

# 第2回：LLMノードとプロンプト設計

Difyで作るAIアプリケーション

# この回で学ぶこと

- LLMノードの基本的な使い方
- 効果的なプロンプトの書き方
- 変数を使った動的なプロンプト作成
- 構造化出力 (JSON Schema)
- 実際に動くAIアプリケーションの作成

💡 プログラミングの知識は不要です

# 1. LLMノードとは

# LLMノードの役割

**LLMノード**は、ChatGPTやClaudeなどのAIモデルと会話するための部品

[ユーザーの質問] → [LLMノード] → [AIの回答]

## できること

- 質問に答える / 文章を要約する
- 翻訳する / アイデアを出す
- 文章を添削する
- など、さまざまなタスク

# ストリーミング vs ブロッキング

## ストリーミングモード (推奨)

AIが考えながら、一文字ずつリアルタイムに表示

「こんに」「ちは」「、」「今日」「は」...

→ 待ち時間が短く感じられる

## ブロッキングモード

AIが回答をすべて作り終えてから、一度に表示

(考え中...) → 「こんにちは、今日はいい天気ですね」

→ 他のシステムと連携する場合に使用

## 2. LLMノードの追加と設定

## ステップ1~3: アプリケーション作成

1. Difyにログイン → 「スタジオ」をクリック
2. 「空白のワークフローから作成」を選択
3. アプリ名を入力（例：「はじめてのLLM」）
4. 「開始」ノードの「+」から「LLM」を選択
5. モデルを選択（初心者は **gemini-2.5-flash** 推奨）

# パラメータ設定：Temperature

AIの「創造性」を調整するパラメータ

設定値	動作	使いどころ
0.0	毎回同じ答え	計算問題、正確な情報
0.5	バランス	一般的なQ&A
0.9	創造的な答え	物語作成、アイデア出し

質問：「1+1は？」

Temperature 0.0 → 毎回「2です」

Temperature 0.9 → 「2です」「二つです」「2ですね」など変化

## パラメータ設定：Maximum Tokens

AIが生成する文章の長さの上限

設定値	目安
512	短い回答（数行～1段落）
1024	標準的な回答（400～800文字）
2048	長文の生成（ブログ記事など）
4096+	非常に長いレポート

 トークン数が多いほど、時間とコストが増加

### 3. プロンプトの書き方

# プロンプトとは

AIに対する「指示文」のこと

## 基本的なプロンプトの構成

[役割] + [タスク] + [制約条件] + [出力形式]

## 悪い例 vs 良い例

✗ **悪い例** : 説明して → 何を説明するのか不明確

✓ **良い例** :

あなたは親切な先生です。  
プロンプトエンジニアリングについて、  
小学生にもわかるように説明してください。

# 詳細なプロンプトの例

## 【役割】

あなたは経験豊富なカスタマーサポート担当者です。

## 【タスク】

顧客からの問い合わせに対して、丁寧で分かりやすい回答を作成してください。

## 【制約条件】

- 敬語を使用すること
- 回答は300文字以内
- 専門用語は避けること

## 【出力形式】

1. お問い合わせ内容の確認
2. 回答
3. 追加で必要な情報があれば案内

# プロンプトのテクニック① : Few-shot (例示)

## AIに「お手本」を見せる方法

以下の例を参考に、商品レビューを分類してください。

### 【例1】

レビュー：「とても良い商品でした！」

分類：ポジティブ

### 【例2】

レビュー：「期待外れでした」

分類：ネガティブ

### 【分類してください】

レビュー：「普通です」

分類：

## プロンプトのテクニック②：ステップバイステップ

複雑なタスクを段階的に実行

以下の手順で文章を要約してください：

- ステップ1：文章の主題を特定する
- ステップ2：重要なポイントを3つ抜き出す
- ステップ3：各ポイントを1文にまとめる
- ステップ4：3つの文を組み合わせる要約を作る

