EDICIÓN #010

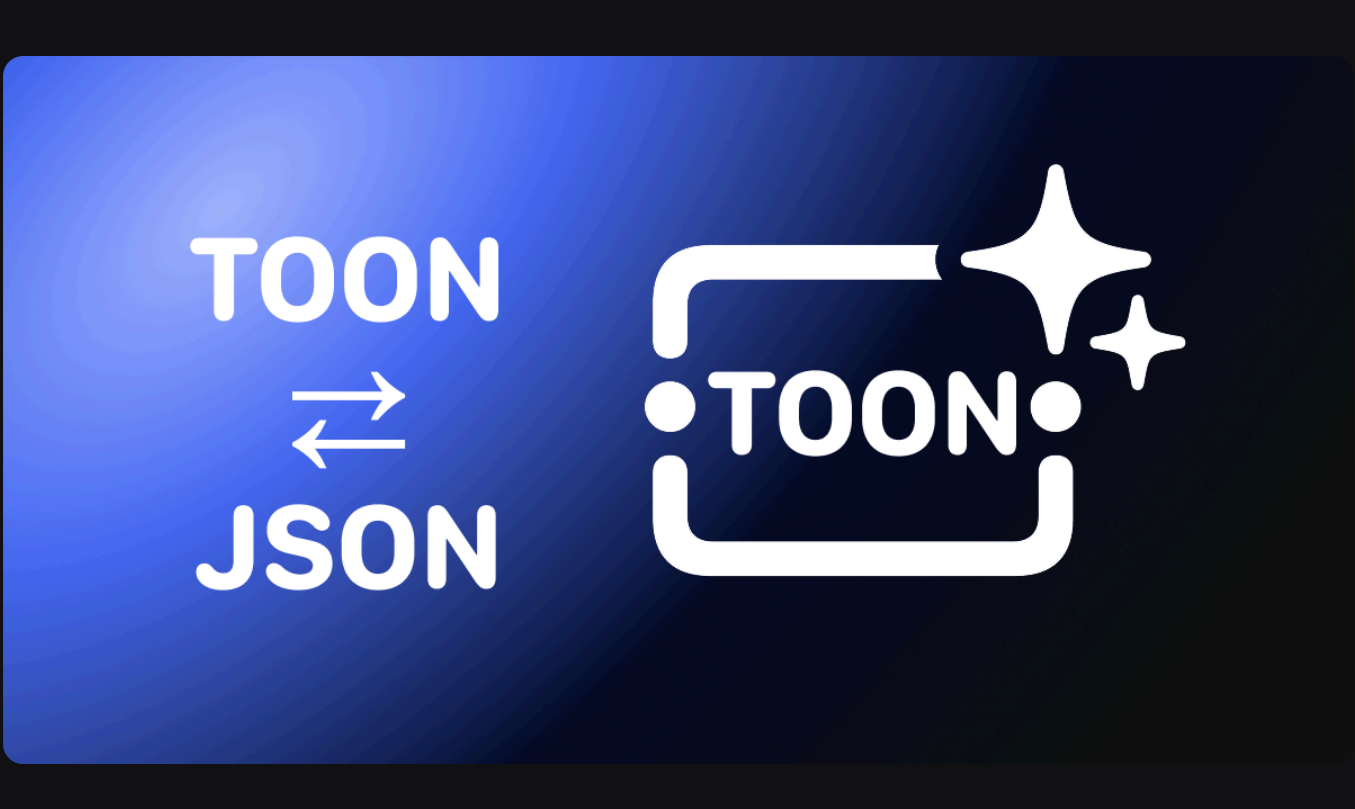
TOON: Paga menos a OpenAI y dile adiós al JSON

12 de Enero de 2026

TOON: Paga menos a OpenAI y dile adiós al JSON

Te explico qué es TOON (Token-Oriented Object Notation) y cómo empezar a usarlo en tus proyectos para ahorrar tokens con LLMs como ChatGPT o Gemini

3 December 2025



Contenido

No sé si has oído hablar de TOON. Si no te suena, tranquilo, estás en el sitio indicado. Y si ya lo conoces, quédate, porque traigo benchmarks que te van a gustar y además te regalo una herramienta gratuita que he desarrollado para transformar JSON a TOON. Vamos allá.

La premisa de TOON es simple: es un **formato de serialización** como JSON, pero diseñado **para ahorrar tokens al pasárselo a un LLM**.

