON ; des types métier vérifiés à l'écriture ; des permissions par colonne, ligne et cellule appliquées à la lecture ; et des opérations de classement, de recherche et de classification dans le fichier. Le prix de ce choix est d'être, pour l'instant, une seule implémentation au noyau fermé (Section 8).

7 Usages

7.1 Modèles d'IA et recherche sémantique

Une base de connaissances tient dans un `.jsw` avec le texte, les métadonnées et le vecteur de chaque passage ; la recherche sémantique est un `avaliar … cosseno` qui ne lit que la colonne du vecteur, et le lexique et les vecteurs creux couvrent la recherche par termes. Avec les k meilleurs et le lexique, le même fichier sert de routeur qui choisit le spécialiste : le lexique classe le texte de la demande et le classement choisit.

7.2 Données d'entreprises

Les registres comme ceux des entreprises (CNPJ) et des codes postaux (CEP) du Brésil sont là où JSON perd le plus : le zéro initial disparaît quand le code devient un nombre, le capital social devient un double et la date perd son fuseau horaire. Dans jsonW, chaque chose a son type :

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

Listing 10. Un schéma de registre d'entreprises : codes exacts octet par octet, texte répété dans un dictionnaire, argent avec devise et un tenant pour séparer les clients.

7.3 Bases de données

`wcon`, le connecteur de bases de données du paquet, emmène les données de PostgreSQL, MySQL et MariaDB, SQLite, Redis et Valkey et MongoDB vers un `.jsw` et retour, avec le schéma tiré du type de chaque colonne : `numeric` devient un décimal exact, `bigint` reste un entier de 64 bits et le `vector` et le `halfvec` de pgvector [30] deviennent `vf32` et `vf16`. Les valeurs voyagent toujours comme des données, précédées de leur longueur, jamais collées dans une commande ; le retour se fait en une seule transaction. L'aller-retour

a été vérifié avec PostgreSQL 14, 16 et 18 (même md5 des deux côtés), MariaDB 11.8 (hex de chaque colonne), SQLite, Redis 7 et MongoDB 7, sous charge hostile : injection, guillemets, `\0`, accents, emoji et texte de droite à gauche [20].

7.4 Serveurs web

Avec `wjsw-cgi` (Section 4.6), un site sert un `.jsw` en JSON, en NDJSON ou tel quel, avec filtre, choix des champs et limite dans l'adresse, toujours avec la vue de la session.

7.5 Stockage sur R2 et S3

Un `.jsw` peut vivre dans un bucket Cloudflare R2 ou n'importe quel S3 et y être interrogé, sans télécharger le fichier entier (Figure 4). Comme R2 ne facture pas la sortie des données [26], interroger un morceau d'un gros fichier coûte le morceau lu plus les opérations, que R2 facture à la requête (le Listing 6 fait 5 requêtes).

8 Limites et prochaines étapes

- **Une implémentation, un noyau fermé.** La disposition binaire n'est pas publiée ; lire et écrire du `.jsw` dépend de `libjsonw`. L'ABI stable et les SDK réduisent le risque, mais ne remplacent pas une spécification ouverte avec des implémentations indépendantes.
- **Pas d'audit indépendant.** Les preuves citées (fuzzing, charges hostiles, comparaison octet par octet entre SDK) sont les nôtres.
- **Plateformes.** Il y a des binaires pour macOS arm64, Linux x86_64 (glibc 2.34 ou plus récente) et Windows x86_64. Il n'y en a pas encore pour macOS Intel, Linux arm64 ni Linux avec musl. Sous Windows, `wcon` et `wremoto` passent par WSL.
- **Fonctions sans commande publique.** L'index par clé, l'écriture en direct, la vue « seulement ce qui est en vigueur » et le compactage existent dans le noyau, mais n'ont ni commande dans `wjsw` ni fonction dans l'ABI 1.3.11 ; le format de la table de grille n'est pas documenté.
- **Le protocole n'est pas dans le paquet.** Le protocole post-quantique n'est pas fourni dans le paquet public 1.3.11.
- **Mots portugais.** Les types et les commandes utilisent des mots portugais (`texto`, `dinheiro`, `medida`, `ver`, `conferir`), ce qui demande un glossaire pour qui ne lit pas le portugais.
- **Bases de données.** MySQL 8 ne fonctionne qu'avec `mysql_native_password` ; la première authentification complète de `caching_sha2_password` n'existe pas encore.
- **Mesures.** Les limites de la Section 5.5.
- **Requête par colonne : le prochain pas.** Sur la base entière de l'INPI (table 5D), `wjsw calcular` 1.3.11 compte les enregistrements en vigueur en 43 ms, devant DuckDB lisant Parquet, et depuis la première série (1.2.0) ce comptage est passé de 677 à 43 ms. Le prochain pas est d'étendre la même lecture par bloc et par lots à l'historique d'un dossier et à la classe 35, où le `.jsw` a pris 207 et 142 ms et DuckDB avec huit cœurs 129 et 41 ms. En taille, les prochains codecs (différence et empaquetage de bits pour les entiers et les dates) visent les 1 938,1 Mo du niveau maximal.

