# Meta Agent - Analytical Intelligence Engine

## 🎯 Tổng quan

**Meta Agent** là analytical brain của hệ thống MAS-Planning, chuyên về task analysis, reasoning, và strategic thinking. Nó đóng vai trò như một senior consultant, phân tích sâu các tasks phức tạp và cung cấp insights để guide execution process.

## 🏗️ Kiến trúc và Positioning

graph TB  
 subgraph "Meta Agent System"  
 A[Task Input] --> B[Meta Expert Engine]  
 B --> C[XML Analysis]  
 C --> D[Task Decomposition]  
   
 D --> E[Agent Identification]  
 D --> F[Task Structure]  
 D --> G[Tool Requirements]  
   
 E --> H[Expert Agent Mapping]  
 F --> I[Dependency Analysis]  
 G --> J[Resource Planning]  
   
 H --> K[Strategic Recommendations]  
 I --> K  
 J --> K  
   
 K --> L[Final Analysis Output]  
 end  
   
 subgraph "Integration Layer"  
 M[Plan Agent] --> A  
 N[Tool Agent] --> A  
 L --> O[Execution Guidance]  
 L --> P[Decision Support]  
 end  
   
 style B fill:#e8f5e8  
 style C fill:#f3e5f5  
 style D fill:#e1f5fe  
 style K fill:#fff3e0

## 🎯 Core Responsibilities

### **1. Task Analysis & Decomposition**

* **Complex Task Breakdown**: Phân tích tasks phức tạp thành sub-components
* **Dependency Mapping**: Identify relationships giữa các task elements
* **Context Understanding**: Deep comprehension của task context và implications
* **Strategic Planning**: Provide strategic guidance cho task execution

### **2. Expert Agent Identification**

* **Agent Capability Mapping**: Determine best agent cho specific tasks
* **Resource Requirements**: Analyze tools và resources needed
* **Skill Matching**: Match task requirements với agent capabilities
* **Workflow Optimization**: Suggest optimal execution pathways

### **3. Intelligent Reasoning**

* **Strategic Analysis**: High-level strategic thinking về tasks
* **Problem Solving**: Creative solutions cho complex challenges
* **Risk Assessment**: Identify potential issues và mitigation strategies
* **Quality Assurance**: Ensure task execution quality

## 🧠 Core Architecture

### **1. Meta Expert Engine**

def meta\_expert(self, state: AgentState):  
 """Core reasoning engine"""  
 # Build context từ conversation history và current state  
 messages = convert\_messages\_list(state['messages'])  
   
 # LLM analysis với advanced prompting  
 llm\_response = self.llm.invoke(messages)  
   
 # Extract structured data từ XML response  
 agent\_data = extract\_from\_xml(llm\_response.content)  
   
 # Process analysis results  
 name = agent\_data.get('Agent Name')  
 description = agent\_data.get('Agent Description')  
 tasks = agent\_data.get('Tasks')  
 tool = agent\_data.get('Tool')  
 answer = agent\_data.get('Answer')  
   
 return {  
 \*\*state,  
 'agent\_data': agent\_data,  
 'messages': [HumanMessage(self.format\_response(agent\_data))]  
 }

**Analysis Components:**

* **Agent Identification**: Determine most suitable agent type
* **Task Structure**: Break down complex tasks
* **Tool Requirements**: Specify required tools và resources
* **Strategic Guidance**: Provide high-level direction

### **2. XML Processing System**

def extract\_from\_xml(text: str) -> Dict[str, Any]:  
 """Extract structured data từ XML-formatted LLM responses"""  
 import re  
   
 # Define extraction patterns  
 patterns = {  
 'Agent Name': r'<agent\_name>(.\*?)</agent\_name>',  
 'Agent Description': r'<agent\_description>(.\*?)</agent\_description>',  
 'Tasks': r'<tasks>(.\*?)</tasks>',  
 'Tool': r'<tool>(.\*?)</tool>',  
 'Answer': r'<answer>(.\*?)</answer>'  
 }  
   
 extracted\_data = {}  
 for key, pattern in patterns.items():  
 match = re.search(pattern, text, re.DOTALL | re.IGNORECASE)  
 if match:  
 content = match.group(1).strip()  
 # Process lists nếu applicable  
 if key == 'Tasks' và '\n' in content:  
 extracted\_data[key] = [task.strip('- ').strip()   
 for task in content.split('\n')   
 if task.strip()]  
 else:  
 extracted\_data[key] = content  
   
 return extracted\_data

**XML Structure Benefits:**

* **Structured Output**: Predictable data format
* **Error Handling**: Robust parsing với fallbacks
* **Extensibility**: Easy to add new data fields
* **Validation**: Schema validation support