Por darte un ejemplo. Según los creadores de TOON, si tienes 1 millón de peticiones al mes, pasar de JSON a **TOON podría ahorrarte unos 55 millones de tokens**. Si usas un modelo premium como Claude 3 Opus (que cuesta aprox. 15 $/1M tokens), eso son más de **825 $ al mes que te dejas de gastar**. (Aquí los devs haciendo un poco de cherry picking pero bueno… entendemos el concepto).

La idea es brutal. Peeero, como siempre, no es oro todo lo que reluce. El ahorro de tokens es solo una parte de la ecuación, la otra es si el modelo entiende bien ese formato.

Hoy te cuento qué es TOON, cómo usarlo e integrarlo en tu flujo de trabajo y por qué todavía hay que mirarlo con cierta cautela.

TOON vs JSON: La batalla por el ahorro de tokens en LLMs

Cualquiera que haya trabajado con las APIs de OpenAI, Gemini, Claude… sabe que el JSON, aunque es un estándar, es muy verboso.

Cada vez que envías un array de objetos, **repites las mismas claves** ("id", "name", "role") una y otra vez. Todos esos corchetes ([]), llaves ({}), comas y comillas dobles **suman tokens**. Y **cada token cuesta dinero**.

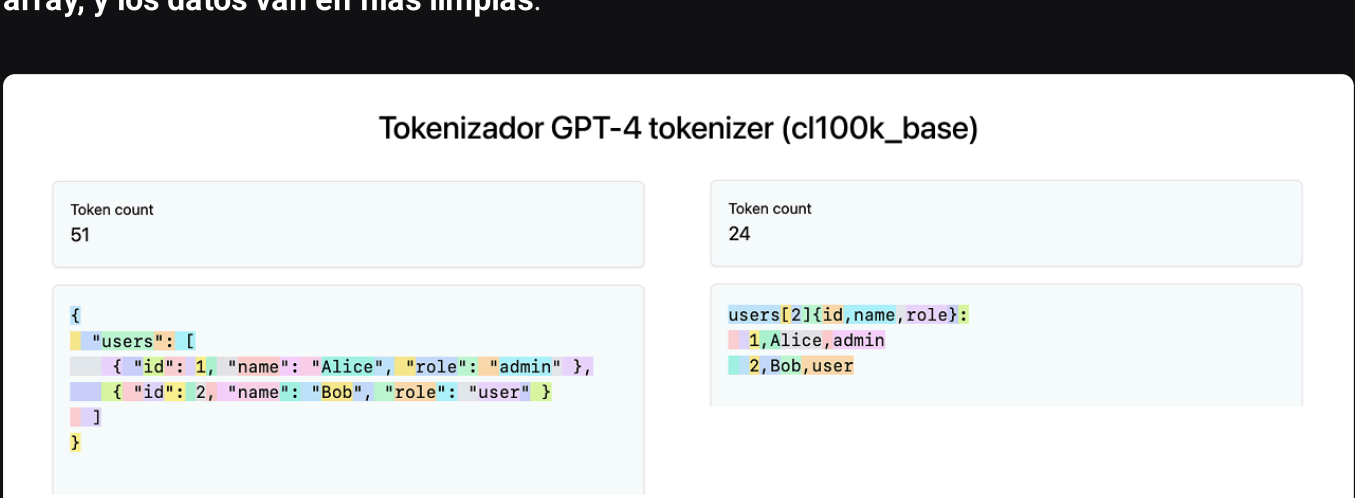
Mira este ejemplo de JSON:

```
{"users":[{"id":1,"name":"Alice","role":"admin"},{"id":2,"name":"Bob","role":"user
```

YAML mejora un poco al quitar llaves y comillas, pero TOON va un paso más allá, inspirándose en la estructura de CSV para los arrays. Equivalencia en TOON:

```
users[2]{id,name,role}:
1,Alice,admin
2,Bob,user
```

Para que veas la diferencia real de tokens (calculada con tiktokenizer), mira esta comparativa. Ya puedes ver por dónde van los tiros: **las claves se declaran una sola vez en la cabecera del array, y los datos van en filas limpias**.

