Análisis Tripartito de la Serialización de Datos: JSON vs. TON (TL-B) vs. TOON

I. Introducción a las Arquitecturas de Serialización de Datos

1.1. El Rol Crítico de la Serialización en el Diseño de Sistemas Distribuidos

La serialización de datos constituye un proceso fundamental en la arquitectura de sistemas distribuidos y descentralizados, refiriéndose a la transformación de estructuras de objetos complejas en un formato adecuado para su almacenamiento, transmisión o verificación. La elección de un formato de serialización determina directamente la eficiencia, seguridad y la interoperabilidad de los límites del sistema. Históricamente, esta selección ha implicado un compromiso entre tres imperativos arquitectónicos: la **universalidad** (la facilidad de uso en diversos entornos), la **eficiencia** (la compacidad y velocidad de análisis) y la **seguridad de tipos** (la aplicación estricta de esquemas y el determinismo).

El análisis de las tendencias actuales en Web3 (Blockchain) y Web3.5 (Agentes de IA) revela que la arquitectura moderna está experimentando un cambio estratégico. Mientras que en la era Web2 se priorizó la universalidad y la legibilidad humana, ejemplificadas por JSON, los nuevos dominios introducen restricciones no negociables, como la verificabilidad criptográfica y la economía de tokens. Este entorno obligó a la creación de formatos altamente especializados, como el TL-B y el TOON, señalando una transición del paradigma de "un formato para todos" hacia protocolos optimizados para dominios específicos.[1] Los arquitectos contemporáneos deben, por lo tanto, gestionar entornos multiprotocolo en lugar de estandarizar una única solución.

1.2. Deconstrucción de la Consulta: Resolución de la Ambigüedad del Término "TOn"

La investigación sobre la notación "TOn" reveló una ambigüedad terminológica crucial. El término se refiere a dos tecnologías de serialización distintas, cada una diseñada para superar limitaciones específicas de JSON en contextos avanzados:

1. **TON / TL-B (Type Language for Binary):** Es el protocolo de serialización binaria central de The Open Network (TON). Su enfoque es resolver los problemas de determinismo y compacidad binaria requeridos por los libros mayores

descentralizados.[1]

2. **TOON (Token-Oriented Object Notation):** Es un formato textual diseñado específicamente para entornos de Inteligencia Artificial (IA), cuya finalidad es mitigar los costos de tokens y mejorar la precisión de los Grandes Modelos de Lenguaje (LLMs) al reducir la verbosidad de JSON.[3]

En consecuencia, el presente informe utiliza JSON como línea de base universal y lo contrasta con el protocolo binario de alta seguridad (TL-B) y el formato textual de alta eficiencia (TOON).

II. JSON: El Estándar Ubicuo y sus Limitaciones Arquitectónicas

2.1. Fundamentos Estructurales, Sintaxis e Interoperabilidad

JSON (JavaScript Object Notation) es el formato de intercambio de datos dominante en el mundo debido a su sencillez sintáctica y su origen en JavaScript, lo que lo hace intrínsecamente compatible con aplicaciones web y servidores.[4] Se basa en dos estructuras de datos universales: colecciones de pares nombre-valor (objetos, delimitados por llaves {}) y listas ordenadas de valores (matrices o *arrays*, delimitados por corchetes ``).[5]

Su naturaleza agnóstica al lenguaje, utilizando convenciones familiares a lenguajes de la familia C (C++, Java, Python), cimentó su posición como el estándar primario para APIs RESTful y GraphQL.[5] La compatibilidad y la estructura ligera y legible para humanos han sido clave para su ubicuidad en el intercambio de datos electrónicos, incluidos los procesos de autenticación y manejo de datos sensibles.[6]

2.2. Debilidades Inherentes en la Aplicación de Tipos y el Rendimiento

A pesar de su éxito en la interoperabilidad, JSON exhibe debilidades estructurales significativas en entornos que exigen alta precisión o rendimiento.

