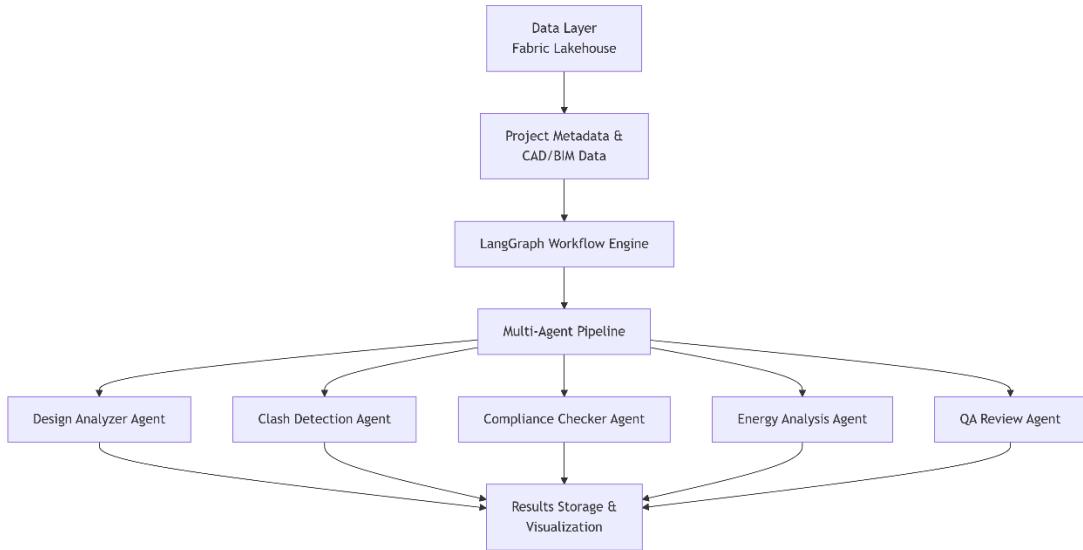


# LangGraph Multi-Agent CAD/BIM Automation System - Technical Documentation

## System Overview

This system implements a **multi-agent workflow** using **LangGraph** and **LangChain** to automate CAD/BIM design analysis for Mechanical & Electrical (M&E) engineering projects. The system processes building design data through 5 specialized AI agents that perform sequential analysis and quality assessment.

## Architecture Diagram



## Core Components

### 1. State Management (AgentState)

```
class AgentState(TypedDict):
    # Project Identification
    project_id: str
    project_name: str
    project_data: Dict[str, Any]
    design_intent: str

    # Agent Results (NEW - Proper state management)
    design_analyzer_result: Dict[str, Any]
    clash_detector_result: Dict[str, Any]
    compliance_checker_result: Dict[str, Any]
    energy_analyst_result: Dict[str, Any]
    qa_reviewer_result: Dict[str, Any]

    # Legacy Fields (Backward compatibility)
    design_analysis: Dict[str, Any]
    clash_analysis: Dict[str, Any]
    compliance_check: Dict[str, Any]
    energy_analysis: Dict[str, Any]
    qa_report: Dict[str, Any]

    # Workflow Tracking
    current_agent: str
    agent_progress: List[str]
    processing_times: Dict[str, float]
    errors: List[str]
    final_report: Dict[str, Any]
```

## Significance:

- **TypedDict** ensures type safety and IDE support
- **Explicit state structure** prevents runtime errors
- **Dual field system** maintains backward compatibility while fixing state persistence issues
- **Comprehensive tracking** enables monitoring and debugging

## 2. Base Agent Class (MEAgen)

class MEAgent:

```
def __init__(self, name: str, system_prompt: str, output_parser=None):
    self.name = name
    self.system_prompt = system_prompt
    self.llm = llm
    self.output_parser = output_parser
    self.result_key = f"{name.lower().replace(' ', '_')}_result"
```