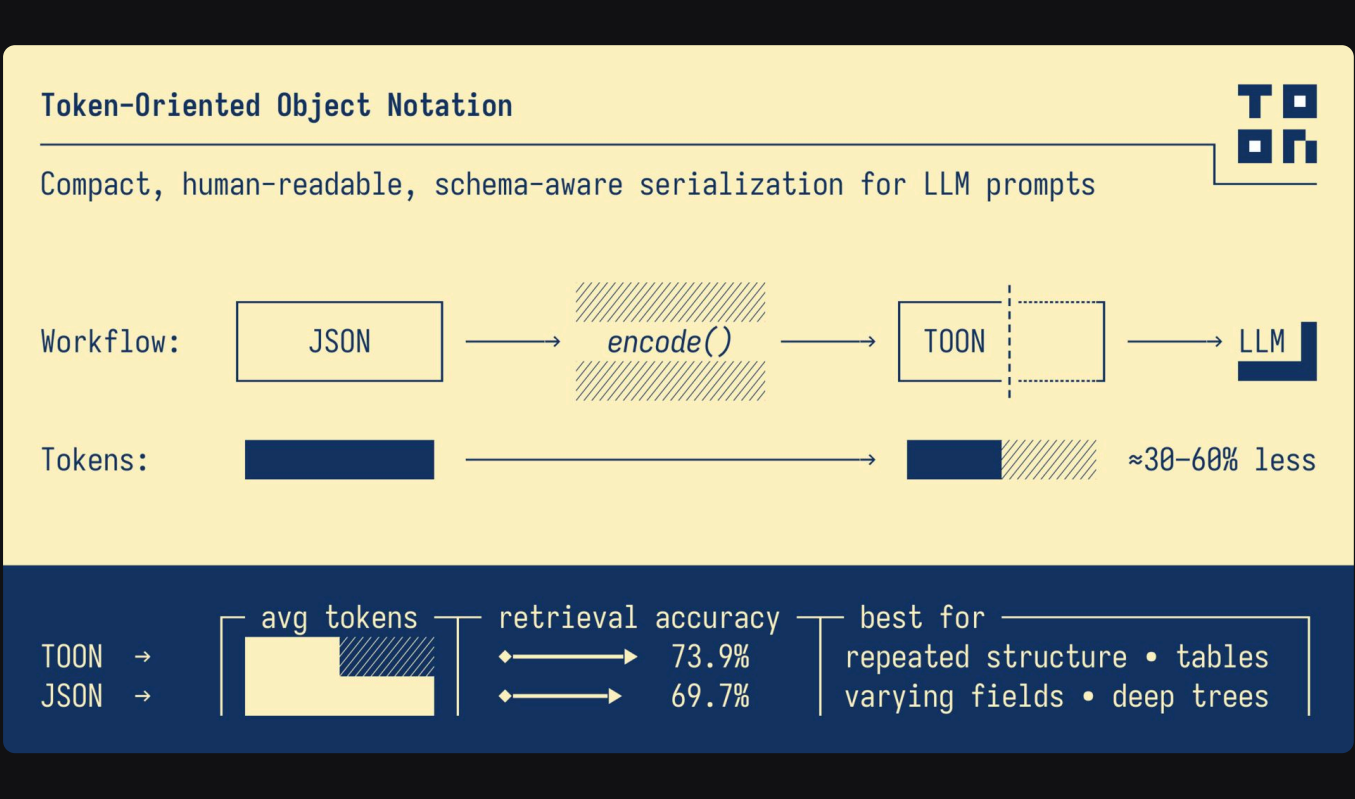


Comparativa de Tokens JSON vs TOON con el tokenizador de GPT-4

TOON explicado: ¿Qué es Token-Oriented Object Notation (TOON)?

TOON (Token-Oriented Object Notation) es un **formato de serialización compacto y legible por humanos**, diseñado específicamente para pasar datos estructurados a LLMs **usando muchos menos tokens**.

Es importante entender que **no busca reemplazar a JSON** en tus APIs o bases de datos. Su objetivo es ser una capa de traducción: usas JSON en tu lógica de backend, **lo conviertes a TOON justo antes de enviarlo al LLM**, y te ahorras unos tokens por el camino.



Token-Oriented Object Notation (TOON)

Uno de sus puntos fuertes son los arrays de objetos uniformes: muchas filas con la misma estructura. **Para datos muy anidados o no uniformes, el propio JSON compacto puede ser más eficiente**.

El formato tiene una especificación completa (v2.0) y se basa en dos ideas:

- Estructura por **indentación** (como YAML) para objetos anidados.
- Formato **tabular** (como CSV) para los arrays de objetos, declarando las claves {id,name,role} una vez.

