# LLMはなぜツールを使えるのか？

～ AIエージェントの仕組みを紐解く ～

社内勉強会 LT

# AIエージェントとは？

**ChatGPT** = 質問に答える（会話のみ）

**AIエージェント** = 質問を理解して**実際に作業する**

> 「ファイルを作って」→ 本当にファイルが作られる
> 「コードを修正して」→ 本当にコードが変わる

**LLM + ツール + ループ = AIエージェント**

# 3つの構成要素

## 1. LLM（大規模言語モデル）

- 考える役割（何をすべきか判断）

## 2. ツール

- 行動する役割（ファイル操作、API呼び出し等）

## 3. ループ

- LLM → ツール → LLM → ... を繰り返す

**たったこれだけ！**

# LLMの本質：Autocompleteエンジン

スマホの**予測変換**と同じ仕組み

```
入力: "I love"  →  LLM  →  予測: "cats"
          ↓
入力: "I love cats"  →  LLM  →  予測: "."
          ↓
結果: "I love cats."  （文章完成）
```

**「次の1語を当てる」をひたすら繰り返す**

# Autocomplete = Autoregressive

| 呼び方 | 意味 |
| --- | --- |
| **Autocomplete** | 体験としての名前（予測変換） |
| **Autoregressive** | 仕組みの名前（自己回帰） |

> "We train GPT-3, an **autoregressive** language model"
> — Language Models are Few-Shot Learners (OpenAI, 2020)

**同じ仕組みを、体験と技術の観点から呼んでいる**

# なぜツールが使えるのか？

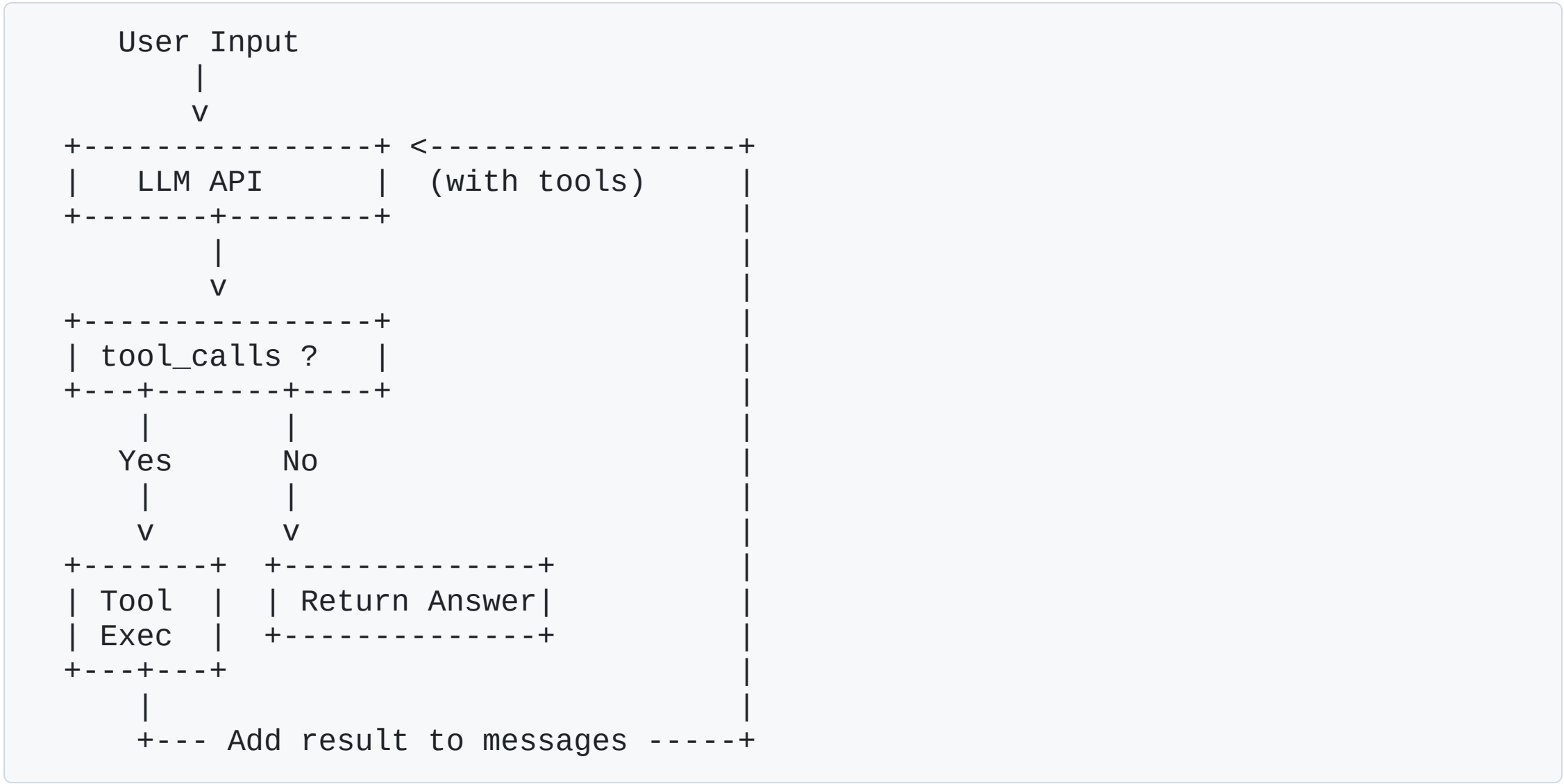
## ツール定義 → プロンプトに埋め込み → 次トークン予測

```
システム:  「list_dir というツールがあります...」
ユーザー:  「ファイル一覧を教えて」
LLM予測:  「list_dir を呼び出すべき」← 学習パターンから
```

**大量のコード・ドキュメントを学習した結果**
→ 「こういう時はこのツールを使う」パターンを習得

> 参考: Toolformer (Meta AI, 2023)

# 全体の流れ（function_call フロー）



# 内部プロンプトの実例（Qwen3-Coder + vLLM）

```
<|im_start|>system
You are a helpful coding assistant...

# Tools
<tools>
<function><name>read_file</name>...</function>
<function><name>list_dir</name>...</function>
</tools>
<|im_end|>
<|im_start|>user
README.md の内容を要約してください。
<|im_end|>
<|im_start|>assistant   ← ここで止まっている！
```

**LLMは「次に何が来るか」を予測** → ツール呼び出しを生成

# デモ: kakko-de

Clojure製AIエージェント（Function Calling ベース）

```
clj -M:run "test/ 配下のファイル一覧を教えて"
```

## 観察ポイント

- LLM が `list_dir` ツールを選択する
- ツール実行結果が LLM に返される
- LLM が結果をもとに回答を生成

# まとめ

## 学んだこと

1. **AIエージェント = LLM + ツール + ループ**
2. **Function Calling** でLLMがツールを呼び出す
3. **仕組みは意外とシンプル**

# 参考: ReActパターン

**ReAct = Reasoning + Acting**（明示的な推論を出力）

| ステップ | 説明 |
| --- | --- |
| Thought | 推論を**テキストで出力** |
| Action | ツールを実行 |
| Observation | 結果を確認 |

Function Calling は ReAct の流れを実用化したもの
（明示的な推論出力 → 暗黙的な推論へ）

> ReAct (Yao et al., 2022)