### Key Methods:

#### invoke() - Core Agent Execution

```
def invoke(self, state: AgentState) -> Dict[str, Any]:
    # Build prompt with project data and context
    prompt_content = self._build_prompt(state)
    messages = [
        SystemMessage(content=self.system_prompt),
        HumanMessage(content=prompt_content)
    ]

    # LLM API Call
    response = self.llm.invoke(messages)

    # Parse JSON response
    result = self._parse_response(response.content)

    # Return state updates
    return {
        self.result_key: result, # Primary result storage
        f"{self.name.lower().split()[0]}_analysis": result, # Legacy field
        "current_agent": self.name,
        "processing_times": {...},
        "agent_progress": [...]
    }
```

## Significance:

- **Standardized interface** for all agents
- **Automatic state key generation** ensures consistency
- **Context-aware prompting** includes previous agent results
- **Robust error handling** prevents workflow failures

#### \_parse\_response() - JSON Response Handling

```
def _parse_response(self, response_text: str) -> Dict[str, Any]:
    # Multiple parsing strategies:
    # 1. Markdown code blocks ('`json`)
    # 2. Raw JSON extraction
    # 3. Fallback to text analysis
```

## Significance:

- **Multiple fallback strategies** handle various LLM response formats
- **Error recovery** ensures workflow continues even with malformed responses
- **Flexible parsing** accommodates different LLM behaviors

## 3. Specialized Agent Classes

Each agent inherits from MEAgent with domain-specific system prompts:

### Design Analyzer Agent

```
system_prompt = """You are a Senior M&E Design Engineer. Analyze CAD/BIM designs for:
```

- Design completeness and technical integrity
- Missing systems and documentation gaps
- Constructability and installation issues
- Technical risk assessment

Return JSON with: completeness\_score, technical\_risks, recommendations"""

### **Clash Detection Agent**

system\_prompt = """You are a BIM Coordination Specialist. Detect spatial clashes between M&E systems. Identify coordination issues and provide resolution strategies.

Return JSON with: total\_clashes, critical\_clashes, clash\_locations, resolution\_strategies"""

### **Compliance Checker Agent**

system\_prompt = """You are a Singapore M&E Compliance Expert. Verify compliance with:

- BCA Green Mark 2021
- SS530 Energy Efficiency
- SS564 Plumbing Standards
- CP52 Electrical Installations

Return JSON with: compliance\_score, violations\_found, certification\_impact"""

### **Energy Analysis Agent**

system\_prompt = """You are an Energy Efficiency Specialist. Analyze:

- Energy consumption optimization opportunities
- Green Mark certification potential
- Cost-saving opportunities
- Equipment efficiency assessment

Return JSON with: savings\_potential, improvement\_measures, payback\_period\_years"""

### **QA Review Agent**

system\_prompt = """You are a Quality Assurance Lead. Provide final quality assessment:

- Overall quality score (0-100)
- Critical issues identification
- Go/No-Go recommendation
- Executive summary for management

Return JSON with: overall\_score, critical\_issues, final\_recommendation, executive\_summary"""

## **LangGraph Workflow Engine**

### **Workflow Construction**

# Initialize State Graph

```
workflow = StateGraph(AgentState)
```

# Add Nodes (Agent Functions)

```
workflow.add_node("design_analyzer", design_analyzer_node)
workflow.add_node("clash_detector", clash_detector_node)
workflow.add_node("compliance_checker", compliance_checker_node)
workflow.add_node("energy_analyst", energy_analyst_node)
workflow.add_node("qa_reviewer", qa_reviewer_node)
workflow.add_node("finalizer", finalizer_node)
```

# Define Execution Flow

```
workflow.set_entry_point("design_analyzer")
workflow.add_edge("design_analyzer", "clash_detector")
workflow.add_edge("clash_detector", "compliance_checker")
workflow.add_edge("compliance_checker", "energy_analyst")
workflow.add_edge("energy_analyst", "qa_reviewer")
workflow.add_edge("qa_reviewer", "finalizer")
```

```

workflow.add_edge("finalizer", END)

# Compile Graph
app = workflow.compile()

```

### LangGraph Significance:

- **Directed Acyclic Graph (DAG)** structure ensures proper execution order
- **State persistence** maintains data across agent executions
- **Compilation** optimizes execution and enables checkpointing
- **Visualization** capabilities for debugging and monitoring

