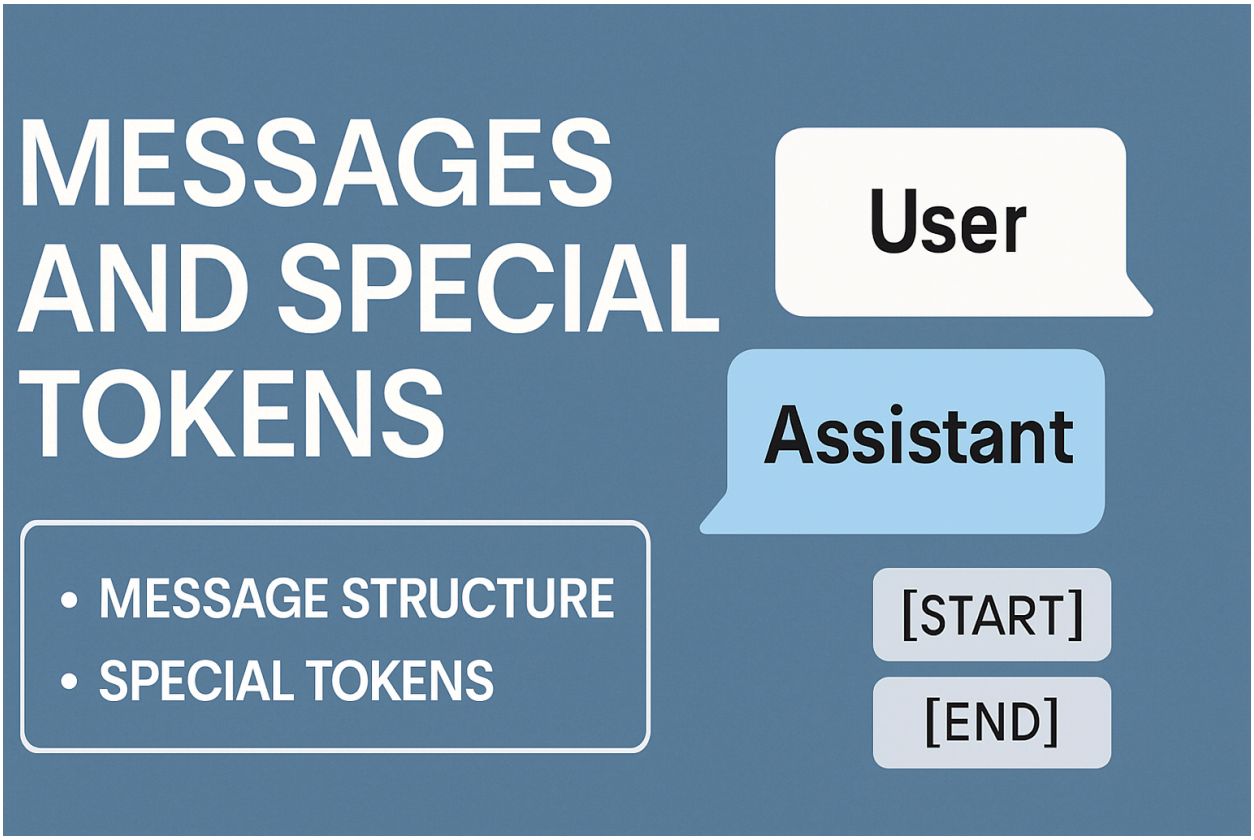
Lesson 3: Messages and Special Tokens

Type	@datasciencebrain



Understanding how LLMs handle conversations, role-based instructions, and structured prompts

1. Why Messages Matter

When we interact with an LLM (like GPT or any chat-based model), it might look like we're just typing in a prompt and getting a response. But under the hood, that's not what's really happening.

Language models don’t remember previous conversations unless we include them in the input again. So every time you send a message, the entire conversation history has to be **reconstructed and sent again**.

For example, if this is the conversation so far:

- User: What’s the capital of France?
- Assistant: Paris
- User: What’s the capital of Italy?

Then behind the scenes, we don’t just send the last question ("What’s the capital of Italy?") to the model. Instead, we send **all the previous turns**, formatted in a special way so the model knows who said what.

This is where **messages** come in. They simulate the flow of a conversation and preserve context.

---

2. What Are Messages?

A **message** is a structured unit of input to the model. Each message has two parts:

- `role`: Who is speaking (`user`, `assistant`, `system`, or others)
- `content`: What they are saying

Example message:

```
{
  "role": "user",
  "content": "What is the capital of France?"
}
```

These messages are not just used for display—they actually **instruct the model** on how to behave and who is speaking.

