Invention Disclosure: Derivux for MATLAB

1. Administrative Information

Field	Value
Title	Derivux for MATLAB: Workspace-Aware AI Integration with Secure Code Execution
Inventor	Sebastian Hondl
Date of Conception	December 16, 2025
Status	Working Prototype (Private Repository)
Prior Public Disclosure	None

University Affiliation

Field	Value
Institution	University of Minnesota
Department	Mechanical Engineering
Development Context	Personal project, not university research

Independent Development Declaration

This invention was developed **independently** using:

- Personal computer and personal time
- Skills from coursework (not research assistantship)
- University-provided MATLAB license (standard student resource, available to all enrolled students)

Not used: Research funding, lab facilities, research staff, faculty collaboration, specialized equipment. The MATLAB license is a standard educational resource (like library access or campus WiFi), distinct from research-specific resources that might trigger university IP claims.

---

2. Summary of Invention

Derivux integrates Claude AI into MATLAB/Simulink with: 1. **Workspace Awareness** — AI sees live variables, types, and values 2. **Secure Execution** — Sandbox blocks dangerous operations before running AI-generated code 3. **Simulink Manipulation** — Programmatic model creation, modification, and layout optimization 4. **Visual Capture** — Plots and diagrams displayed inline in chat 5. **Multi-Session Isolation** — Independent contexts per chat tab 6. **Agent Routing** — Pattern-based routing to specialized agents

---

3. Technical Architecture

3.1 Three-Layer System Design

Layer 1: MATLAB Layer

Component	Role
DerivuxApp	Main orchestrator, UI management
SimulinkBridge	Model access and manipulation
CodeExecutor	Security validation sandbox
WorkspaceContextProvider	Real-time variable introspection

Connects via: `py.derivux.MatlabBridge()` **Layer 2: Python Bridge Layer**

Component	Role
MatlabBridge	Orchestrates Python-side operations
SpecializedAgent Router	Pattern-based agent selection
SessionContext	Per-tab conversation isolation
Persistent Async Event Loop	Thread-safe SDK client state

Connects via: Claude Agent SDK **Layer 3: AI Layer**

Component	Role
MatlabAgent	Claude SDK client wrapper
MCP Server	In-process tool server

Tool Registry	matlab_, *simulink_*, file_* tools

Connects to: Claude API

3.2 Data Flow

Step	Component	Description
1	Chat UI	User enters message
2	webwindow msg	JavaScript postMessage to MATLAB
3	ChatUIController	MATLAB class handles UI events
4	MatlabBridge	`start_async_message()` routes to agent
5	MatlabAgent	`asyncio.run_coroutine_threadsafe()`
6	MCP Tool Handler	Tool calls (`mcp__matlab__*`)
7	MATLAB Engine	Code execution via `matlab.engine` API

3.3 Communication Protocol

Challenge: MATLAB is single-threaded and synchronous; Claude SDK requires async Python with persistent state. **Solution:** Bidirectional async bridge with chunk-based streaming.

```
Python side - accumulates response chunks
```

```
class MatlabBridge:
    def start_async_message(self, message: str) -> None:
        future = asyncio.run_coroutine_threadsafe(
            self._process_message(message),
            self._loop  # Persistent event loop in background thread
        )

    def poll_async_chunks(self) -> List[str]:
        with self._async_lock:
            chunks = self._async_chunks.copy()
            self._async_chunks = []
            return chunks
```

```
% MATLAB side - polls for updates without blocking UI
while ~obj.PythonBridge.is_async_complete()
    chunks = obj.PythonBridge.poll_async_chunks();
    for i = 1:length(chunks)
        obj.appendToResponse(string(chunks{i}));
    end
    drawnow;  % Keep UI responsive
    pause(0.05);
end
```

---

4. Novel Claims with Technical Detail

Claim 1: Bidirectional Async Communication Bridge

Problem: MATLAB's execution model is synchronous and single-threaded. Claude SDK requires async Python with persistent client state for conversation memory. Spawning a new process per request loses context. **Solution:** A persistent Python event loop running in a background thread maintains the SDK client across multiple MATLAB calls. **Implementation (`bridge.py:124-186`):**

```
def _start_persistent_loop(self) -> None:
    """Start persistent event loop in background thread."""
    def run_loop():
        self._loop = asyncio.new_event_loop()
        asyncio.set_event_loop(self._loop)
        self._loop.run_forever()

    self._loop_thread = threading.Thread(target=run_loop, daemon=True)
    self._loop_thread.start()

    # Wait for loop to be ready
    while self._loop is None or not self._loop.is_running():
        time.sleep(0.01)
```

Key innovation: Thread-safe future management allows synchronous MATLAB calls to interact with async Python without blocking, while preserving conversation state across requests.