### Node Functions

Each node wraps an agent's invoke method and adds debugging:

```

def qa_reviewer_node(state: AgentState):
    result = qa_agent.invoke(state)
    print(f" 📁 QA Reviewer returning keys: {list(result.keys())}")
    if 'qa_reviewer_result' in result and result['qa_reviewer_result']:
        qa_data = result['qa_reviewer_result']
        if isinstance(qa_data, dict) and 'overall_score' in qa_data:
            print(f" ⚡ QA Reviewer has overall_score: {qa_data['overall_score']}")
    return result

```

### Finalizer Node

Compiles all agent results into a comprehensive report:

```

def finalizer_node(state: AgentState):
    final_report = {
        "project_info": {...},
        "agent_analyses": {
            "design_analysis": state.get("design_analyzer_result", {}),
            "clash_analysis": state.get("clash_detector_result", {}),
            # ... other agents
        },
        "performance_metrics": {...}
    }
    return {"final_report": final_report}

```

## Data Flow & State Transitions

### Initial State

```

initial_state = AgentState(
    project_id="WL-FABRIC-2024-001",
    project_name="Commercial Tower- Stone Lake",
    project_data={}, # Loaded from Lakehouse
    design_intent="Certified Green Mark",
    # All result fields initialized as empty dicts
    design_analyzer_result={},
    clash_detector_result={},
    # ... other fields
)

```

### State Evolution Through Workflow

1. **Design Analyzer Node:**
  - Input: initial\_state
  - Output: design\_analyzer\_result populated
  - State: {..., "design\_analyzer\_result": {"completeness\_score": 85, ...}}
2. **Clash Detector Node:**
  - Input: State from Design Analyzer
  - Output: clash\_detector\_result populated
  - State: {..., "clash\_detector\_result": {"total\_clashes": 12, ...}}

3. **Compliance Checker Node:**
  - Input: State with design and clash results
  - Output: compliance\_checker\_result populated
  - **Context Building:** Previous agent results included in prompt
4. **Energy Analyst Node:**
  - Input: State with all previous results
  - Output: energy\_analyst\_result populated
5. **QA Reviewer Node:**
  - Input: Complete state with all technical analyses
  - Output: qa\_reviewer\_result with final assessment
  - Contains: overall\_score, final\_recommendation
6. **Finalizer Node:**
  - Input: Complete analyzed state
  - Output: Compiled final\_report
  - Saves: All results to Lakehouse

## Data Integration Layer

### Lakehouse Data Loading

```
def load_project_data_from_lakehouse(project_id=None):
    # Spark SQL queries to Fabric Lakehouse
    project_df = spark.sql("SELECT * FROM project_metadata")
    cad_df = spark.sql("SELECT * FROM cad_bim_data")

    # Merge and parse JSON fields
    merged_data = []
    for project in projects_pd.iterrows():
        # Complex data merging logic
        full_project_data = merge_project_cad_data(project, cad_data)
        merged_data.append(full_project_data)
```

### Significance:

- **Unified data access** from Fabric Lakehouse
- **Automatic JSON parsing** for complex CAD/BIM data
- **Project-data correlation** ensures correct data relationships

### Results Storage

```
def save_results_to_lakehouse(state: AgentState):
    for agent_name, results in agents_to_save:
        result_data = [
            "project_id": state["project_id"],
            "analysis_timestamp": datetime.now().isoformat(),
            "agent_name": agent_name,
            "results_json": json.dumps(results),
            "processing_time": processing_time
        ]
    # Save to appropriate Lakehouse table
```

## Key Design Patterns

### 1. Strategy Pattern - Agent Specialization

Each agent implements the same interface but with different expertise and prompts.

### 2. Chain of Responsibility - Sequential Processing

Agents process data in a specific order, each building on previous results.

### 3. Observer Pattern - State Monitoring

Debugging output monitors state changes at each step.

### 4. Template Method - Agent Execution

Base MEAgent class defines the execution template, specialized agents fill in details.

