**# WP2-2 論争解析モジュール 技術文書**

**## 1. 概要**

**### 1.1 モジュールの目的**

WP2-2論争解析モジュールは、双方の発言ログを解析して「論点化」および「対立関係の抽出」を行うシステムです。Google Gemini APIとBERT（自然言語分類モデル）を組み合わせて、発言間の立場・主張・論点を自動抽出し、構造化されたデータとして出力します。

**### 1.2 技術的特徴**

- **\*\*ハイブリッド解析\*\***: Gemini API（自然言語理解）とBERT（構造分析）の組み合わせ

- **\*\*リアルタイム処理\*\***: FastAPIによる高速なREST API

- **\*\*スケーラブル設計\*\***: モジュール化された構造で拡張性を確保

- **\*\*エラー耐性\*\***: 包括的なエラーハンドリングとフォールバック機能

**## 2. ファイル構成と動作**

**### 2.1 ディレクトリ構造**

```

WP2-2　論争解析モジュール/

├── app/ # メインアプリケーションディレクトリ

│ ├── \_\_init\_\_.py # パッケージ初期化ファイル

│ ├── main.py # FastAPIメインアプリケーション

│ ├── config.py # 設定管理（環境変数読み込み）

│ ├── logger.py # ログ設定とロガー管理

│ ├── schemas.py # データモデル定義（Pydantic）

│ ├── clients/ # 外部APIクライアント

│ │ ├── \_\_init\_\_.py

│ │ ├── gemini\_client.py # Gemini APIクライアント

│ │ └── bert\_client.py # BERT分類クライアント

│ ├── services/ # ビジネスロジック

│ │ ├── \_\_init\_\_.py

│ │ └── dispute\_analysis\_service.py # 論争解析メインサービス

│ └── utils/ # ユーティリティ

│ ├── \_\_init\_\_.py

│ └── error\_mapping.py # エラーハンドリング

├── requirements.txt # Python依存関係

├── env.example # 環境変数設定例

├── 処理フロー図.md # システム処理フロー

├── 入出力例.md # API使用例とサンプル

└── README.md # モジュール説明書