文章：  
{ここにユーザーの文章が入る}

## プロンプトのテクニック③：ペルソナ設定

AIに具体的な役割を与える

あなたは以下の設定のキャラクターです：

- 名前：さくらAI
- 性格：明るく親しみやすい
- 口調：「です・ます」調
- 特徴：絵文字を適度に使う（多用しない）

この設定で、ユーザーの質問に答えてください。

## 💡 プロのテクニック：自動生成機能

自分でゼロからプロンプトを書くのが難しい場合

1. プロンプト入力欄の右上にある「✨」をクリック
2. やりたいことを一言入力
  - 例：「美味しいカレーの作り方を教えるボット」
3. 「生成」ボタンを押す
4. AIが詳細なプロンプトを自動生成！

## 4. 変数を使ったプロンプト

## 変数とは

ユーザーの入力や他のノードの結果を受け取る「箱」

変数なし：「天気について教えて」  
→ 毎回「天気」についてしか答えない

変数あり：「`{{topic}}`について教えて」  
→ 箱(topic)に「料理」→「料理について教えて」  
→ 箱(topic)に「歴史」→「歴史について教えて」

**ポイント：** `{{変数名}}` で変数を参照

# 開始ブロック（開始ノード）

ワークフローの入り口となる特別なノード

[ユーザー] → 入力 → [開始ブロック] → 変数として保存 → [LLMノードで使用]

## 設定できる項目

項目	説明	例
フィールド名	変数の名前	<code>user_question</code> , <code>topic</code>
フィールドタイプ	データの種類	テキスト、段落、数値など
必須	入力が必要か	はい/いいえ
説明	ユーザーへのガイド	「質問を入力してください」

## フィールドタイプの種類

タイプ	説明	例
テキスト	短い1行の入力	名前、タイトル
段落	複数行の長文入力	質問文、レビュー
数値	数字のみの入力	年齢、金額
ファイル	ファイルアップロード	画像、PDF
選択	プルダウンから選択	「簡単/普通/詳しく」

## 開始ブロックの変数 vs 会話変数

項目	開始ブロックの変数	会話変数
スコープ	1回のワークフロー実行	会話全体
値の保持	終了で消える	会話が続く限り保持
使用場面	ワークフロー型	チャットボット型
更新方法	毎回ユーザーが入力	ワークフロー内で更新可

# 複数の変数を使う例

## 入力変数

- `topic` : 話題
- `detail_level` : 詳しさ
- `target_audience` : 対象者

## プロンプト

対象者 : {{target\_audience}}

{{topic}}について、{{detail\_level}}説明してください。

説明のポイント :

- 対象者に合わせた言葉遣い
- 具体例を含める

# 条件分岐ノード

## 条件分岐ノード (IF/ELSEノード)

変数の値によって異なる処理を実行

[開始ノード]



[条件分岐ノード]

- └ detail\_level = "簡単" → [LLMノード：小学生向け]
- └ detail\_level = "普通" → [LLMノード：一般向け]
- └ detail\_level = "専門的" → [LLMノード：専門家向け]



[終了ノード]

→ 各レベルに特化したプロンプトで高品質な回答

## ノード間での変数の受け渡し

```
[開始ノード]
  ↓ user_question
[LLMノード1] ← 質問に回答
  ↓ output (生成された回答)
[LLMノード2] ← 回答を要約
  ↓
[終了ノード]
```

## LLMノード2のプロンプト

以下の文章を3行で要約してください：

```
{{LLMノード1.output}}
```

## 5. 実践：AIアシスタントを作る

# 演習1: シンプルな質問応答ボット

## 設定内容

- 入力変数 : `question` (テキスト、必須)
- モデル : `gemini-2.5-flash`
- Temperature : 0.7

## プロンプト

あなたは親切で知識豊富なAIアシスタントです。  
ユーザーの質問に対して、正確で分かりやすい回答を提供してください。

質問 : `{{question}}`

回答 :