Over a conversation, the messages are added in order to create a sequence like:

```
[
  {"role": "system", "content": "You are a helpful assistant."},
```

```
  {"role": "user", "content": "What's the capital of France?"},
  {"role": "assistant", "content": "Paris."},
  {"role": "user", "content": "What's the capital of Italy?"}
]
```

This format keeps track of the entire dialogue, not just the most recent question.

---

3. The Purpose of Each Role

System

Used to **set behavior**, instructions, or context for the model.

Example:

```
{"role": "system", "content": "You are a polite customer support agent."}
```

User

Represents the person interacting with the model — the prompt giver.

```
{"role": "user", "content": "What's my order status?"}
```

Assistant

This is the model's reply. The system expects the model to fill in this message.

```
{"role": "assistant", "content": "Can you please provide your order number?"}
```

Tool or Function (optional, used in agents)

Used when tools like a calculator or code executor are involved. We'll cover this in more depth in later agent-related lessons.

---

4. Why Not Just Use Plain Text?

If you only send plain text like:

```
User: What's the weather today?
Assistant: It's 25°C and sunny.
User: Should I wear a jacket?
```

You're forcing the model to figure out the roles just by parsing text, which is unreliable. Instead, message formatting gives it **clear structure**, and that improves accuracy and behavior.

---

5. What Are Special Tokens?

When we use these messages, the tokenizer (the tool that prepares input for the model) **converts them into one long string**—but with **special tokens** inserted.

These special tokens are model-specific indicators that tell the model which role a line belongs to.

For example:

- `<|user|>` marks a user message
- `<|assistant|>` marks a model reply
- `<|system|>` marks a system prompt

So the message list we showed earlier becomes something like:

```
<|system|>
You are a helpful assistant.
<|user|>
What is the capital of France?
<|assistant|>
Paris.
<|user|>
What is the capital of Italy?
<|assistant|>
```

These tokens are **not shown to the user**, but they are inserted automatically and the model was trained with them. They are **crucial** to the model's ability to follow structure and context.

Different models have different special tokens. For example:

- SmolLM uses `<|im_start|>` and `<|im_end|>`
- Llama 3 uses `<|begin_of_text|>` and `<|eot_id|>`

You don't have to memorize these—the tokenizer handles them—but you must understand they exist and are part of why multi-turn chats work.

---

6. What is a Chat Template?

Every LLM that supports chat-style interaction comes with a **chat template**. This is a rule (often written using Jinja2 templating syntax) that defines:

- How to convert structured messages into a tokenized string
- What special tokens to insert
- What to do if the system message is missing
- Where the model should begin generating output

Example (SmolLM template)

```
{% for message in messages %}
<|im_start|>{{ message['role'] }}
{{ message['content'] }}<|im_end|>
{% endfor %}
```

This loops over each message and wraps it with `<|im_start|>` and `<|im_end|>`, plus adds the role. The model is trained on exactly this format, so sticking to it ensures better results.

Different models will have different templates.

---

7. Base Models vs Instruct Models

Understanding the difference is important:

Base Models

- Only trained to predict the next token in raw text
- Do not understand messages or chat templates
- You must prompt them manually and carefully

Instruct or Chat Models

- Fine-tuned using conversation-like data (ChatML or similar)
- Expect messages with roles and structure
- Use templates and special tokens behind the scenes

If you send chat-format messages to a **base model**, it won't know what to do. You must use the **right format for the right model**.

---

8. How to Use Messages in Code

With the `transformers` library, you don't need to write templates manually. You just provide your messages, and the tokenizer handles it using the model's template.

Example:

```
from transformers import AutoTokenizer

messages = [
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Hi!"},
    {"role": "assistant", "content": "Hello! How can I help you?"},
    {"role": "user", "content": "What's the capital of Japan?"}
]

tokenizer = AutoTokenizer.from_pretrained("HuggingFaceTB/SmolLM2-1.7B-Instruct")

# Convert messages into a full prompt string
prompt = tokenizer.apply_chat_template(
    messages,
    tokenize=False,             # return raw text instead of token IDs
```

```
    add_generation_prompt=True    # insert space for the model to reply
)

print(prompt)
```

This generates a properly formatted string using special tokens based on the model's expected style.

Summary

- LLMs don't remember—they read everything from scratch every time.
- A conversation is sent as a **list of messages**, each with a role and content.
- Roles include: `system`, `user`, `assistant`, and (for agents) `tool`, `tool_response`.
- **Special tokens** mark the boundaries between roles in the raw input.
- Every model has a **chat template** that defines how to format messages into a prompt.
- Use the correct chat format and tokenizer logic to get the best results from each model.

This structure is especially important when building **agents**, where the model must not only respond but also decide when to act, what tool to use, and how to interpret tool outputs.