Prochaines étapes de ce texte et du paquet : binaires pour macOS Intel, Linux arm64 et musl ; une requête qui ne vérifie que le CRC32C de chaque bloc et laisse le SHA-256 au `conferir` (le Parquet ne vérifie aucun hachage à la lecture, et une partie de l'écart restant vient de là) ; le codec de dictionnaire à fenêtre, pour des colonnes comme la classe de Nice ; le codage par différences avec empaquetage de bits pour les entiers et les dates, mesuré de nouveau face aux Tableaux 5C et 5D ; et la comparaison de performances avec Arrow sur la même machine.

9 Disponibilité et licence

L'outil, la bibliothèque, l'ABI, les SDK, les configurations de serveurs web et la documentation sont sur github.com/developer-wortic/jsonw [20], et le SDK Node.js est sur npm sous le nom `@worticteCnologia/jsonw`. L'usage est régi par la Licence d'usage durable de jsonW (en portugais), qui prévaut sur ce résumé : jsonW est libre d'usage sur autant de serveurs et d'applications que nécessaire, produits commerciaux compris, tant qu'il est un composant et non le service vendu ; sans accord écrit avec Wortic, personne ne peut vendre, héberger ni offrir à des tiers jsonW ou un service dont la fonction principale est celle de jsonW (stocker, convertir, servir ou transporter des données en jsonW), payant ou gratuit ; les bibliothèques et outils distribués en binaire ne peuvent pas faire l'objet de rétro-ingénierie, sauf dans la mesure permise par la loi ; et la disposition interne du fichier, les codecs, le protocole et le noyau appartiennent à Wortic, donc une implémentation indépendante de ceux-ci demande aussi un accord écrit. La documentation publique peut être lue, citée et utilisée librement pour construire des programmes qui utilisent jsonW.

Par transparence : la première fois qu'il s'exécute sur chaque machine, l'outil `wjsw` n'envoie que la version et la plateforme (par exemple `v=1.3.11&so=linux-x86_64`) pour que nous puissions compter les installations, et ne le refait pas ensuite. L'IP arrive au serveur avec la connexion mais n'est pas conservée, pas plus qu'aucun nom, chemin ou donnée : le serveur n'enregistre que le jour, la plateforme et la version. `JSONW_SEM_PING=1` ou le fichier `~/.jsonw/sem-ping` coupent l'avis avant la première exécution. La bibliothèque et les SDK n'accèdent jamais au réseau.

Références

[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)*, qui définit le 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: dépôt public, documentation (docs/JSONW.md), ABI (sdk/jsonw.h) et binaires*. 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*, section 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* (champ `crc` de `PageHeader`). github.com/apache/parquet-format/blob/master/src/main/thrift/parquet.thrift

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

[38] Instituto Nacional da Propriedade Industrial (INPI, Brésil). *Données ouvertes sur les marques* (`marcas.zip`, sept fichiers CSV), téléchargées le 12/11/2026. dadosabertos.inpi.gov.br/download/marcas/marcas.zip

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

Comment citer

Wortic Tecnologia. *jsonW : un modèle de données, trois formes sans perte et les permissions dans le format*. Texte 1.1, sur jsonW 1.3.11. 13 novembre 2026. Disponible sur : https://jsonw.org/fr/paper.

```
@techreport{jsonw2026fr,
  title       = {jsonW : un mod{\`e}le de donn{\'e}es, trois formes sans perte
                 et les permissions dans le format},
  author      = {{Wortic Tecnologia}},
  institution = {Wortic Tecnologia},
  year        = {2026},
  month       = nov,
  note        = {Texte 1.1, sur jsonW 1.3.11},
  url         = {https://jsonw.org/fr/paper}
}
```

Texte 1.1, publié le 13/11/2026, sur jsonW 1.3.11 (le texte 1.0, du 12/11/2026, portait sur la 1.2.1). jsonW est json Wondrous ; le noyau en C est maintenu par Wortic Tecnologia. Les corrections et les mesures de tiers sont les bienvenues via le dépôt public. La version originale est en portugais ; cette traduction garde les mêmes chiffres et les mêmes termes.