```

**### 2.2 各ファイルの役割**

**#### 2.2.1 メインアプリケーション**

- **\*\***`main.py`**\*\***: FastAPIアプリケーションのエントリーポイント

- APIエンドポイント定義

- リクエスト/レスポンス処理

- エラーハンドリング

- CORS設定

**#### 2.2.2 設定・ログ管理**

- **\*\***`config.py`**\*\***: 環境変数とアプリケーション設定の管理

- Gemini API設定

- BERTモデル設定

- ネットワーク設定

- 論争解析パラメータ

- **\*\***`logger.py`**\*\***: ログ設定とロガー管理

- ログレベル設定

- フォーマッター設定

- コンソール出力設定

**#### 2.2.3 データモデル**

- **\*\***`schemas.py`**\*\***: Pydanticベースのデータモデル定義

- リクエスト/レスポンススキーマ

- データバリデーション

- 型安全性の確保

**#### 2.2.4 外部APIクライアント**

- **\*\***`gemini\_client.py`**\*\***: Google Gemini APIクライアント

- 論点分析API呼び出し

- 立場分析API呼び出し

- 関係分析API呼び出し

- タイムアウト・エラーハンドリング

- **\*\***`bert\_client.py`**\*\***: BERT分類モデルクライアント

- 発言分類処理

- 信頼度評価

- サブカテゴリ決定

- キーワード抽出

**#### 2.2.5 ビジネスロジック**

- **\*\***`dispute\_analysis\_service.py`**\*\***: 論争解析メインサービス

- 解析処理の統合

- 結果のマージ

- サマリー生成

- 使用量計算

**#### 2.2.6 ユーティリティ**

- **\*\***`error\_mapping.py`**\*\***: エラーハンドリング

- カスタムエラー定義

- HTTPステータスコードマッピング

- エラーレスポンス生成

**## 3. 機能概要**

**### 3.1 主要機能**

**#### 3.1.1 論点抽出機能**

- **\*\*目的\*\***: 対話ログから主要な論点を自動抽出

- **\*\*技術\*\***: Gemini APIの自然言語理解

- **\*\*出力\*\***: 論点名、信頼度、キーワードリスト

**#### 3.1.2 立場分析機能**

- **\*\*目的\*\***: 各発言者の論点に対する立場を分析

- **\*\*技術\*\***: Gemini APIの文脈理解

- **\*\*出力\*\***: A/Bの立場、信頼度、根拠となる発言

**#### 3.1.3 関係分析機能**

- **\*\*目的\*\***: 論点間の対立・合意関係を分析

- **\*\*技術\*\***: Gemini APIの意味的関係理解

- **\*\*出力\*\***: 関係タイプ、対立強度

**#### 3.1.4 発言分類機能**

- **\*\*目的\*\***: 各発言を構造的に分類

- **\*\*技術\*\***: BERT分類モデル

- **\*\*出力\*\***: カテゴリ、信頼度、サブカテゴリ

**### 3.2 処理フロー**

```

入力: 対話ログ（双方の発言）

↓

┌─────────────────────────────────────────────────────────────┐

│ 前処理 │

│ - 発言データの正規化 │

│ - 入力検証 │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ BERT分類処理 │

│ - 各発言を「主張/根拠/反論/補足」等に分類 │

│ - 信頼度スコア付与 │

│ - サブカテゴリ決定 │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ Gemini API処理（並列実行） │

│ │

│ ┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐ │

│ │ 論点分析 │ │ 立場分析 │ │ 関係分析 │ │

│ │ │ │ │ │ │ │

│ │ - 主要論点抽出 │ │ - A/Bの立場特定 │ │ - 対立強度計算 │ │

│ │ - キーワード抽出 │ │ - 根拠発言特定 │ │ - 関係タイプ分類 │ │

│ │ - 信頼度評価 │ │ - 信頼度評価 │ │ - 信頼度評価 │ │

│ └─────────────────┘ └─────────────────┘ └─────────────────┘ │

└─────────────────────────────────────────────────────────────┘

↓

┌─────────────────────────────────────────────────────────────┐

│ 結果統合 │

│ - BERT分類結果とGemini分析結果をマージ │

│ - 論点構造の構築 │

│ - 対立関係図データの生成 │

│ - サマリー情報の計算 │

└─────────────────────────────────────────────────────────────┘

↓

出力: 論争解析結果（JSON形式）

```

**## 4. 入出力仕様**

**### 4.1 入力仕様**

**#### 4.1.1 リクエスト形式**

```json

{

"messages": [

{

"speaker": "A",

"text": "発言内容1"

},

{

"speaker": "B",

"text": "発言内容2"

}

],

"analysis\_depth": "standard"

}

```

**#### 4.1.2 入力パラメータ**

- **\*\***`messages`**\*\*** (必須): 発言リスト

- `speaker`: 発言者（"A" または "B"）

- `text`: 発言内容（文字列）

- **\*\***`analysis\_depth`**\*\*** (オプション): 解析深度

- `"basic"`: 基本解析

- `"standard"`: 標準解析（デフォルト）

- `"detailed"`: 詳細解析

**#### 4.1.3 入力制約**

- 発言数: 最低2件以上

- 発言者: "A" または "B" のみ

- 文字数: 各発言最大512文字（BERT制限）

**### 4.2 出力仕様**

**#### 4.2.1 レスポンス形式**

```json

{

"success": true,

"data": {

"analysis": {

"topics": [...], // 論点リスト

"relations": [...], // 対立関係データ

"message\_analyses": [...], // 発言分析結果

"summary": {...} // 解析サマリー

},

"usage": {...}, // 使用量情報

"meta": {...} // メタ情報

},

"error": null

}

```

**#### 4.2.2 論点データ構造**

```json

{

"topic\_id": "topic\_1",

"topic\_name": "AI導入の労働効率向上",

"confidence": 0.92,

"keywords": ["AI", "労働効率", "自動化"]

}