### **3. Intelligent Routing Logic**

def controller(self, state: AgentState):  
 """Control workflow routing based on analysis quality"""  
 if self.max\_iteration > self.iteration:  
 self.iteration += 1  
 agent\_data = state.get('agent\_data')  
   
 if agent\_data and agent\_data.get('Answer'):  
 # Complete analysis available  
 return 'Answer'  
 elif agent\_data and self.is\_analysis\_sufficient(agent\_data):  
 # Sufficient analysis for next step  
 return 'Answer'   
 else:  
 # Need more analysis  
 return 'Meta' # Continue analysis  
 else:  
 # Iteration limit reached  
 return 'Answer'  
  
def is\_analysis\_sufficient(self, agent\_data: Dict) -> bool:  
 """Determine if analysis is sufficient for execution"""  
 required\_fields = ['Agent Name', 'Tasks', 'Tool']  
 return all(agent\_data.get(field) for field in required\_fields)

## 🔄 Analysis Workflows

### **1. Task Decomposition Flow**

sequenceDiagram  
 participant P as Plan Agent  
 participant M as Meta Agent  
 participant LLM as Language Model  
 participant X as XML Parser  
   
 P->>M: Complex Task Input  
 M->>LLM: Analysis Request + Context  
 LLM->>M: XML-Structured Response  
 M->>X: Parse XML Content  
 X->>M: Structured Analysis Data  
 M->>M: Validate Analysis Quality  
 M->>P: Task Breakdown + Guidance

### **2. Strategic Analysis Flow**

sequenceDiagram  
 participant I as Input  
 participant M as Meta Agent  
 participant A as Analysis Engine  
 participant R as Reasoning Module  
 participant O as Output Formatter  
   
 I->>M: Strategic Question  
 M->>A: Context Analysis  
 A->>R: Deep Reasoning  
 R->>R: Strategic Planning  
 R->>O: Format Strategic Response  
 O->>M: Structured Guidance  
 M->>I: Strategic Recommendations

### **3. Multi-Iteration Refinement**

sequenceDiagram  
 participant M as Meta Agent  
 participant C as Controller  
 participant A as Analysis Engine  
   
 loop Until Complete or Max Iterations  
 M->>A: Perform Analysis  
 A->>M: Analysis Result  
 M->>C: Check Completeness  
 C->>C: Evaluate Quality  
 alt Analysis Complete  
 C->>M: Route to Answer  
 else Need More Analysis   
 C->>M: Continue Analysis  
 end  
 end

## 🧩 Advanced Features

### **1. Context-Aware Analysis**

def invoke(self, input\_data) -> dict:  
 """Context-aware analysis với multiple input formats"""  
 # Handle different input formats  
 if isinstance(input\_data, dict):  
 task = input\_data.get('task', '')  
 context = input\_data.get('context', '')  
 previous\_results = input\_data.get('previous\_results', [])  
   
 input\_text = f"Task: {task}\nContext: {context}"  
 if previous\_results:  
 input\_text += f"\nPrevious Results: {previous\_results}"  
 else:  
 input\_text = str(input\_data)  
   
 # Build comprehensive context  
 state = {  
 'input': input\_text,  
 'messages': [  
 SystemMessage(self.system\_prompt),  
 HumanMessage(f'User Query: {input\_text}')  
 ],  
 'output': '',  
 }

**Context Elements:**

* **Task Description**: Primary task to analyze
* **Environmental Context**: Surrounding conditions và constraints
* **Historical Data**: Previous execution results
* **User Preferences**: Learned patterns và preferences

### **2. Multi-Modal Reasoning**

class MetaAgent(BaseAgent):  
 """Advanced meta-reasoning capabilities"""  
   
 def \_\_init\_\_(self,   
 tools: list = [],   
 model: str = "gemini-2.5-pro",  
 temperature: float = 0.2,  
 max\_iteration: int = 10,  
 json\_mode: bool = False,  
 verbose: bool = False,  
 llm=None):  
   
 # Reasoning modes  
 self.reasoning\_modes = {  
 'analytical': self.analytical\_reasoning,  
 'creative': self.creative\_reasoning,  
 'strategic': self.strategic\_reasoning,  
 'tactical': self.tactical\_reasoning  
 }

**Reasoning Modes:**

* **Analytical**: Data-driven, logical analysis
* **Creative**: Innovative solutions và alternatives
* **Strategic**: Long-term planning và optimization
* **Tactical**: Immediate execution guidance

### **3. Dynamic Prompting System**

META\_PROMPT = """  
You are a Meta-Analysis Expert trong smart home automation system.  
  