## 演習2: レビュー感情分析ツール

### プロンプト (Few-shot使用)

以下の例を参考に、レビューを分類してください。

#### 【例1】

レビュー: 「素晴らしい商品です。買って良かった！」

分類: ポジティブ

#### 【例2】

レビュー: 「思っていたものと違いました。残念です。」

分類: ネガティブ

#### 【分類対象】

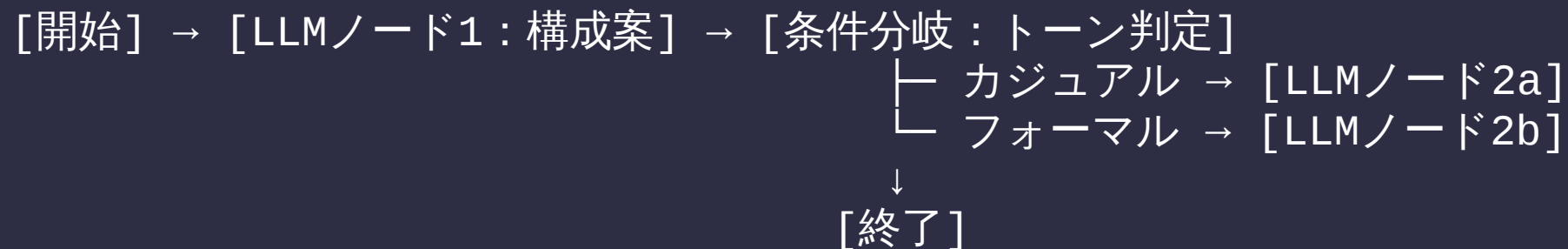
レビュー: {{review\_text}}

分類: [ポジティブ/ネガティブ/ニュートラル]

理由: [簡潔に説明]

# 演習3: ブログ記事生成アシスタント

## ワークフロー構成



## 入力変数

- `article_topic` : 記事のテーマ
- `target_length` : 文字数
- `tone` : トーン (カジュアル/フォーマル)

## 6. 構造化出力 (Structured Output)

# 構造化出力とは

AIの出力を決まった形式（JSON形式）で受け取る機能

## 普通のテキスト出力

ユーザー名： 山田太郎  
年齢： 30  
趣味： 読書、旅行

→ 人間には読みやすいが、プログラムで処理しにくい

## 構造化出力（JSON）

```
{  
  "username": "山田太郎",  
  "age": 30,  
  "interests": ["読書", "旅行"]  
}
```

## いつ使うのか

使用場面	説明
データベース保存	ユーザー情報、商品情報などを保存
API連携	他のシステムにデータを渡す
ワークフロー連携	次のノードで特定の値だけ使いたい
複数の値を抽出	文章から名前、日付、金額などを抽出

# JSON Schemaとは

JSONデータの「型紙」 - どんなデータがどんな形で出てくるかを定義

```
{
  "type": "object",
  "properties": {
    "username": {
      "type": "string",
      "description": "ユーザーの名前"
    },
    "age": {
      "type": "number",
      "description": "ユーザーの年齢"
    }
  },
  "required": ["username", "age"]
}
```

## 主なデータ型

データ型	説明	例
<b>string</b>	文字列	"こんにちは"
<b>number</b>	数値	30 , 3.14
<b>boolean</b>	真偽値	true , false
<b>array</b>	配列（リスト）	["りんご", "バナナ"]
<b>object</b>	オブジェクト	{"name": "太郎"}

# 構造化出力の設定方法

## 対応モデルを選択

- OpenAI GPT-4o / GPT-4o-mini
- Claude 4.5 Sonnet / Claude 4.5 Opus
- Google Gemini 2.5 Flash / Pro

## 設定手順

1. LLMノードの設定パネルを開く
2. 「応答形式」を **JSON Schema** に変更
3. JSON Schemaエディターに構造を定義

# 実例：商品レビュー分析

## JSON Schema

```
{
  "type": "object",
  "properties": {
    "sentiment": {
      "type": "string",
      "enum": ["ポジティブ", "ネガティブ", "ニュートラル"]
    },
    "rating": {
      "type": "number",
      "description": "評価 (1~5) "
    },
    "key_points": {
      "type": "array",
      "items": { "type": "string" }
    }
  },
  "required": ["sentiment", "rating", "key_points"]
}
```