```

**#### 4.2.3 対立関係データ構造**

```json

{

"topic": "AI導入の労働効率向上",

"a\_position": "賛成",

"b\_position": "懸念",

"relation\_type": "対立",

"intensity": 0.7

}

```

**#### 4.2.4 発言分析データ構造**

```json

{

"speaker": "A",

"text": "AIの導入により労働効率が向上します。",

"classification": {

"category": "主張",

"confidence": 0.89,

"subcategory": "積極的主張"

},

"topics": ["AI導入の労働効率向上"],

"sentiment": null

}

```

**## 5. 主要関数**

**### 5.1 APIエンドポイント**

**#### 5.1.1 論争解析エンドポイント**

```python

@app.post("/v1/analyze", response\_model=ApiResponse)

async def analyze\_dispute(req: DisputeAnalysisRequest):

"""

論争解析エンドポイント

Args:

req: 論争解析リクエスト

Returns:

統一JSONレスポンス形式

"""

```

**#### 5.1.2 ヘルスチェックエンドポイント**

```python

@app.get("/health")

async def health\_check():

"""

詳細ヘルスチェック

システム状態とモデル情報を返す

"""

```

**### 5.2 サービス関数**

**#### 5.2.1 論争解析メイン関数**

```python

async def analyze\_dispute(self, request: DisputeAnalysisRequest) -> SuccessData:

"""

論争解析を実行

Args:

request: 論争解析リクエスト

Returns:

解析結果

"""

```

**#### 5.2.2 結果統合関数**

```python

def \_integrate\_results(

self,

topics: List[Dict[str, Any]],

positions: List[Dict[str, Any]],

relations: List[Dict[str, Any]],

bert\_results: List[Dict[str, Any]],

messages: List[Dict[str, str]]

) -> DisputeAnalysisData:

"""

各分析結果を統合して最終結果を構築

"""

```

**### 5.3 クライアント関数**

**#### 5.3.1 Gemini API関数**

```python

async def analyze\_dispute\_topics(self, messages: List[Dict[str, str]]) -> Tuple[str, Dict[str, Any]]:

"""論争の論点を分析"""

async def analyze\_positions(self, messages: List[Dict[str, str]], topics: List[str]) -> Tuple[str, Dict[str, Any]]:

"""各論点での立場を分析"""

async def analyze\_relations(self, messages: List[Dict[str, str]], topics: List[str]) -> Tuple[str, Dict[str, Any]]:

"""論点間の関係を分析"""

```

**#### 5.3.2 BERT分類関数**

```python

async def classify\_messages(self, messages: List[Dict[str, str]]) -> List[Dict[str, Any]]:

"""発言リストを分類"""

def extract\_topics\_from\_messages(self, messages: List[Dict[str, str]]) -> List[str]:

"""発言から論点キーワードを抽出"""

```

**## 6. 依存関係**

**### 6.1 Pythonパッケージ**

**#### 6.1.1 コア依存関係**

- **\*\*fastapi==0.104.1\*\***: Web APIフレームワーク

- **\*\*uvicorn==0.24.0\*\***: ASGIサーバー

- **\*\*httpx==0.25.2\*\***: HTTPクライアント

- **\*\*pydantic==2.5.0\*\***: データバリデーション

- **\*\*pydantic-settings==2.1.0\*\***: 設定管理

**#### 6.1.2 AI/ML依存関係**

- **\*\*transformers==4.36.0\*\***: Hugging Face Transformers

- **\*\*torch>=2.2.0\*\***: PyTorch（機械学習フレームワーク）

- **\*\*numpy>=1.24.3\*\***: 数値計算ライブラリ

- **\*\*scikit-learn>=1.3.2\*\***: 機械学習ライブラリ

**#### 6.1.3 間接依存関係**

- **\*\*anyio\*\***: 非同期I/O

- **\*\*starlette\*\***: Webフレームワーク（FastAPIの基盤）

- **\*\*huggingface-hub\*\***: Hugging Face Hub

- **\*\*tokenizers\*\***: トークナイザー

- **\*\*safetensors\*\***: セーフテンソル

**### 6.2 外部サービス**

**#### 6.2.1 Google Gemini API**

- **\*\*エンドポイント\*\***: `https://generativelanguage.googleapis.com/v1beta/models/{model}:generateContent`

