👋 BYE !

JSON

for LLMs

reduced the token usage by 30-50%

Say Hi to TOON

for LLMs

JSON

```
{
  "products": [
    {
      "id": 1,
      "name": "Laptop",
      "price": 3999.90
    },
    {
      "id": 2,
      "name": "Mouse",
      "price": 149.90
    },
    {
      "id": 3,
      "name": "Headset",
      "price": 499.00
    }
  ]
}
```

✗

125 tokens

TOON

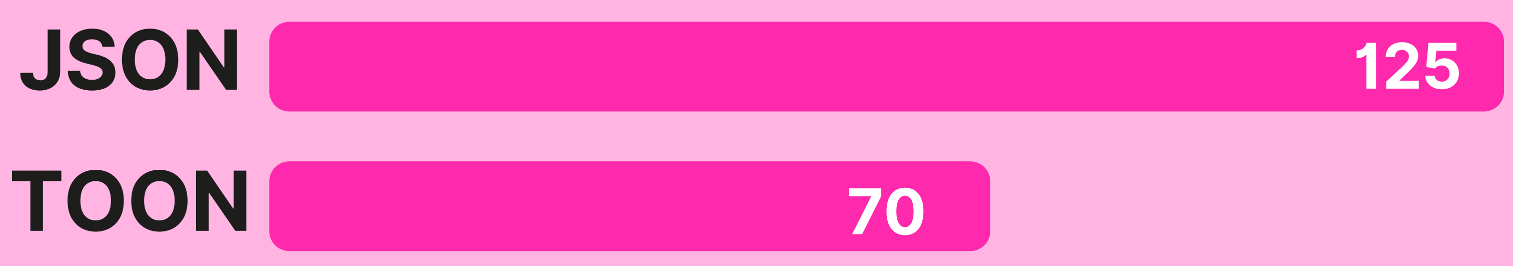
```
products[3]
{id,name,price} :
1,Laptop,3999.90
2,Mouse,149.90
3,Headset,499.00
```

✓

70 tokens

Reduce the number of tokens when exchanging structured data with language models.

Average Token Usage Comparison



What is Toon?

TOON is a new data serialization format designed with a code objective:

Reduce the number of tokens when exchanging structured data with language models.

While JSON uses verbose syntax with braces, quotes, and commas, TOON relies on a token-efficient tabular style, which is much closer to how LLMs naturally understand structured data.

Let’s make a quick comparison between JSON and TOON:

Here is some JSON with a users array that contains information about two users (two objects):

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

If you wanted to represent the same data in TOON, it would look like this:

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

Did you notice the differences?

- No quotes, braces, or colons in TOON.
- The users[2]{id,name,role}: declares an array of two objects with the fields id, name, and role.
- The lines below are simply the data rows.

You can see that TOON visibly reduced the token usage by 30-50%, depending on the data shape.

Why is TOON Important Now?

LLMs like GPT, Gemini, and Claude are token-based systems.

Each word, symbol, or chunk costs tokens for input and output. So, if you're preparing an LLM with structured data input/output like this:

```
{ "products": [ ... 300, "items" ... ] }
```

You might waste thousands of tokens in quotes, braces, colons, and repeated keys. TOON solves that by focusing on a compact yet structured representation.

Some of the key benefits of TOON are:
- 30-50% fewer tokens for uniform data sets.
- It has less syntactic clutter, which makes it easier for LLMs to reason about.
- It can be nested as we do with JSON.
- Works well with languages like Python, Go, Rust, and JavaScript.

Did you notice the differences?

- No quotes, braces, or colons in TOON.
- The users[2]{id,name,role}: declares an array of two objects with the fields id, name, and role.
- The lines below are simply the data rows.

You can see that TOON visibly reduced the token usage by 30-50%, depending on the data shape.

JSON vs TOON – Learn With Examples

Now that you have a basic idea of what TOON does and why it's helpful, let's look at some of the most used JSON structures and their equivalent representation in TOON.

1. A Simple Object

Here's how you'd represent an object with JSON:

```
{ "name": "Alice", "age": 30, "city": "Bengaluru" }
```