Primero, JSON es un **formato débilmente tipado**, ya que solo soporta seis tipos primitivos (cadena, número, booleano, nulo, objeto, matriz).[5] Carece de diferenciación explícita para tipos complejos esenciales en sistemas financieros o criptográficos, como fechas y horas específicas, o enteros de 64 bits o mayores. Esto obliga a codificar dichos tipos complejos como tipos primitivos, generalmente cadenas, lo que requiere que las aplicaciones implementen lógica de conversión personalizada (como usar JsonConverter en C# para manejar identificadores fuertemente tipados) o recurran a mecanismos de validación de esquemas externos.[7]

Segundo, su naturaleza textual introduce problemas de **verbosidad y sobrecarga de análisis**. JSON depende de delimitadores repetitivos ({} y ``), lo que obliga a los analizadores sintácticos (*parsers*) a leer y cotejar la totalidad de la estructura de texto antes de poder

comprender la jerarquía completa del objeto.[9] Esta dependencia del análisis carácter por carácter impide la precalculación de desplazamientos de datos, lo que es común en formatos binarios o bases de datos estructuradas. Por lo tanto, JSON no está optimizado para el procesamiento de grandes volúmenes de datos, sino más bien para el intercambio web de archivos pequeños.[9]

Existe un equilibrio fundamental en la arquitectura de JSON: su diseño prioriza la legibilidad humana y la simplicidad, pero esto se logra a costa de la seguridad intrínseca. JSON no incorpora características de seguridad integradas y debe depender completamente de mecanismos de capa de aplicación externos, como la transmisión segura (HTTPS) y la validación rigurosa de la carga útil.[6] En entornos de alto riesgo, esta dependencia significa que JSON requiere una superficie de ataque más amplia que formatos con seguridad de tipos integrada, lo que resulta en procesos de validación costosos y propensos a errores.

La siguiente tabla resume las inconsistencias de tipos primitivos de JSON:
JSON Primitive Type Consistency

Tipo JSON	Debilidad	Implicación Arquitectónica
Cadena (String)	Ambigüedad (puede representar UUIDs, Fechas, Hashes)	Requiere validación de esquema externa para determinar el significado [7]
Número (Number)	Falta de garantías de precisión (flotantes estándar, enteros hasta 53 bits)	No es apto para representaciones enteras criptográficas o financieras (ej. claves de 256 bits)
Null	Ambigüedad semántica (Ausente, Desconocido, Explícitamente Nulo)	Exige un manejo complejo durante la deserialización para distinguir la intención [10]

III. TON / TL-B: Protocolo Binario Determinista para el Estado Blockchain

3.1. Orígenes y Mandato Arquitectónico de The Open Network (TON)

The Open Network (TON), una red descentralizada de múltiples cadenas altamente escalable, surgió originalmente de Telegram.[11] El imperativo fundamental de una blockchain de alto rendimiento es garantizar la **ejecución determinista del estado** y la **máxima eficiencia** para sus contratos inteligentes. Los formatos textuales tradicionales como JSON son inherentemente no deterministas y demasiado verbosos para la ejecución de transacciones dentro de la máquina virtual de alta velocidad de TON (TVM).

Para cumplir con este requisito, TON utiliza TL-B (Type Language for Binary) como su

columna vertebral de serialización. Los contratos inteligentes de TON se escriben en FunC, un lenguaje de tipado estático optimizado para la seguridad y la eficiencia.[12] TL-B es el protocolo que traduce estos tipos fuertes en datos binarios verificables, haciendo cumplir el uso correcto de los tipos de datos.[13]

3.2. Mecánica Central de TL-B (Type Language for Binary)

El Fundamento Estructural: La Estructura de Celda

La unidad de datos fundamental e inmutable en la blockchain de TON es la **Celda** (*Cell*).[2] Las celdas se utilizan para almacenar código de contrato inteligente, datos persistentes y bloques enteros.[2] Cada celda está diseñada para almacenar hasta 1023 bits de datos y puede mantener hasta 4 referencias a otras celdas.
La serialización de este sistema se logra a través de un **Bag of Cells (BoC)**, un formato que codifica las celdas interconectadas en matrices de bytes. Este enfoque permite una arquitectura de datos modular, flexible y, crucialmente, *hashable*. Las celdas se comparan mediante sus *hashes* para eliminar duplicados, lo que es esencial para la integridad de los datos en la blockchain.[2]