- **\*\*認証\*\***: APIキーベース

- **\*\*制限\*\***: トークン数制限、レート制限

**#### 6.2.2 Hugging Face Hub**

- **\*\*モデル\*\***: `cl-tohoku/bert-base-japanese-v3`

- **\*\*用途\*\***: 日本語BERT分類モデル

- **\*\*ダウンロード\*\***: 初回実行時に自動ダウンロード

**## 7. 実行環境**

**### 7.1 システム要件**

**#### 7.1.1 ハードウェア要件**

- **\*\*CPU\*\***: 2コア以上推奨

- **\*\*メモリ\*\***: 4GB以上（BERTモデル用）

- **\*\*ストレージ\*\***: 2GB以上の空き容量

- **\*\*GPU\*\***: オプション（CUDA対応GPUで高速化可能）

**#### 7.1.2 ソフトウェア要件**

- **\*\*OS\*\***: macOS, Linux, Windows

- **\*\*Python\*\***: 3.8以上（3.12推奨）

- **\*\*pip\*\***: 最新版推奨

**### 7.2 環境設定**

**#### 7.2.1 環境変数**

```bash

# Google Gemini API設定

GEMINI\_API\_KEY=your\_gemini\_api\_key\_here

GEMINI\_MODEL=gemini-1.5-flash

# BERTモデル設定

BERT\_MODEL\_NAME=cl-tohoku/bert-base-japanese-v3

BERT\_MAX\_LENGTH=512

# ネットワーク設定

REQUEST\_TIMEOUT\_SEC=30

CONNECT\_TIMEOUT\_SEC=5

# ログ設定

LOG\_LEVEL=INFO

# 論争解析設定

MAX\_TOPICS=10

MIN\_CONFIDENCE\_THRESHOLD=0.7

```

**#### 7.2.2 セットアップ手順**

1. 依存関係インストール

```bash

pip install -r requirements.txt

```

2. 環境変数設定

```bash

cp env.example .env

# .envファイルを編集

```

3. アプリケーション起動

```bash

python -m app.main

# または

uvicorn app.main:app --host 0.0.0.0 --port 8082

```

**### 7.3 パフォーマンス特性**

**#### 7.3.1 処理時間**

- **\*\*BERT分類\*\***: 1発言あたり約100-200ms

- **\*\*Gemini API\*\***: 1リクエストあたり約1-3秒

- **\*\*全体処理\*\***: 5発言で約3-5秒

**#### 7.3.2 メモリ使用量**

- **\*\*BERTモデル\*\***: 約1.5GB

- **\*\*アプリケーション\*\***: 約500MB

- **\*\*合計\*\***: 約2GB

**## 8. 結合時の注意点**

**### 8.1 WP2-1との統合**

**#### 8.1.1 API設計の一貫性**

- **\*\*レスポンス形式\*\***: WP2-1と同じ`ApiResponse`形式を使用

- **\*\*エラーハンドリング\*\***: 統一されたエラーコード体系

- **\*\*ログ形式\*\***: 同じログフォーマット

**#### 8.1.2 設定管理**

- **\*\*環境変数\*\***: WP2-1と共通の設定項目

- **\*\*設定ファイル\*\***: 同じ`.env`形式

- **\*\*ログレベル\*\***: 統一されたログ設定

**### 8.2 WP2-3（可視化モジュール）との統合**

**#### 8.2.1 データ形式**

- **\*\*JSON構造\*\***: 可視化に適した構造化データ

- **\*\*論点ID\*\***: 一意な識別子で論点を特定

- **\*\*関係データ\*\***: 対立強度と関係タイプを含む

**#### 8.2.2 出力データの活用**

```json

{

"topics": [

{

"topic\_id": "topic\_1",

"topic\_name": "論点名",

"confidence": 0.92,

"keywords": ["キーワード1", "キーワード2"]

}

],

"relations": [

{

"topic": "論点名",

"a\_position": "賛成",

"b\_position": "反対",

"relation\_type": "対立",

"intensity": 0.8

}

]

}