## Configuration & Optimization

### Cost Optimization

```
# Use cheaper model for testing
llm = ChatOpenAI(
    model="gpt-3.5-turbo", # Instead of gpt-4
    temperature=0.1,      # Lower temperature for consistent results
    openai_api_key=OPENAI_API_KEY
)
```

```
# Process limited projects for testing
projects_to_process = projects_data[:1] # Only first project
```

### Performance Monitoring

- Processing times tracked per agent
- Error collection for failure analysis
- Progress tracking for workflow monitoring

## Execution Flow in Detail

### 1. Data Preparation

- Load project metadata and CAD/BIM data from Lakehouse
- Merge and parse complex JSON structures
- Initialize LangGraph state with project context

### 2. Sequential Agent Execution

Design Analyzer → Clash Detector → Compliance Checker → Energy Analyst → QA Reviewer

Each agent:

- Receives current state with previous results
- Processes using domain-specific expertise
- Returns updated state with new analysis
- Maintains context for subsequent agents

### 3. Result Compilation & Storage

- Finalizer node aggregates all agent results
- Comprehensive report generation
- Persistent storage to Lakehouse tables
- Performance metrics collection

## Key Innovations

### 1. State Management Solution

Solved the critical issue of agent result persistence by:

- Explicit state field definitions in TypedDict
- Consistent key naming conventions (\*\_result)
- Dual-field system for backward compatibility

### 2. Robust JSON Parsing

Multiple fallback strategies handle various LLM response formats:

- Markdown code block extraction
- Raw JSON parsing
- Text analysis fallback

### 3. Context-Aware Agent Prompts

Each agent receives relevant context from previous analyses, enabling:

- Cumulative intelligence building
- Cross-agent dependency resolution
- Comprehensive final assessment

#### 4. Production-Ready Architecture

- Error handling and recovery
- Performance monitoring
- Scalable data integration
- Cost-effective execution

This documentation provides a comprehensive understanding of the LangGraph multi-agent system's architecture, implementation details, and the significance of each component in creating a robust CAD/BIM automation workflow.

Our current implementation uses in-memory state management for simplicity in the demo. However, LangGraph provides enterprise-grade checkpointing with **SqliteSaver**, which we can easily integrate for production deployment. This would give us fault tolerance- if a workflow fails mid-execution, we can resume from the last checkpoint instead of starting over.