Tipado Explícito Mediante Etiquetas de Constructor

TL-B impone un tipado fuerte utilizando **etiquetas de constructor** explícitas que se colocan al inicio del flujo de datos binarios.[1] Esto contrasta directamente con la ambigüedad de JSON.
La definición del constructor en TL-B especifica la etiqueta de bitstring utilizada para la codificación. Por ejemplo, bool_true$1 = Bool indica que el tipo *Bool* se serializa con la etiqueta binaria $1 (un bit de valor 1).[1] Durante el proceso de deserialización, el analizador TL-B lee esta etiqueta (que puede ser un valor hexadecimal de 32 bits, como #3f5476ca, o una secuencia binaria) para **determinar de manera inequívoca el tipo exacto y la estructura** del bitstring siguiente.[1] Si no se proporciona una etiqueta, se calcula una por defecto de 32 bits aplicando el algoritmo CRC32 a la ecuación del constructor. Este mecanismo de etiquetado explícito elimina la ambigüedad de tipos, que es un problema inherente en JSON, y es vital para la seguridad.
El diseño de la Celda impone que los datos se almacenen en bloques finitos y pequeños, vinculados por referencias explícitas. Esta arquitectura fuerza la modularidad y hace que los datos sean inherentemente verificables a través del *hashing* criptográfico (hashing de Bag of Cells). El protocolo TL-B, con su estructura binaria y etiquetas de tipo explícitas, actúa como un **protocolo de seguridad obligatorio** en el nivel de serialización. Su implementación es una defensa fundamental contra el no-determinismo de la ejecución y mitiga vulnerabilidades comunes en contratos inteligentes, como el uso incorrecto de tipos de datos.[13]

3.3. El Sistema de Tipos Especializados de TON

El sistema de tipos de TON está específicamente adaptado a las necesidades de la TVM [15]:

- **Int:** Un entero con signo de 257 bits. Esta longitud no es arbitraria; acomoda claves criptográficas y *hashes* estándar de 256 bits, más un bit de signo. Las comprobaciones de desbordamiento están activadas por defecto y activan una excepción si se superan, un requisito de seguridad crítica.[15]
- **Cell:** El tipo fundamental de celda TVM, utilizado para el almacenamiento persistente e inmutable.[15]
- **Slice:** Una vista de solo lectura de una celda, crucial para el acceso secuencial eficiente a los datos y referencias de la celda durante la ejecución.[15]
- **Builder:** Una estructura mutable utilizada para construir nuevas celdas antes de su finalización y almacenamiento.[15]

3.4. Ecosistema y Herramientas para la Serialización TL-B

Debido a la complejidad intrínseca del manejo de Celdas y TL-B, la adopción se basa en SDKs especializados en lugar de analizadores JSON universales.[16] Existen bibliotecas fundamentales para los principales ecosistemas de desarrollo, incluyendo Python (tonutils, pytoniq), Java (ton4j), y TypeScript (tonweb).[16]

Además, el proceso de desarrollo se apoya en herramientas de **generación de código**. Paquetes como tlb-codegen generan código TypeScript para serializar y deserializar estructuras basándose en un esquema TL-B proporcionado.[17] Esta herramienta es esencial, ya que TL-B requiere una definición similar a la de un compilador para interpretar correctamente el flujo binario, asegurando la adherencia al tipo. Por ejemplo, bibliotecas como TonLib.NET en C# proporcionan *wrappers* que interactúan directamente con los *lite servers* de TON a través del protocolo ADNL, permitiendo a los desarrolladores trabajar con *Bags of Cells* (BoCs) y *Slices*.[18]

IV. TOON: Optimización Orientada a Tokens para Flujos de Trabajo de IA

4.1. Filosofía de Diseño y Contexto LLM

TOON (Token-Oriented Object Notation) es un formato de datos textual diseñado específicamente para la optimización de entrada y salida con Grandes Modelos de Lenguaje (LLMs).[3] Su diseño responde a una motivación puramente económica: la interacción con LLMs es sensible al costo de los tokens. El objetivo principal de TOON es maximizar la cantidad de datos significativos transferidos por token, lo que resulta en costos de API reducidos y una

ventana de contexto efectiva ampliada.[3]
TOON logra esto mediante una fusión sintáctica. Combina la legibilidad basada en la indentación de YAML con la claridad tabular de CSV, al tiempo que elimina la puntuación redundante y las claves repetidas que caracterizan a JSON.[3]