Además, incluye *guardrails* para el LLM, como `users[2]`, que le dice explícitamente al modelo cuántos elementos esperar, ayudando a validar que los datos no estén truncados.

```
[2]:
- name: Alice Smith
id: 101
skills[3]: JavaScript,CSS,HTML
- name: Bob Johnson
id: 102
skills[3]: Python,SQL,Java
```

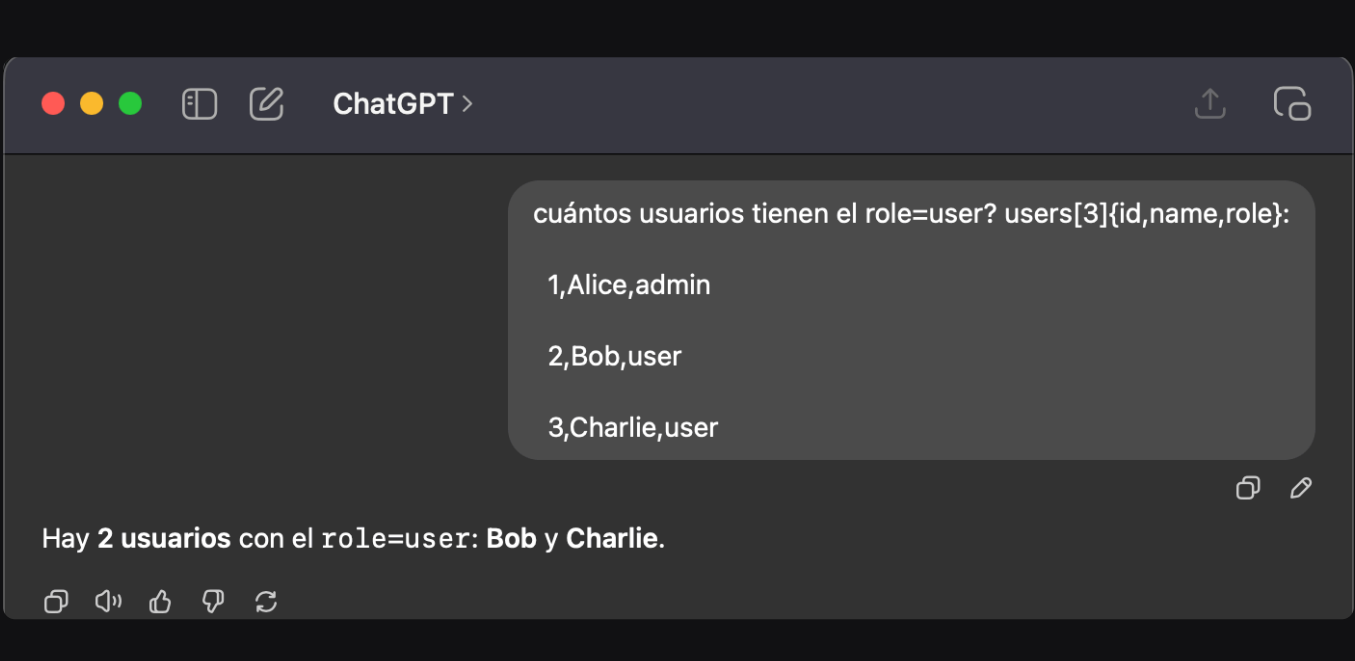
Primeros pasos con TOON: Cómo convertir JSON a TOON (y viceversa)

Lo mejor de TOON es que **NO tienes que explicarle el formato al LLM**. Los modelos lo pillan rápido al parecerse a YAML y CSV.

Para empezar simplemente dale tus datos en TOON al LLM. Aquí tienes un **ejemplo** mínimo en formato TOON para que pruebes:

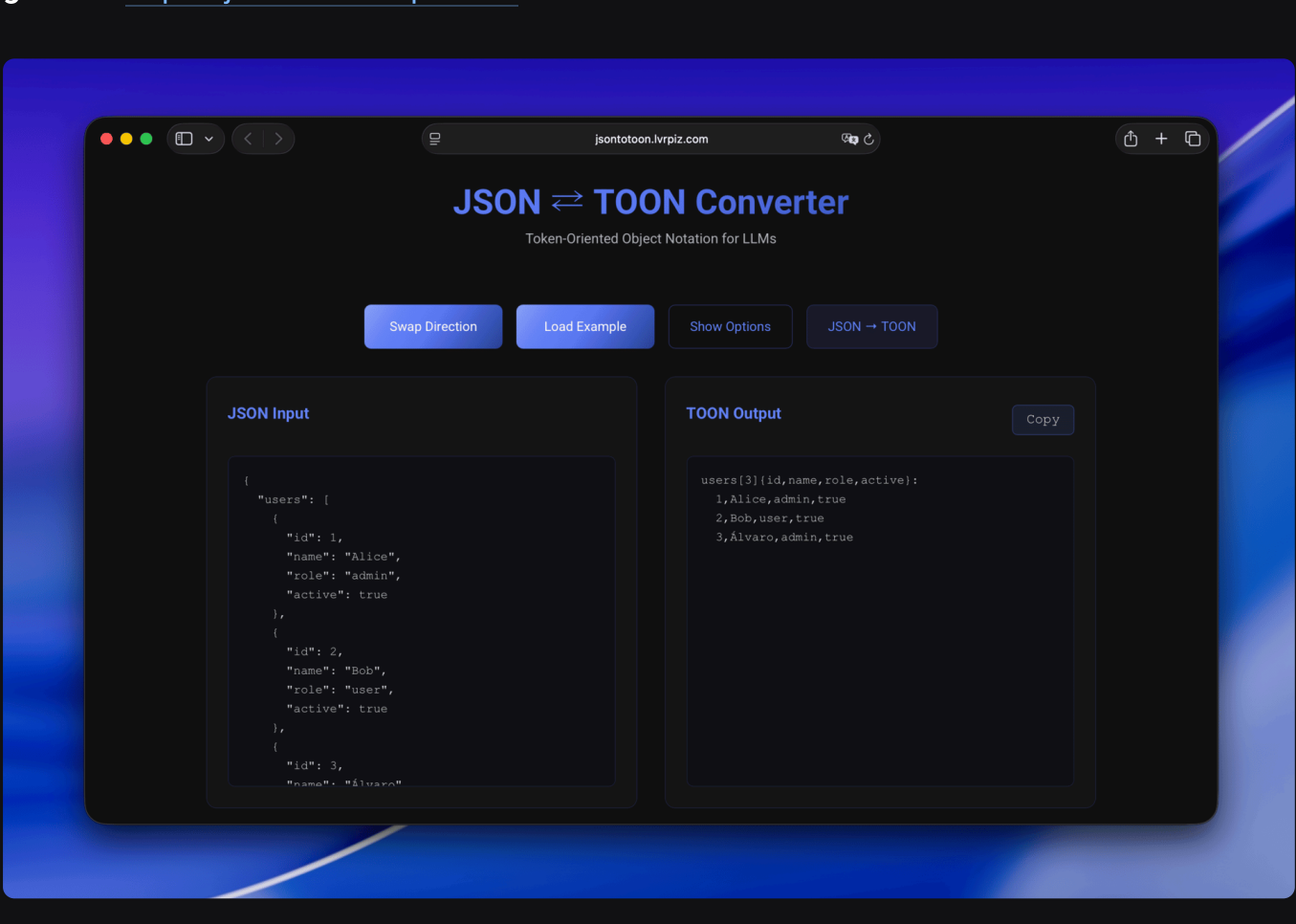
```
users[3]{id,name,role}:

1,Alice,admin

2,Bob,user

3,Charlie,user
```

Si le pides a ChatGPT que analice los usuarios con el rol *user* obtenemos lo siguiente:



Como ves, los LLMs no tienen gran problema entendiendo TOON sencillos (luego analizaremos benchmarks con ejemplos complejos). Ahora te preguntarás: Álvaro, ¿cómo paso mi JSON a TOON?

