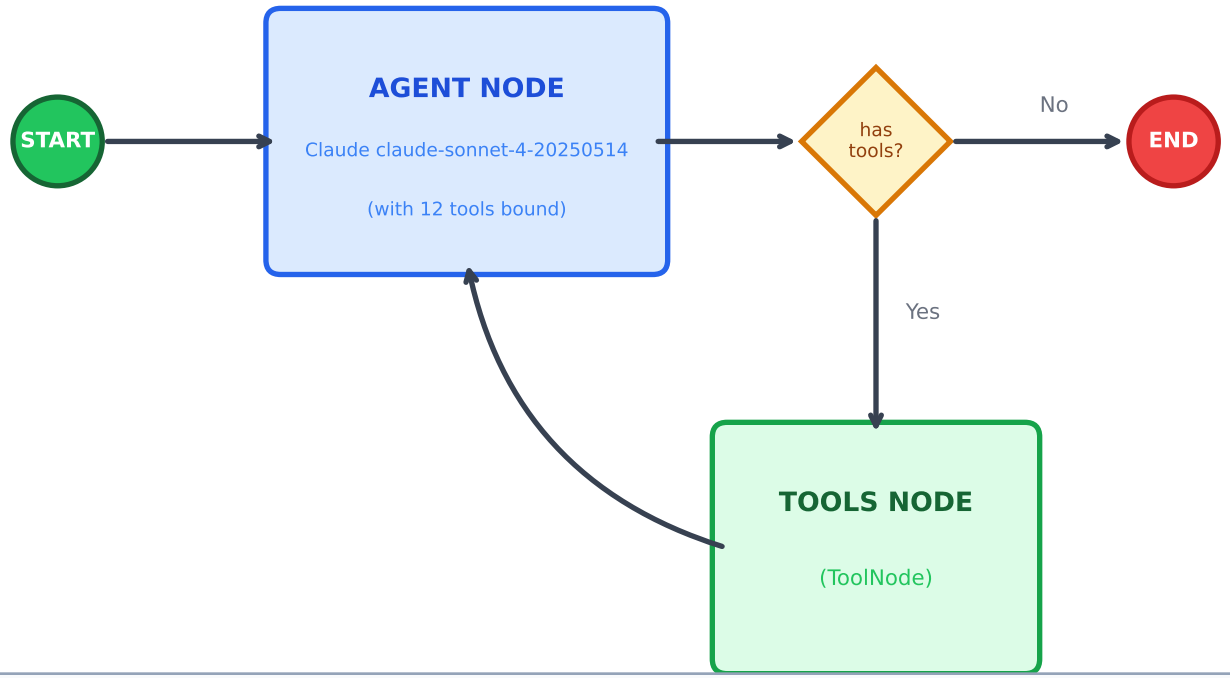
ISR Mission Planning Agent

Architecture Documentation

```
SYSTEM OVERVIEW

The ISR (Intelligence, Surveillance, Reconnaissance) Mission Planning system uses
a SINGLE LangGraph-based agent with 12 specialized tools to plan multi-drone
reconnaissance missions.

AGENT COUNT: 1 Main Agent

The system follows a ReAct (Reasoning + Acting) pattern:
  1. Agent receives user query + environment context
  2. Agent reasons about what tool to call
  3. Tool executes and returns result
  4. Agent reasons again (may call more tools)
  5. Agent produces final response with ROUTE_Dx outputs

KEY COMPONENTS:
  - 1 LangGraph StateGraph workflow
  - 1 LLM node (Claude claude-sonnet-4-20250514)
  - 1 ToolNode (with 12 tools)
  - Conditional edge for tool/end routing
```

LangGraph Workflow Architecture



Workflow Execution:

1. User query enters at START 2. AGENT NODE calls LLM 3. Conditional edge checks tool_calls
4. YES -> TOOLS NODE executes 5. Loop back to AGENT 6. NO -> END (return response)

Agent Tools (12 Total)

PLANNING TOOLS (9)

`get_mission_overview`	Get all airports, targets, SAMs, drone configs
`get_drone_info`	Get detailed constraints for a specific drone
`get_distance`	Look up distance between two waypoints
`calculate_route_fuel`	Calculate total fuel for a drone route
`validate_drone_route`	Validate route against all constraints
`find_accessible_targets`	Find targets drone can reach, by efficiency
`suggest_drone_route`	Generate greedy baseline route for drone
`check_target_conflicts`	Check for duplicate target assignments
`get_mission_summary`	Get summary stats for multi-drone plan