4.2. Métricas de Eficiencia y Ganancias Arquitectónicas

Los *benchmarks* de rendimiento han demostrado que TOON reduce el número de tokens requeridos entre un **30% y un 60%** en comparación con estructuras JSON equivalentes.[3]
Este incremento de eficiencia se produce principalmente en escenarios de **datos tabulares y uniformes** (matrices uniformes de objetos). En lugar de repetir las claves para cada objeto, TOON declara la longitud de la matriz y los nombres de los campos (encabezados) una sola vez, para luego transmitir las filas de datos de manera eficiente.[3] Por ejemplo, una lista de usuarios en JSON requiere la repetición de las claves id, name, y role para cada entrada, mientras que TOON las define una vez en el encabezado.[3]
Además de la eficiencia económica, las pruebas de precisión indican que TOON mejora la precisión de la recuperación de datos de los LLMs en varios puntos porcentuales en comparación con JSON.[3] La estructura tabular explícita de TOON funciona como "barandillas" estructurales para el modelo, minimizando la probabilidad de que el modelo alucine o malinterprete delimitadores faltantes o límites de matriz, un problema común con la verbosidad de JSON.

4.3. Restricciones Estructurales y Compromisos

TOON es una herramienta de nicho, altamente especializada para **matrices de datos tabulares y uniformes** (por ejemplo, listas de comandos de agentes, registros de usuarios o elementos de inventario).[3]
Sin embargo, su eficiencia se disipa cuando las estructuras de datos son profundamente anidadas o altamente irregulares. En estos escenarios, la dependencia de la indentación y los encabezados explícitos puede volverse más prolija y propensa a errores que el uso claro de llaves y corchetes como delimitadores en JSON.[3] Por esta razón, JSON sigue siendo superior para la representación de datos complejos, no estructurados o jerárquicos profundos.
La ventaja principal de TOON no es la velocidad de análisis técnico (como el enfoque binario de TL-B), sino la **eficiencia semántica** dentro del contexto del LLM. Al minimizar el ruido sintáctico, maximiza la proporción de tokens de contenido frente a tokens de estructura. Por lo tanto, la adopción de TOON está impulsada por la **economía del costo de API** y la **confiabilidad de los agentes**, en lugar de la optimización tradicional de CPU o red.

V. Análisis Comparativo Multicriterio de Paradigmas de Serialización

Este análisis formaliza los compromisos entre los tres formatos a lo largo de vectores de ingeniería clave.

5.1. Seguridad de Tipos y Aplicación de Esquemas

TON/TL-B ofrece la seguridad de tipos más alta. La información de tipo es obligatoria e integrada directamente en el flujo binario mediante etiquetas de constructor.[1] Esto es exigido por el lenguaje FunC en tiempo de compilación [12], proporcionando el máximo determinismo y minimizando la ambigüedad en tiempo de ejecución.
JSON posee la seguridad de tipos más débil, dependiendo enteramente de esquemas externos y la disciplina del desarrollador. Sus tipos primitivos son inherentemente ambiguos.[7]
TOON se sitúa en un nivel moderado. Proporciona un contexto de tipo explícito a través de definiciones de encabezados, lo que ofrece una claridad estructural superior a la de JSON para datos columnares, pero carece de la verificación de tipos binarios profunda y obligatoria del TL-B.

5.2. Eficiencia de Serialización: Codificación Binaria vs. Textual

TON/TL-B exhibe la máxima compacidad y la mayor velocidad de análisis. Su codificación binaria minimiza el espacio en memoria y el tamaño de transferencia para la persistencia *on-chain*.[2] El análisis está diseñado para la TVM, operando en *bitstreams* y referencias explícitas para una sobrecarga mínima.
JSON presenta la eficiencia más baja. La codificación textual verbosa resulta en archivos más grandes. El análisis requiere un tiempo de CPU significativo debido a la interpretación carácter por carácter y la coincidencia de estructuras, lo que lo hace lento para grandes volúmenes.[9]
TOON es el más eficiente textualmente. Logra una reducción sustancial de tokens, lo que lo hace óptimo para la E/S de LLM, donde el número de tokens es la restricción de recursos primordial.[3]