CORE COMPETENCIES:  
1. Task Analysis & Decomposition  
2. Strategic Planning & Reasoning   
3. Agent Capability Assessment  
4. Resource Optimization  
5. Risk Assessment & Mitigation  
  
ANALYSIS FRAMEWORK:  
- Break down complex tasks into manageable components  
- Identify dependencies và execution order  
- Recommend appropriate tools và agents  
- Provide strategic guidance và alternatives  
  
OUTPUT FORMAT:  
Always respond in structured XML format:  
  
<agent\_name>  
[Best agent cho this task]  
</agent\_name>  
  
<agent\_description>  
[Why this agent is optimal]  
</agent\_description>  
  
<tasks>  
[Detailed task breakdown]  
- Task 1: Description  
- Task 2: Description  
</tasks>  
  
<tool>  
[Required tools và resources]  
</tool>  
  
<answer>  
[Strategic guidance và recommendations]  
</answer>  
  
Focus on actionable insights và practical execution guidance.  
"""

## 🔧 Technical Implementation

### **1. LangGraph Integration**

def create\_graph(self):  
 """Create analysis workflow graph"""  
 graph = StateGraph(AgentState)  
   
 # Core analysis node  
 graph.add\_node('Meta', self.meta\_expert)  
   
 # Final answer processing  
 graph.add\_node('Answer', self.final)  
   
 # Entry point  
 graph.set\_entry\_point('Meta')  
   
 # Conditional routing cho iterative refinement  
 graph.add\_conditional\_edges('Meta', self.controller)  
   
 # Terminal node  
 graph.add\_edge('Answer', END)  
   
 return graph.compile(debug=False)

### **2. State Management**

class AgentState(TypedDict):  
 input: str  
 messages: List[Message]  
 agent\_data: Dict[str, Any]  
 output: str  
 iteration\_count: int  
 analysis\_quality: float  
 confidence\_score: float

**State Components:**

* **Input Processing**: Raw input và formatted queries
* **Message History**: Conversation context và analysis chain
* **Analysis Data**: Structured results từ XML parsing
* **Quality Metrics**: Analysis confidence và completeness scores

### **3. Error Handling & Recovery**

def robust\_xml\_extraction(self, text: str) -> Dict[str, Any]:  
 """Robust XML extraction với multiple fallback strategies"""  
 try:  
 # Primary XML extraction  
 return self.extract\_from\_xml(text)  
 except XMLParseError:  
 # Fallback to regex patterns  
 return self.regex\_fallback\_extraction(text)  
 except Exception as e:  
 # Final fallback to keyword extraction  
 logger.warning(f"XML extraction failed: {e}")  
 return self.keyword\_fallback\_extraction(text)  
  
def fallback\_response\_generation(self, input\_text: str) -> Dict[str, Any]:  
 """Generate fallback response when analysis fails"""  
 return {  
 'Agent Name': 'General Assistant',  
 'Agent Description': 'Fallback analysis due to processing error',  
 'Tasks': [f'Process request: {input\_text[:100]}...'],  
 'Tool': 'Standard tools',  
 'Answer': 'Analysis completed with fallback method'  
 }

## 📊 Performance Optimizations

### **1. Caching Strategy**

* **Analysis Cache**: Cache similar task analyses
* **Pattern Recognition**: Reuse analysis patterns
* **Context Compression**: Efficient context storage
* **LLM Response Cache**: Cache structured responses

### **2. Iteration Optimization**

def optimize\_iteration\_strategy(self, state: AgentState) -> str:  
 """Optimize số lượng iterations needed"""  
 analysis\_quality = self.assess\_analysis\_quality(state)  
   
 if analysis\_quality > 0.9:  
 # High quality, complete early  
 return 'Answer'  
 elif analysis\_quality > 0.7 and self.iteration > 2:  
 # Good enough quality after some iterations  
 return 'Answer'  
 elif self.iteration >= self.max\_iteration:  
 # Prevent infinite loops  
 return 'Answer'  
 else:  
 # Continue analysis  
 return 'Meta'

### **3. Resource Management**

* **Memory Optimization**: Efficient state management
* **CPU Utilization**: Optimized parsing algorithms
* **LLM Call Optimization**: Minimize unnecessary API calls
* **Response Size Management**: Control output verbosity

