

WP2-1 対話理解モジュール設計書

#対話理解モジュール ユーザー発話をテキスト／音声から解析(最後に組み込む予定) Gemini API(LLM)
+ Whisper API(音声→テキスト) + MeCab/SudachiPy

gemini モデルのインストールコマンド:

```
# pip install google-generativeai
```

API キーの設定(安全管理)

API キーを環境変数として設定します

```
export GOOGLE_API_KEY="YOUR_GEMINI_API_KEY"
```

Python では:

```
import os
```

```
import google.generativeai as genai
```

```
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
```

モデル呼び出し

Gemini-1.5-Pro(または最新モデル)を使用します。


```
model = genai.GenerativeModel("gemini-1.5-pro")
```

```
response = model.generate_content("こんにちは、あなたは法律 AI です。")
```

```
print(response.text)
```

データフロー:

入出力仕様(I/O 設計)

 入力(Input)

種別 内容 型 例

音声 ユーザー発話(.wav / .mp3) bytes / str "user_input.wav"

テキスト 音声変換後 or 直接入力 str 「夫が家事を全くしません」

 出力(Output)

種別 内容 型 例

テキスト解析結果 発話の構造と意味分析 dict { "intent": "不満訴え", "subject": "夫",
"topic": "家事負担" }

Gemini 解析結果 Gemini 応答 str 「家事分担の偏りに関する不満が検出されました。」

全体フロー

[音声入力]

↓

Whisper(音声→テキスト)

↓

MeCab/SudachiPy(形態素解析)

↓

Gemini(文脈理解・意図抽出)

↓

[出力: 構文 + 意図解析結果]

将来の統合を見据えた統合設計

riri_core/

```
├─ dialogue_understanding/
│   ├── __init__.py
│   ├── whisper_interface.py      # 音声→テキスト
│   ├── morpho_parser.py          # MeCab or SudachiPy
│   ├── gemini_core.py            # Gemini 連携
│   └─ dialogue_understander.py  # 統合制御
├─ emotion_analysis/
├─ legal_reasoning/
├─ main.py
└─ config.py
```

将来統合のための設計上の注意点

項目 説明 理由

入出力を JSON 化 他モジュール間連携を容易にする Emotion / Legal モジュールに渡しやすくする

Whisper・Gemini・形態素解析をクラス分離 交換性を確保 モデル切替や API 変更に対応

Gemini の呼び出し結果を構造化 JSON 出力 法令推論や折衷案生成に転用可 下流層での処理が容易

環境変数管理(.env 使用) API キーのセキュリティ確保 外部公開を防止

モジュール内でロギング処理追加