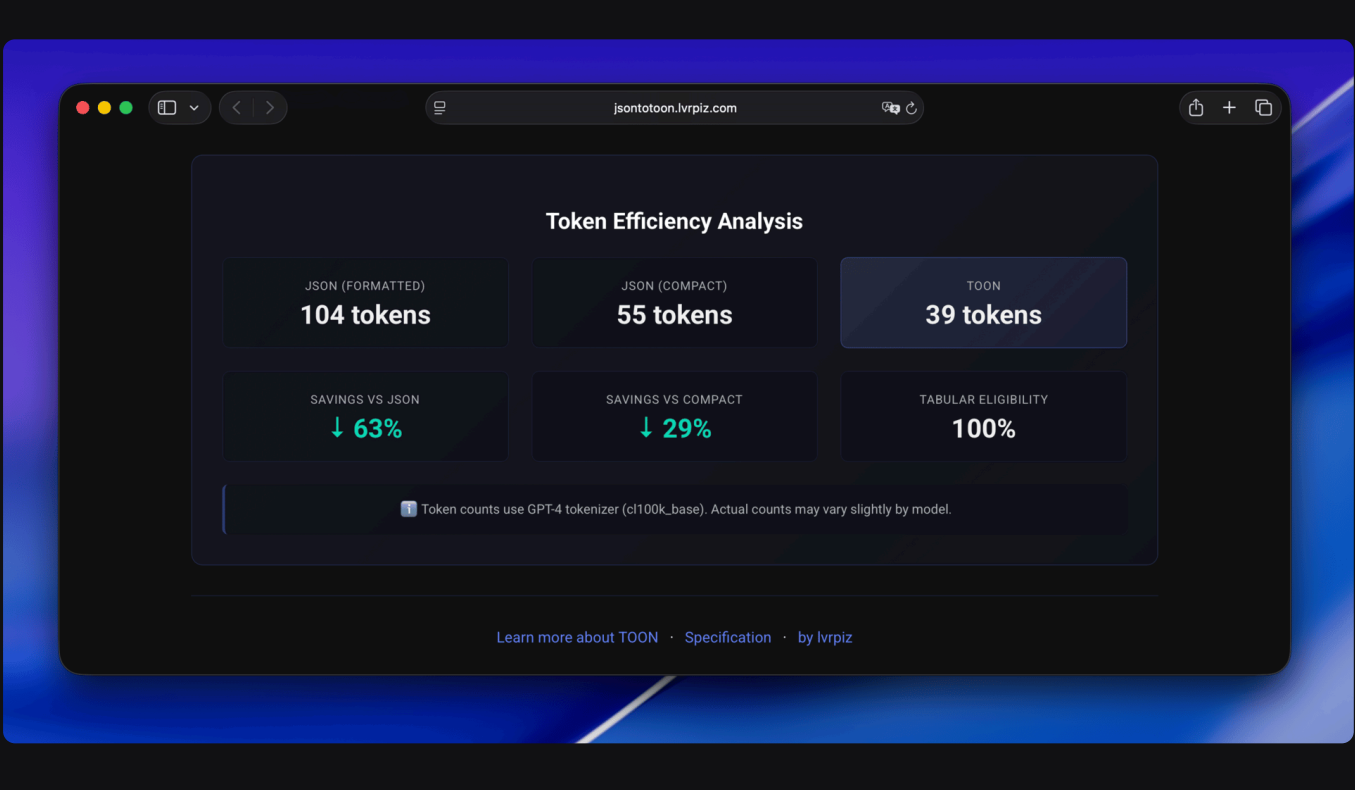
Para facilitar las pruebas y ver el ahorro real, he desarrollado una **pequeña herramienta online gratuita**: https://jsontotoon.lvrpiz.com



Conversor de JSON a TOON bidireccional

Es un **conversor bidireccional**: pegas tu JSON y te da el TOON, y viceversa. Pero lo más útil es que he implementado un **análisis de tokens** usando el tokenizador cl100k_base (el de GPT-4), algo que no he visto en otros conversores y que nos da una **idea del ahorro que vamos a tener.**

Te dice exactamente cuántos tokens te ahorras frente a JSON formateado y compacto, y qué porcentaje de tus datos es elegible para el formato tabular.



Análisis de eficiencia de Tokens TOON vs JSON

Ahora bien, esta herramienta está muy bien para jugar, probar combinaciones, ver el **impacto en Tokens** de cambiar a TOON y tener una estimación del ahorro pero ChatGPT no nos cobra más si interactuamos con su aplicación, nos cobra por Tokens en su API. ¿Cómo integramos el formato TOON en nuestras llamadas a la API? 👉 `@toon-format/toon`

Cómo instalar y usar la librería TOON en Node.js (npm/pnpm): `@toon-format/toon`

Si quieres integrarlo en tu backend (Node.js), la instalación es estándar:

```
# npm
npm install @toon-format/toon

# pnpm
pnpm add @toon-format/toon
```

Y el uso básico es muy directo. Usas la **función encode** para convertir tu objeto JSON a un string TOON:

```
import { encode } from '@toon-format/toon'

const data = {
users: [
{ id: 1, name: 'Alice', role: 'admin' },
{ id: 2, name: 'Bob', role: 'user' }
]
}

const toonString = encode(data)
console.log(toonString)

// users[2]{id,name,role}:
//   1,Alice,admin
//   2,Bob,user
```

Con esto ya puedes pasar tus JSON a formato TOON antes de volcarlos al LLM para ahorrarte unos tokens en cada llamada. También tienes la **función decode**(toonString) para hacer el camino inverso.

