CogniMail-AI: Code Reference Guide - AI Components

==========

1. Natural Language Processing Core

```python
class NLPCore:
    def process_command(self, user_input: str) -> Intent:
        # Convert user input to system intent
        tokens = self.tokenize(user_input)
        entities = self.extract_entities(tokens)
        return Intent(
            command_type=self.classify_intent(tokens),
            entities=entities,
            context=self.context_manager.get_current_context()
        )

    def extract_entities(self, tokens: List[str]) -> Dict[str, Any]:
        # Extract email addresses, dates, times, subjects, etc.
        return {
            'emails': self.extract_email_addresses(tokens),
            'dates': self.extract_dates(tokens),
            'times': self.extract_times(tokens),
            'subjects': self.extract_subjects(tokens)
        }
```

2. AI Task Scheduler

```python
class AIScheduler:
    def optimize_schedule(self, tasks: List[Task]) -> Schedule:
        # Prioritize and schedule tasks based on AI analysis
        priorities = self.priority_analyzer.analyze(tasks)
        time_slots = self.time_slot_optimizer.find_optimal_slots(tasks)
        return self.create_optimized_schedule(tasks, priorities, time_slots)

    def predict_meeting_duration(self, meeting: Meeting) -> timedelta:
        # Predict optimal meeting duration based on participants and agenda
        return self.duration_predictor.predict(
            participants=meeting.participants,
            agenda=meeting.agenda,
            historical_data=self.meeting_history
        )
```

3. Email Intelligence

```python
class EmailAI:
    def categorize_email(self, email: Email) -> Category:
        # Categorize emails using ML model
        features = self.feature_extractor.extract(email)
        return self.classifier.predict(features)

    def generate_response(self, email: Email) -> str:
```

```
        # Generate smart email response
        context = self.context_analyzer.analyze(email)
        tone = self.tone_analyzer.detect_tone(email)
        return self.response_generator.generate(
            context=context,
            tone=tone,
            user_style=self.user_preferences
        )
```

4. Context Management

```python
class ContextManager:
    def update_context(self, new_data: Dict[str, Any]) -> None:
        # Update conversation context using ML
        self.current_context = self.context_model.merge(
            old_context=self.current_context,
            new_data=new_data,
            importance=self.calculate_importance(new_data)
        )

    def predict_next_action(self) -> Action:
        # Predict user's next likely action
        return self.action_predictor.predict(
            context=self.current_context,
            user_history=self.user_behavior_log
        )
```

```

5. Smart Notification System

---

```python
class NotificationAI:
    def should_notify(self, event: Event) -> bool:
        # Determine if and when to notify using ML
        importance = self.importance_scorer.score(event)
        user_state = self.user_state_detector.detect()
        return self.notification_model.decide(
            importance=importance,
            user_state=user_state,
            time_context=self.time_analyzer.analyze()
        )
```

Key AI Features:

â€¢ Natural Language Understanding
  - Intent classification
  - Entity extraction
  - Context awareness

â€¢ Smart Scheduling
  - Meeting duration prediction
  - Optimal time slot selection
  - Priority-based scheduling
```

â€¢ Email Intelligence

- Smart categorization
- Response generation
- Tone analysis

â€¢ Context Management

- Continuous learning
- Action prediction
- Behavior analysis

â€¢ Notification Intelligence

- Importance scoring
- User state detection
- Optimal timing

Implementation Notes:

- All AI models use transfer learning from pre-trained language models
- Regular model updates based on user feedback and behavior
- Privacy-first approach with local model fine-tuning
- Modular design for easy model swapping and updates
- Extensive use of async processing for real-time responsiveness