#### Outputs

```
📄 CHECKING EXISTING DATA IN LAKEHOUSE
=====
✅ Project Metadata: 5 records
✅ CAD/BIM Data: 5 records

📊 SAMPLE PROJECTS:


| ABC project_id       | ABC project_name                     | ABC building_type   | ABC green_mark_target |
|----------------------|--------------------------------------|---------------------|-----------------------|
| 1 WL-FABRIC-2024-004 | Industrial Building - Jessica Bridge | Industrial Building | Certified             |
| 2 WL-FABRIC-2024-003 | Mixed Development - Charles River    | Mixed Development   | Certified             |
| 3 WL-FABRIC-2024-001 | Commercial Tower - Stone Lake        | Commercial Tower    | Certified             |
| 4 WL-FABRIC-2024-002 | Commercial Tower - Stephen Circle    | Commercial Tower    | Silver                |
| 5 WL-FABRIC-2024-005 | Residential Condo - Jamie Points     | Residential Condo   | Certified             |



🚀 INITIALIZING LANGGRAPH MULTI-AGENT SYSTEM...
✅ LangGraph Multi-Agent System initialized!
🤖 Available Agents: Design Analyzer, Clash Detector, Compliance Checker, Energy Analyst, QA Reviewer
🔧 Using model: gpt-3.5-turbo (cost-effective)

🏗 BUILDING LANGGRAPH WORKFLOW...
✅ LangGraph Workflow Built Successfully!
📝 Workflow Structure: Design → Clash → Compliance → Energy → QA → Final

⌚ STARTING LANGGRAPH MULTI-AGENT WORKFLOW
=====
This will:
1. 📄 Load project data from Lakehouse
2. 🤖 Run 5 specialized AI agents using LangGraph
3. 📈 Generate comprehensive analysis reports
4. 📁 Save all results back to Lakehouse
🔥 COST-SAVING: Processing only 1 project with GPT-3.5-turbo
=====

🚀 Executing LangGraph Multi-Agent workflow...
✅ Loaded 5 projects from Lakehouse
📝 Found 5 projects to analyze
🔥 COST-SAVING: Processing only 1 project instead of 5
=====

📝 PROCESSING PROJECT 1/1
📄 Name: Industrial Building - Jessica Bridge
>ID ID: WL-FABRIC-2024-004
⌚ Target: Certified Green Mark
🏗 Type: Industrial Building
📍 Location: Jurong East
```

```

✖ STARTING LANGGRAPH WORKFLOW...
[Design Analyzer] Processing...
✓ Design Analyzer completed in 5.01s
⚠ Design Analyzer returning keys: ['design_analyzer_result', 'design_analysis', 'current_agent', 'processing_times', 'agent_progress']
[Clash Detector] Processing...
✓ Clash Detector completed in 3.69s
⚠ Clash Detector returning keys: ['clash_detector_result', 'clash_analysis', 'current_agent', 'processing_times', 'agent_progress']
[Compliance Checker] Processing...
✓ Compliance Checker completed in 4.45s
⚠ Compliance Checker returning keys: ['compliance_checker_result', 'compliance_analysis', 'current_agent', 'processing_times', 'agent_progress']
[Energy Analyst] Processing...
✓ Energy Analyst completed in 4.08s
⚠ Energy Analyst returning keys: ['energy_analyst_result', 'energy_analysis', 'current_agent', 'processing_times', 'agent_progress']
[QA Reviewer] Processing...
✓ QA Reviewer completed in 2.36s
⚠ QA Reviewer returning keys: ['qa_reviewer_result', 'qa_analysis', 'current_agent', 'processing_times', 'agent_progress']
🎯 QA Reviewer has overall_score: 85
[Finalizer] Compiling final report...
🔍 Finalizer - All result fields:
design_analyzer_result: <class 'dict'> - keys: ['completeness_score', 'technical_risks', 'recommendations']
clash_detector_result: <class 'dict'> - keys: ['total_clashes', 'critical_clashes', 'clash_locations', 'resolution_strategies']
compliance_checker_result: <class 'dict'> - keys: ['compliance_score', 'violations_found', 'certification_impact']
energy_analyst_result: <class 'dict'> - keys: ['savings_potential', 'improvement_measures', 'payback_period_years']
qa_reviewer_result: <class 'dict'> - keys: ['overall_score', 'critical_issues', 'final_recommendation', 'executive_summary']
🎯 FOUND overall_score: 85
✓ Workflow completed in 19.58s
💾 All agent results saved to Lakehouse tables
FINAL SCORE: 85/100
✓ Recommendation: Go
⌚ Total Time: 19.58s
🤖 Agents Completed: 5
=====
```

🎉 LANGGRAPH WORKFLOW COMPLETED!

✅ Processed 1 project successfully!

💻 Project 1: Industrial Building - Jessica Bridge

📊 Quality Score: 85/100

✓ Recommendation: Go

⌚ Processing Time: 19.58s

💰 COST SAVINGS ACHIEVED:

- Used GPT-3.5-turbo instead of GPT-4 ( $\approx 10x$  cheaper)
- Processed 1 project instead of 5 (5x cheaper)
- Total cost reduction:  $\approx 50x$  cheaper than full run

🎯 QA FINAL REPORTS SUMMARY:

	ABC project_id	ABC quality_score	ABC recommendation
1	WL-FABRIC-2024-004	85	Go
2	WL-FABRIC-2024-004	85	Go
3	WL-FABRIC-2024-004	79	APPROVED with conditions
4	WL-FABRIC-2024-005	78	PROCEED with improvements
5	WL-FABRIC-2024-003	88	PROCEED with improvements
6	WL-FABRIC-2024-001	82	REVIEW REQUIRED
7	WL-FABRIC-2024-002	89	REVIEW REQUIRED

💡 LANGGRAPH WORKFLOW STRUCTURE:

Workflow: Design Analyzer → Clash Detector → Compliance Checker → Energy Analyst → QA Reviewer → Finalizer

✓ Graph-based workflow with proper state management and error handling

🎉 LANGGRAPH MULTI-AGENT SYSTEM IMPLEMENTATION COMPLETED!