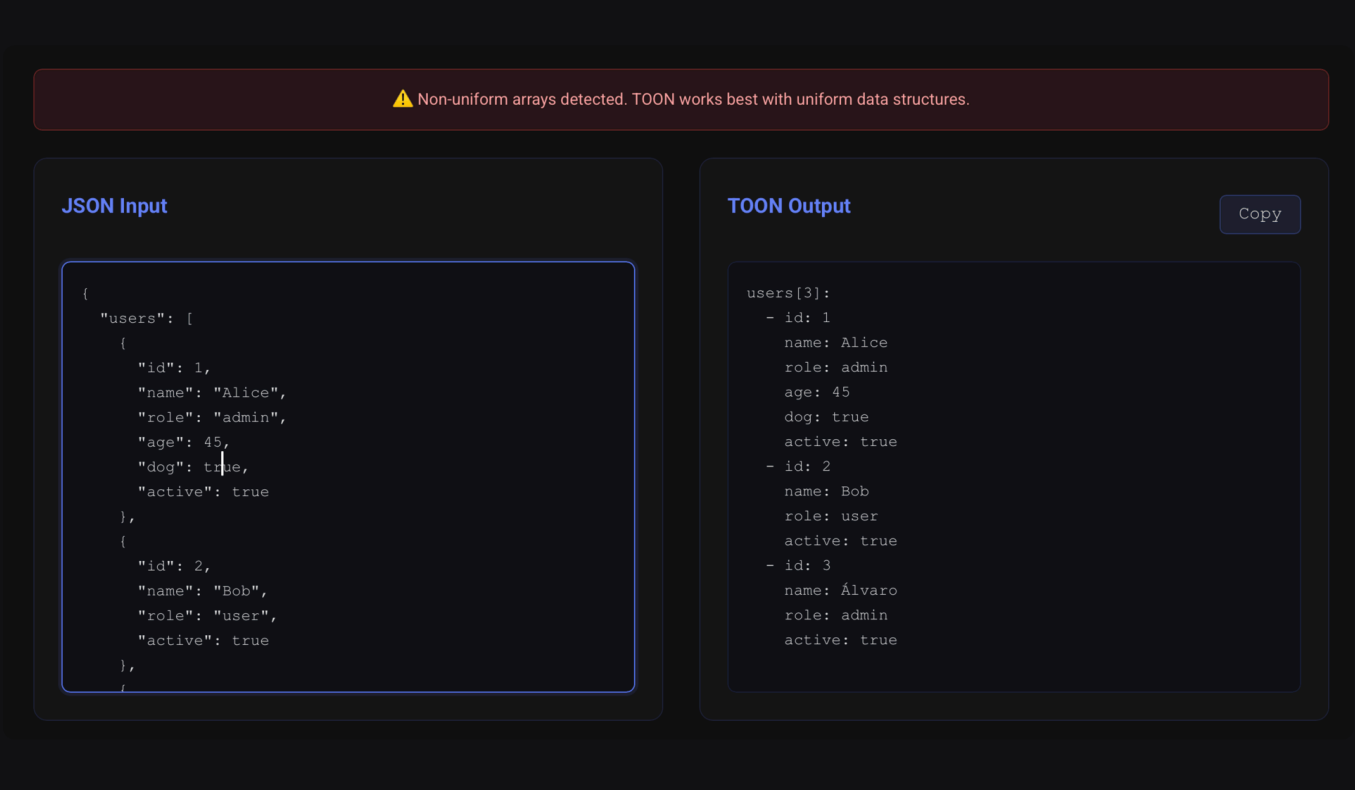
Limitaciones de TOON: ¿Volvemos a YAML o JSON?

Aquí es donde toca analizar nuestros datos y decidir. **TOON es excelente con arrays de objetos que sean uniformes**. Si tus datos no encajan ahí, el ahorro desaparece e incluso puede ser contraproducente.

No deberías usar TOON si:

- Tus **datos son muy anidados o no uniformes**: Si tienes un JSON de configuración complejo, con muchos niveles y objetos que no comparten claves, el JSON compacto (minificado) probablemente usará menos tokens.
- Tienes **arrays semi-uniformes**: Si en un array algunos objetos tienen 3 claves y otros 5, TOON no puede usar el modo tabular y pasa a un modo "lista" (con guiones, como YAML) que es menos eficiente.
- Son **datos puramente tabulares y planos**: Si solo tienes una tabla simple, un CSV de toda la vida sigue siendo mejor en cuanto a tokens. Aquí TOON añade un pequeño overhead (un 5-10% más que CSV) a cambio de dar más estructura (el [N] y las claves {}), lo cual ayuda al LLM a no equivocarse.

Si usas la herramienta que te comentaba previamente y el formato de tus datos no es óptimo para TOON te saldrá un warning y lo verás reflejado a mayores en el análisis de Tokens.



Aviso de JSON no uniforme detectado en la herramienta de JSONtoTOON

Análisis de benchmarks: ¿Entienden los LLMs el formato TOON?