5.3. Seguridad y Ejecución Determinista

TON/TL-B tiene el determinismo como núcleo de seguridad. Las etiquetas de constructor garantizan que el mismo flujo binario siempre se deserialice en la misma estructura, lo cual es innegociable para el consenso blockchain y la prevención de vulnerabilidades de contratos inteligentes.[1]
JSON externaliza la seguridad. Carece de características de seguridad inherentes, dependiendo de protocolos externos (TLS) y validación rigurosa de entradas para evitar ataques.[6]
TOON no proporciona capacidades de validación criptográfica o seguridad binaria de bajo nivel, aunque su estructura explícita mejora la confiabilidad del *parsing* en el contexto del

agente de IA.

VI. Recomendaciones Arquitectónicas Estratégicas

La selección del formato de serialización debe estar guiada por el conjunto de restricciones arquitectónicas más crítico para el dominio de aplicación.

6.1. Cuándo Mandatar TON / TL-B: Determinismo e Integridad On-Chain

TL-B debe ser el mandato arquitectónico siempre que la **integridad de los datos de estado deba ser verificable criptográficamente, inmutable y ejecutada de forma determinista**.[2]
Esto incluye:

- El almacenamiento de datos de contratos inteligentes y la lógica dentro de la Máquina Virtual de TON.
- La comunicación entre contratos (cuerpos de mensajes).[19]
- La serialización de encabezados de bloques y transacciones.

La adopción de TL-B requiere aceptar la complejidad inicial y la dependencia de SDKs especializados (como tonutils o TonLib.NET) [17], un costo necesario para asegurar el rendimiento y la seguridad al nivel de la blockchain.

6.2. Cuándo Utilizar TOON: Optimización de Agentes LLM

TOON es la opción óptima cuando la **restricción operativa primaria es la reducción del costo de API y la optimización del contexto de *prompt*** en flujos de trabajo impulsados por IA.
Esto aplica a:

- La comunicación entre agentes donde los datos son predominantemente tabulares (listas de comandos, registros uniformes).[3]
- La generación de resultados estructurados por LLMs, donde la fiabilidad del análisis es crítica.[3]

Es fundamental limitar su uso estrictamente a estructuras de datos uniformes y poco profundas. Los datos irregulares o profundamente anidados deben volver a utilizar JSON para garantizar la claridad de los delimitadores.[3]

6.3. Cuándo JSON Sigue Siendo Dominante: Interoperabilidad

JSON sigue siendo la opción predeterminada indiscutible para el **intercambio universal de datos, la comunicación cliente-servidor y la integración de sistemas heterogéneos**.[4]
Sus casos de uso primarios incluyen:

- Respuestas tradicionales de APIs REST.

- Aplicaciones web del lado del cliente (debido a la compatibilidad nativa con JavaScript).[4]
- Archivos de configuración que priorizan la edición humana sobre la eficiencia de la máquina.[20]

6.4. Conclusiones y Matriz de Idoneidad Contextual

El éxito de JSON se debe a su adopción universal y madurez de herramientas. El futuro de TL-B está intrínsecamente ligado al crecimiento del ecosistema TON y al desarrollo de SDKs de alta calidad.[16] La viabilidad de TOON depende de su adopción por parte de los proveedores de LLM y de la existencia de herramientas sencillas de conversión a JSON para la integración *backend*.[10] La elección de TL-B o TOON representa una inversión en un ecosistema tecnológico específico, mientras que JSON representa una inversión en estabilidad y universalidad del mercado.
A medida que las arquitecturas se especializan, la necesidad de capas de conversión robustas y conscientes del tipo aumenta. Los sistemas deben integrar SDKs que puedan traducir salidas complejas (como un resultado TOON de un LLM) en el formato interno requerido (como la construcción de una Celda TL-B para su presentación *on-chain*).[17]

La siguiente matriz ofrece una guía para la selección estratégica del formato en función de los requisitos arquitectónicos clave:
Contextual Suitability Matrix for Serialization Format Selection