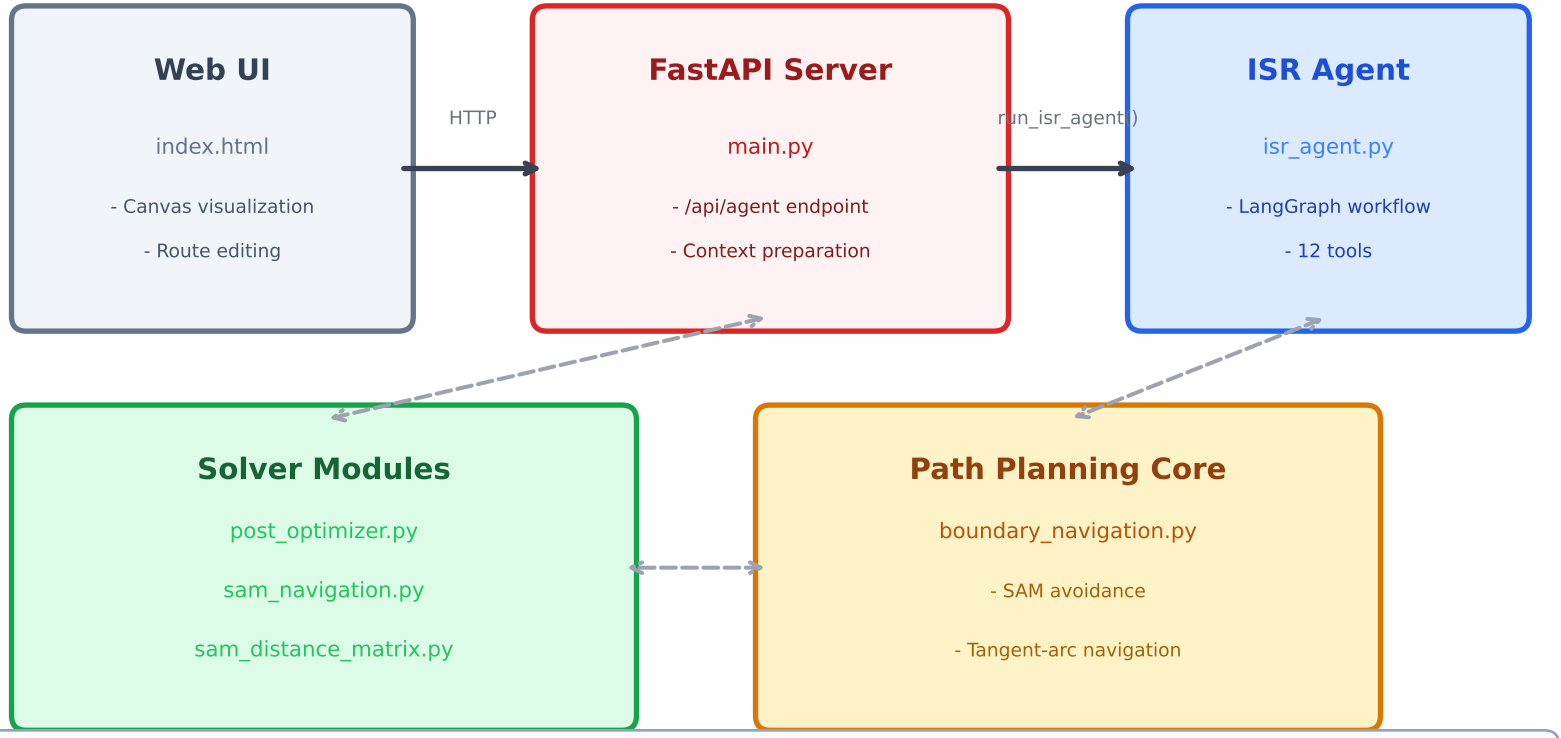
OPTIMIZATION TOOLS (3)

`optimize_assign_unvisited`	Insert unvisited targets into routes
`optimize_reassign_targets`	Swap targets to closer drone trajectories
`optimize_remove_crossings`	Fix self-crossing trajectories (2-opt)

Tool Input/Output Pattern:

```
All tools follow a consistent pattern:
  - Input: Tool-specific parameters (drone_id, waypoints, routes dict, etc.)
  - Output: Formatted string with results

Tools access shared context via global variables:
  _current_env        Environment with airports, targets, SAMs
  _drone_configs      Per-drone configuration (fuel, airports, access)
  _distance_matrix    Pre-computed SAM-aware distances
  _sam_paths          Pre-computed SAM-avoiding paths
```

System Integration & Data Flow



Data Flow:

1. User interacts with Web UI (draws, edits, clicks "Run Agent")
2. Web UI sends POST to /api/agent with environment + query
3. FastAPI prepares context and calls run_isr_agent()
4. Agent uses tools which access Solver modules for optimization
5. Solver uses Path Planning Core for SAM-aware distances
6. Agent returns ROUTE_Dx formatted routes -> Web UI renders on canvas