De nada sirve ahorrar un 40% en tokens si el LLM se confunde y te da una respuesta incorrecta. Aquí es donde las cosas se ponen interesantes. Vamos a analizar los benchmarks disponibles:

Los benchmarks oficiales de TOON (enlace a su GitHub) son muy positivos. En sus pruebas de recuperación de datos (con modelos como GPT-5-nano y Gemini-2.5-flash), TOON consigue una precisión media del 73.9%, **superando al JSON (69.7%) y usando un 39.6% menos de tokens**.

```
TOON           ████████████████████   26.9  │  73.9% acc  │  2,744 tokens
JSON compact   █████████████████░░░   22.9  │  70.7% acc  │  3,081 tokens
YAML           ██████████████░░░░░░   18.6  │  69.0% acc  │  3,719 tokens
JSON           ███████████░░░░░░░░░   15.3  │  69.7% acc  │  4,545 tokens
XML            ██████████░░░░░░░░░░   13.0  │  67.1% acc  │  5,167 tokens
```

Peeero, he encontrado un análisis de un tercero que pone esto en duda y obliga a mirar los datos con más calma.

En sus pruebas, la cosa cambia, y es crucial analizar el **trade-off entre precisión y tokens**.

Test con Datos Tabulares

A primera vista, si solo miras la columna de precisión, **TOON (47.5%) parece rendir peor que formatos que consumen más tokens como JSON (52.3%) o YAML (54.7%)**.

Formato	Accuracy	Intervalo de confianza del 95 %	Tokens
Markdown-KV	**60.7%**	57.6% – 63.7%	52,104
XML	56.0%	52.9% – 59.0%	**76,114**
YAML	54.7%	51.6% – 57.8%	55,395
HTML	53.6%	50.5% – 56.7%	75,204
JSON	52.3%	49.2% – 55.4%	66,396
TOON	**47.5%**	**44.4% – 50.6%**	**21,518**
JSONL	45.0%	41.9% – 48.1%	54,407
CSV	**44.3%**	41.2% – 47.4%	**19,524**

Pero, ahora mira la columna de Tokens.

- JSON: 52.3% de precisión costando 66,396 tokens.
- TOON: 47.5% de precisión costando 21,518 tokens.

Aquí se ve el **trade-off real**: **TOON usa un 68% menos de tokens que JSON**. La pregunta que debes hacerte es: **¿estoy dispuesto a asumir una caída de ~5 puntos en la precisión a cambio de pagar casi 3 veces menos?**

Para datos tabulares, **TOON compite en eficiencia con CSV** (que saca una precisión y un coste similar), pero queda claro que los formatos más verbosos como JSON o YAML obtienen algo más de precisión, **pagando un precio mucho más alto**.

Test con Datos Anidados

Aquí es donde TOON sale peor parado y se confirma lo que te comentaba en la sección anterior.

Formato	Accuracy	Intervalo de confianza del 95 %	Tokens
YAML	**62.1%**	59.1%, 65.1%	42,477
Markdown	54.3%	51.2%, 57.4%	**38,357**
JSON	50.3%	47.2%, 53.4%	657,933
XML	44.4%	41.3%, 47.5%	68,804
TOON	**43.1%**	**40.0%, 46.2%**	**45,436**

En este escenario de datos anidados, **TOON no solo fue el formato con la peor precisión (43.1%), sino que además usó más tokens que YAML (45k vs 42k)**, que encima le sacó 19 puntos de precisión.

El **punto fuerte de TOON son los datos tabulares uniformes**, pero para estructuras anidadas, YAML parece una opción mucho más equilibrada.

Por resumir… ¿Qué significa todo esto?

Todavía es **pronto para sacar conclusiones**. Los benchmarks son contradictorios y dependen mucho del modelo y del tipo de datos que usemos.

Es muy probable que formatos como JSON o YAML funcionen mejor (aunque cuesten más) simplemente **porque los LLMs han sido entrenados con trillones de ejemplos de ellos**.

TOON es nuevo y no está en los datos de entrenamiento. Es posible que su rendimiento mejore a medida que los modelos se reentrenen y "aprendan" el formato.

Mi recomendación: si trabajas con arrays de objetos muy grandes y uniformes, donde el coste de los tokens es un problema real, dale una oportunidad. Pasa tus datos por la herramienta de conversión, mira el ahorro potencial y haz pruebas A/B de precisión con tus propios prompts.

Para datos anidados o críticos donde la precisión es lo único que importa, de momento me quedaría con JSON compacto o YAML.

Hasta el próximo miércoles,

Álvaro