Requisito / Enfoque Arquitectónico	**JSON**	**TON / TL-B**	**TOON**
Interoperabilidad de API (Web2)	Excelente	Pobre (Requiere SDKs especializados) [16]	Moderada (Requiere conversión)
Validación Determinista de Datos	Baja (Depende de esquema externo)	Esencial (Núcleo del diseño) [1]	Baja
Optimización de Costos (Uso de Tokens LLM)	Baja (Altamente verboso)	N/A (Binario)	Alta (Ahorros del 30-60%) [3]
Criticidad de Seguridad (On-Chain)	Inadecuado	Obligatorio	Inadecuado
Estructura de Datos Preferida	Jerarquía Profunda / Sin Estructura	Grafo de Celdas (BoC) [2]	Datos Tabulares/Columnares [3]

Fuentes citadas

1. TL-B language | The Open Network - TON Docs, acceso: noviembre 3, 2025, https://docs.ton.org/v3/documentation/data-formats/tlb/overview

2. Overview | The Open Network - TON Docs, acceso: noviembre 3, 2025, https://docs.ton.org/v3/documentation/data-formats/cells/overview
3. JSON vs TOON: The New Structure for Large Language Models | by ..., acceso: noviembre 3, 2025, https://medium.com/@vandanbsheth9/json-vs-toon-the-new-structure-for-large-language-models-c135ee09b751
4. JSON - Wikipedia, acceso: noviembre 3, 2025, https://en.wikipedia.org/wiki/JSON
5. JSON, acceso: noviembre 3, 2025, https://www.json.org/
6. Is JSON Secure? Understanding Its Relationship with Authentication - DEV Community, acceso: noviembre 3, 2025, https://dev.to/ameliax/is-json-secure-understanding-its-relationship-with-authentication-d9p
7. JSON Data (Standard) - Oracle Help Center, acceso: noviembre 3, 2025, https://docs.oracle.com/en/database/oracle/oracle-database/21/adjsn/json-data.html
8. C# 9 records as strongly-typed ids - Part 3: JSON serialization, acceso: noviembre 3, 2025, https://thomaslevesque.com/2020/12/07/csharp-9-records-as-strongly-typed-ids-part-3-json-serialization/
9. speed of parsing json structures - Stack Overflow, acceso: noviembre 3, 2025, https://stackoverflow.com/questions/4435563/speed-of-parsing-json-structures
10. TOON – Token Oriented Object Notation | Hacker News, acceso: noviembre 3, 2025, https://news.ycombinator.com/item?id=45715632
11. TON Blockchain Explained: How It Works and Why It Matters - Nansen, acceso: noviembre 3, 2025, https://www.nansen.ai/post/ton-blockchain-explained
12. A Complete Guide to TON Blockchain Development - Features, Use Cases and Best Practices - A3Logics, acceso: noviembre 3, 2025, https://www.a3logics.com/blog/ton-blockchain-development/
13. TON smart contract security best practices | The Open Network, acceso: noviembre 3, 2025, https://docs.ton.org/v3/guidelines/smart-contracts/security/common-vulnerabilities
14. Serialization | The Open Network - TON Docs, acceso: noviembre 3, 2025, https://docs.ton.org/v3/documentation/data-formats/cells/serialization
15. Types | The Open Network - TON Docs, acceso: noviembre 3, 2025, https://docs.ton.org/v3/documentation/smart-contracts/func/docs/types
16. SDKs | The Open Network - TON Docs, acceso: noviembre 3, 2025, https://docs.ton.org/v3/guidelines/dapps/apis-sdks/sdk
17. Tools | The Open Network - TON Docs, acceso: noviembre 3, 2025, https://docs.ton.org/v3/documentation/data-formats/tlb/tools
18. justdmitry/TonLib.NET: tonlibjson wrapper for .NET - GitHub, acceso: noviembre 3, 2025, https://github.com/justdmitry/TonLib.NET
19. TON cookbook | The Open Network, acceso: noviembre 3, 2025, https://docs.ton.org/v3/guidelines/dapps/cookbook
20. JSON vs YAML vs TOML vs XML: Best Data Format in 2025 - DEV Community,

acceso: noviembre 3, 2025, https://dev.to/leapcell/json-vs-yaml-vs-toml-vs-xml-best-data-format-in-2025-5444
21. TON Ecosystem Support Just Got a Major Upgrade, acceso: noviembre 3, 2025, https://blog.ton.org/ton-ecosystem-support