```

**### 8.3 システム統合時の考慮事項**

**#### 8.3.1 ネットワーク設定**

- **\*\*ポート\*\***: 8082（WP2-1は8081）

- **\*\*CORS\*\***: フロントエンドからのアクセス許可

- **\*\*タイムアウト\*\***: 適切なタイムアウト設定

**#### 8.3.2 エラーハンドリング**

- **\*\*API失敗時\*\***: フォールバック処理

- **\*\*モデル読み込み失敗\*\***: エラーメッセージとログ出力

- **\*\*入力検証\*\***: 適切なバリデーション

**#### 8.3.3 セキュリティ**

- **\*\*APIキー管理\*\***: 環境変数での管理

- **\*\*入力サニタイゼーション\*\***: 悪意のある入力の検出

- **\*\*レート制限\*\***: API呼び出し制限

**### 8.4 運用時の注意点**

**#### 8.4.1 監視**

- **\*\*ヘルスチェック\*\***: `/health`エンドポイントの監視

- **\*\*ログ監視\*\***: エラーログの監視

- **\*\*パフォーマンス\*\***: レスポンス時間の監視

**#### 8.4.2 スケーリング**

- **\*\*水平スケーリング\*\***: 複数インスタンスでの負荷分散

- **\*\*キャッシュ\*\***: 頻繁に使用されるデータのキャッシュ

- **\*\*非同期処理\*\***: 長時間処理の非同期化

**#### 8.4.3 メンテナンス**

- **\*\*モデル更新\*\***: BERTモデルの定期更新

- **\*\*依存関係更新\*\***: セキュリティパッチの適用

- **\*\*ログローテーション\*\***: ログファイルの管理

**## 9. トラブルシューティング**

**### 9.1 よくあるエラー**

**#### 9.1.1 APIキーエラー**

```

AppError: GEMINI\_API\_KEY is not set

```

**\*\*解決方法\*\***: 環境変数`GEMINI\_API\_KEY`を設定

**#### 9.1.2 モデル読み込みエラー**

```

AppError: BERT\_MODEL\_ERROR

```

**\*\*解決方法\*\***: インターネット接続確認、モデル名確認

**#### 9.1.3 タイムアウトエラー**

```

AppError: GEMINI\_TIMEOUT

```

**\*\*解決方法\*\***: ネットワーク接続確認、タイムアウト設定調整

**### 9.2 デバッグ方法**

**#### 9.2.1 ログレベル設定**

```bash

LOG\_LEVEL=DEBUG

```

**#### 9.2.2 ヘルスチェック**

```bash

curl http://localhost:8082/health

```

**#### 9.2.3 テストリクエスト**

```bash

curl -X POST "http://localhost:8082/v1/analyze" \

-H "Content-Type: application/json" \

-d '{"messages": [{"speaker": "A", "text": "テスト"}]}'

```

**## 10. 今後の拡張予定**

**### 10.1 機能拡張**

- **\*\*感情分析\*\***: 発言の感情分析機能

- **\*\*多言語対応\*\***: 英語・中国語等の対応

- **\*\*リアルタイム解析\*\***: WebSocket対応

- **\*\*カスタムモデル\*\***: ドメイン特化モデルの学習

**### 10.2 パフォーマンス改善**

- **\*\*GPU対応\*\***: CUDA GPUでの高速化

- **\*\*キャッシュ機能\*\***: Redis等でのキャッシュ

- **\*\*並列処理\*\***: 複数リクエストの並列処理

- **\*\*モデル最適化\*\***: 軽量化モデルの使用

**### 10.3 運用改善**

- **\*\*監視機能\*\***: Prometheus/Grafana対応

- **\*\*ログ分析\*\***: ELK Stack対応

- **\*\*自動スケーリング\*\***: Kubernetes対応

- **\*\*CI/CD\*\***: GitHub Actions対応

---

この技術文書により、知識がない人でもWP2-2論争解析モジュールの全体像を理解し、適切に運用・拡張できるようになります。