## 🔒 Security Features

### **1. Input Sanitization**

def sanitize\_input(self, input\_text: str) -> str:  
 """Sanitize input to prevent injection attacks"""  
 # Remove potentially harmful content  
 sanitized = self.remove\_script\_tags(input\_text)  
 sanitized = self.escape\_special\_characters(sanitized)  
 sanitized = self.limit\_input\_length(sanitized)  
   
 return sanitized

### **2. Output Validation**

* **Schema Validation**: Ensure output conforms to expected structure
* **Content Filtering**: Filter potentially harmful recommendations
* **Privilege Checking**: Validate recommended actions against permissions
* **Audit Logging**: Comprehensive logging của analysis decisions

### **3. Safe Analysis Practices**

* **Sandboxed Execution**: Isolated analysis environment
* **Rate Limiting**: Prevent analysis abuse
* **Access Control**: Role-based analysis permissions
* **Data Privacy**: Anonymize sensitive information trong logs

## 📈 Analytics và Monitoring

### **1. Analysis Quality Metrics**

def assess\_analysis\_quality(self, state: AgentState) -> float:  
 """Assess quality của current analysis"""  
 agent\_data = state.get('agent\_data', {})  
   
 # Completeness score  
 required\_fields = ['Agent Name', 'Tasks', 'Tool', 'Answer']  
 completeness = sum(1 for field in required\_fields   
 if agent\_data.get(field)) / len(required\_fields)  
   
 # Detail score  
 detail\_score = self.calculate\_detail\_score(agent\_data)  
   
 # Coherence score  
 coherence\_score = self.calculate\_coherence\_score(agent\_data)  
   
 return (completeness + detail\_score + coherence\_score) / 3

### **2. Performance Tracking**

* **Response Time**: Analysis processing duration
* **Accuracy Metrics**: Correctness của agent recommendations
* **User Satisfaction**: Implicit feedback từ subsequent actions
* **Iteration Efficiency**: Optimal iterations per analysis

### **3. Learning Analytics**

* **Pattern Recognition**: Common analysis patterns
* **Success Predictors**: Factors leading to successful analyses
* **Failure Analysis**: Understanding analysis failures
* **Improvement Opportunities**: Areas cho enhancement

## 🚀 Advanced Capabilities

### **1. Domain-Specific Analysis**

def domain\_specific\_analysis(self, task: str, domain: str) -> Dict:  
 """Specialized analysis cho specific domains"""  
 domain\_handlers = {  
 'security': self.security\_analysis,  
 'energy': self.energy\_analysis,  
 'convenience': self.convenience\_analysis,  
 'maintenance': self.maintenance\_analysis  
 }  
   
 handler = domain\_handlers.get(domain, self.general\_analysis)  
 return handler(task)

### **2. Predictive Analysis**

* **Outcome Prediction**: Predict task execution outcomes
* **Resource Forecasting**: Estimate resource requirements
* **Risk Assessment**: Identify potential execution risks
* **Success Probability**: Calculate execution success likelihood

### **3. Collaborative Analysis**

* **Multi-Agent Consultation**: Collaborate với other agents
* **Expert System Integration**: Leverage external expertise
* **Crowd-Sourced Insights**: Incorporate community knowledge
* **Continuous Learning**: Adapt based on execution feedback

## 🎯 Integration Patterns

### **1. Plan Agent Integration**

# Plan Agent calls Meta Agent for task analysis  
meta\_input = {  
 "input": task,  
 "context": f"This is task {i} of {len(selected\_plan)} from {plan\_type}",  
 "previous\_results": execution\_results[-3:]  
}  
  
meta\_result = self.meta\_agent.invoke(meta\_input)

### **2. Tool Agent Collaboration**

* **Tool Recommendation**: Suggest optimal tools cho tasks
* **Execution Guidance**: Provide tactical execution advice
* **Error Recovery**: Assist trong troubleshooting failures
* **Performance Optimization**: Optimize tool usage patterns

### **3. Real-Time Advisory**

* **Live Consultation**: Real-time analysis during execution
* **Dynamic Adjustment**: Adjust strategies based on current state
* **Immediate Feedback**: Provide instant recommendations
* **Continuous Monitoring**: Ongoing analysis của execution progress

*Meta Agent represents the analytical intelligence của MAS-Planning system, ensuring mọi task được approached với strategic thinking và deep understanding. Nó là bridge giữa high-level planning và concrete execution, providing the wisdom needed cho successful smart home automation.*