# 実例：問い合わせ分類

## JSON Schema

```
{
  "type": "object",
  "properties": {
    "category": {
      "type": "string",
      "enum": ["配送", "返品・交換", "商品情報", "支払い", "その他"]
    },
    "urgency": {
      "type": "string",
      "enum": ["高", "中", "低"]
    },
    "summary": {
      "type": "string",
      "description": "問い合わせ内容の要約"
    }
  },
  "required": ["category", "urgency", "summary"]
}
```

# 演習8: 問い合わせ自動振り分けシステム

## ワークフロー構成

[開始ノード]



[LLMノード：問い合わせ分析] (構造化出力)



[条件分岐ノード：緊急度判定]



緊急度「高」 → [LLMノード：緊急対応メッセージ]

緊急度「中」 → [LLMノード：標準対応メッセージ]

緊急度「低」 → [LLMノード：自動応答メッセージ]



[終了ノード]

→ 構造化出力 + 条件分岐の実践的な活用例

## 演習8: テストケース

### ケース1: 緊急度「高」

システムにログインできません。  
「エラーコード500」と表示されて、本日中に提出しないといけない重要な書類が取り出せず、非常に困っています。

→ 専任サポートチームが即時対応

### ケース2: 緊急度「低」

アプリの通知設定を変更したいのですが、設定画面の場所がわかりません。

→ FAQページ案内 + 自動応答

# 構造化出力のトラブルシューティング

## よくあるエラーと対処法

エラー	原因	対処法
Failed to parse	モデル非対応	GPT-4o、Claude等に変更
フィールド欠落	required未設定	必須フィールドを追加
型が異なる	説明不足	descriptionを詳しく記載

# 構造化出力のベストプラクティス

## ✓ Do (推奨)

- 必須フィールドは必ず `required` に指定
- 各フィールドに `description` を記載
- 選択肢が限られる場合は `enum` を使用

## ✗ Don't (非推奨)



- 複雑すぎるネスト構造 (3階層以上)
- あいまいなフィールド名 ( `data` , `info` など )
- 対応していないモデルで使用

## よくある質問 (FAQ)

## Q: Temperatureは何に設定すればいい？

タスク	推奨値
事実確認、計算	0.0 - 0.3
一般的なQ&A	0.5 - 0.7
創作、アイデア出し	0.8 - 1.0

## Q: 変数名に使える文字は？

- 英数字とアンダースコア `_` のみ
- 数字から始めない
-  `user_question`, `topic1`
-  `ユーザー質問`, `1topic`

## Q: 構造化出力とテキスト出力の使い分けは？

用途	推奨形式
人が読むための回答	テキスト
データベースに保存	構造化出力
次のノードで特定の値を使う	構造化出力
自由な形式の文章生成	テキスト

まとめ

# この回で学んだこと

## LLMノードの基本

- ストリーミングとブロッキングの違い
- Temperature、Max Tokensのパラメータ設定

## プロンプトの書き方

- 役割、タスク、制約条件、出力形式
- Few-shot、ステップバイステップ、ペルソナ設定

## 変数と条件分岐

- `{{変数名}}` で動的なプロンプト
- IF/ELSEノードで処理を分岐

## この回で学んだこと（続き）

### 構造化出力（JSON Schema）

- JSON形式でのデータ受け取り
- データ型：string, number, boolean, array, object
- enum による選択肢の限定
- 条件分岐との組み合わせ

### 実践的なアプリケーション

- 質問応答ボット / 感情分析ツール
- 問い合わせ自動振り分けシステム

# 次のステップ

## 第3回：ナレッジベースとRAG

- 独自のドキュメントを使った回答生成
- PDFやWebサイトの情報を活用
- より正確で信頼性の高いAIアプリ

## 練習課題

1. 翻訳アシスタント - 日本語 → 多言語翻訳
2. 文章添削ツール - 改善案と理由を提示
3. キャッチコピー生成 - 3つの案を出力
4. 求人情報の構造化抽出 (構造化出力)
5. 会議議事録の要約 (構造化出力)

## 参考リンク

- [Dify公式ドキュメント](#)
- [プロンプトエンジニアリングガイド](#)

お疲れ様でした！ 🎉

分からないことがあれば、  
コミュニティやサポートに質問してみましょう。