And here's how it works with TOON:

```
name: Alice
age: 30
city: Bengaluru
```

2. Array of Values

With JSON:

```
{ "colors": ["red", "green", "blue"] }
```

With TOON:

```
colors[3]: red,green,blue
```

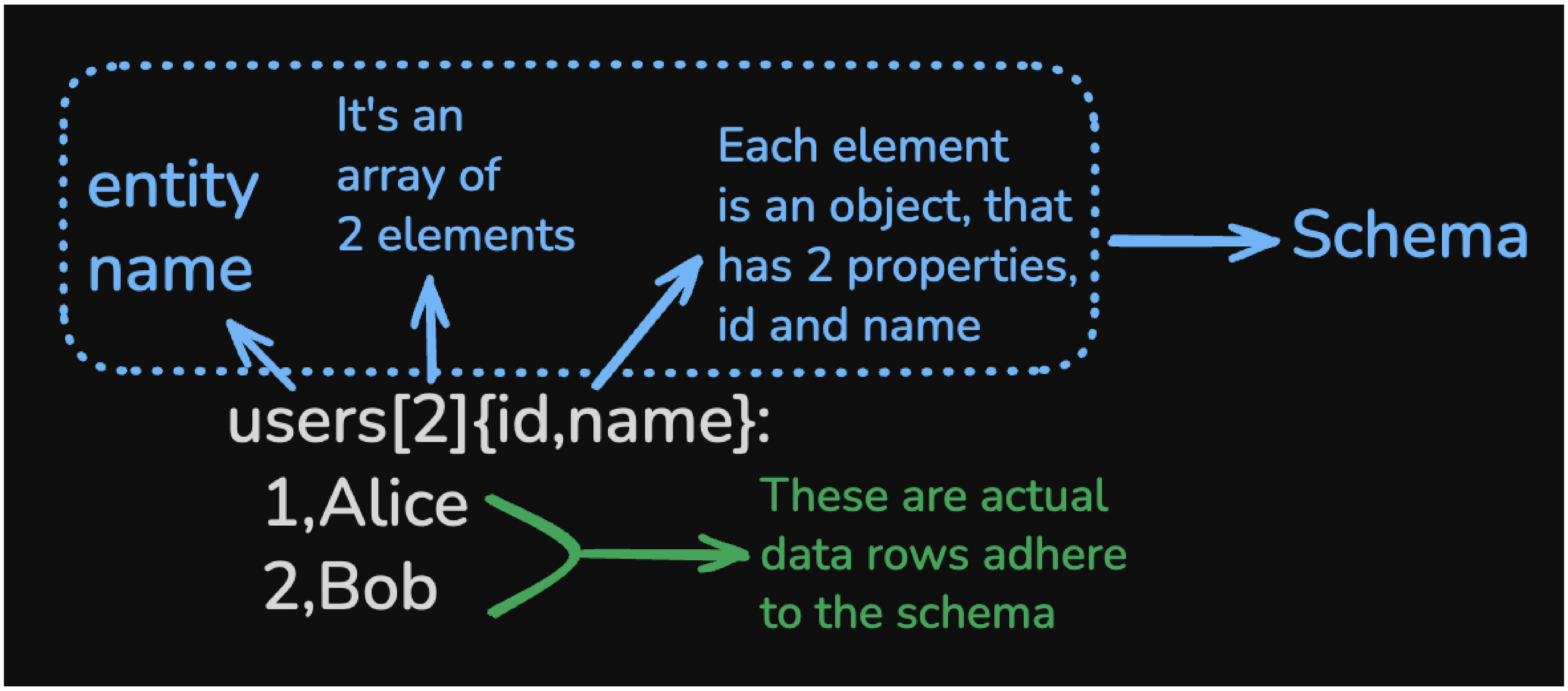
3. Array of Objects

With JSON:

```
{
  "users": [
    { "id": 1, "name": "Alice" },
    { "id": 2, "name": "Bob" }
  ]
}
```

With TOON:

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


entity name
It's an array of 2 elements
Each element is an object, that has 2 properties, id and name
Schema
users[2]{id,name}:
1,Alice
2,Bob
These are actual data rows adhere to the schema

4. Nested Objects

With JSON:

```
{
  "user": {
    "id": 1,
    "name": "Alice",
    "profile": { "age": 30, "city": "Bengaluru" }
  }
}
```

With TOON:

```
user:
  id: 1
  name: Alice
  profile:
    age: 30
    city: Bengaluru
```

Indentation represents nesting. It’s almost YAML-like, but it’s still structured.

5. Array of Objects With Nested Fields

With JSON:

```
{
  "teams": [
    {
      "name": "Team Alpha",
      "members": [
        { "id": 1, "name": "Alice" },
        { "id": 2, "name": "Bob" }
      ]
    }
  ]
}
```

With TOON:

```
teams[1]:
  - name: Team Alpha
    members[2]{id,name}:
      1,Alice
      2,Bob
```

This is still perfectly understandable, and much smaller than the JSON format.

Now that you know a bit about TOON syntax, let’s see how to use it with different programming languages.

JSON Might Still Be Better

Let's make it clear that TOON is NOT a universal replacement for JSON. In fact, you should still prefer JSON in many cases, such as when:

- Your data is deeply nested.
- Your data is irregular (for example, varying object shapes).
- Your application needs strict schema validations or type enforcement.
- NON-AI use cases where JSON still stands out and does its job perfectly.

A hybrid approach may even work better. Keep JSON for your application's data exchange format with APIs, but convert to TOON when it comes to sending data to LLMs.

TOON has already been explored for:

- Less token overhead for structured training data to fine-tune LLMs.
- Compact data exchange in Agent frameworks.
- Faster data serialization and deserialization between the MCP and AI workflow engines.
- With Serverless AI APIs, where cost and speed matter a lot.