Claim 2: Real-Time Workspace Introspection

Problem: AI assistants cannot see user's MATLAB workspace, leading to generic suggestions that ignore actual variable state. **Solution:** Type-aware extraction that formats each variable appropriately for AI context, with intelligent truncation for large data. **Implementation (`WorkspaceContextProvider.m:31-120`):**

```
function context = getWorkspaceContext(obj)
    vars = evalin('base', 'whos');
    [~, idx] = sort([vars.bytes]);  % Smaller variables first
    vars = vars(idx);

    for i = 1:length(vars)
        v = vars(i);
        if v.bytes > 1e7, continue; end  % Skip >10MB

        switch v.class
            case 'double'
                if numel(val) == 1
                    str = sprintf('%s = %.6g', v.name, val);
                elseif numel(val) <= 10
                    str = sprintf('%s = [%s]', v.name, num2str(val'));
                else
                    str = sprintf('%s: %s double, range [%.3g, %.3g]', ...
                        v.name, mat2str(v.size), min(val(:)), max(val(:)));
                end
            case 'struct'
                str = sprintf('%s: struct with fields {%s}', ...
                    v.name, strjoin(fieldnames(val), ', '));
            % ... other types
        end
    end
end
```

Key innovation: Priority ordering (smaller variables first) ensures the most useful context fits within token limits. Type-specific formatting preserves semantic meaning (e.g., showing struct field names rather than raw data).

Claim 3: Security-Sandboxed Code Execution

Problem: AI-generated code may contain dangerous operations—file deletion, system commands, eval chains—that could harm the user's system if executed blindly. **Solution:** Domain-specific validation that blocks dangerous MATLAB operations while allowing full engineering functionality. **Implementation (`CodeExecutor.m:21-27, 111-175`):**

```
properties (Constant)
    BLOCKED_FUNCTIONS = {'system', 'dos', 'unix', 'perl', 'python', '!', ...
                         'eval', 'evalin', 'evalc', 'feval', 'builtin', ...
                         'delete', 'rmdir', 'movefile', 'copyfile', ...
                         'java.lang.Runtime', 'py.os', 'py.subprocess', ...
                         'webread', 'webwrite', 'websave', 'ftp', ...
                         'NET.', 'COM.'}

    BLOCKED_PATTERNS = {'^\s*!',           % Shell escape
                        'java\.lang\.',    % Java runtime
                        'py\.',            % Python escape
                        'NET\.',           % .NET escape
                        'COM\.'}           % COM escape
end

function [isValid, reason] = validateCode(obj, code)
    % Check blocked functions
    for i = 1:length(obj.BLOCKED_FUNCTIONS)
        funcName = obj.BLOCKED_FUNCTIONS{i};
        pattern = sprintf('\\b%s\\s*[\\(\\.]?', ...
            regexptranslate('escape', funcName));
        if ~isempty(regexp(code, pattern, 'once'))
            isValid = false;
            reason = sprintf('Blocked: %s', funcName);
            return;
        end
    end

    % Check blocked patterns
    for i = 1:length(obj.BLOCKED_PATTERNS)
        if ~isempty(regexp(code, obj.BLOCKED_PATTERNS{i}, 'once'))
            isValid = false;
            reason = 'Blocked pattern detected';
            return;
        end
    end

    isValid = true;
end
```

Execution modes:

- `prompt`: Ask user before each execution (safest)
- `auto`: Execute if validation passes (balanced)
- `bypass`: Expert mode, no validation (power users)

Key innovation: Unlike OS-level sandboxing (which would break legitimate MATLAB operations), this uses domain knowledge to block only dangerous patterns while allowing full engineering functionality.

Claim 4: Simulink Layout Algorithm

Problem: Simulink's built-in auto-arrange produces cluttered diagrams. Engineers spend significant time manually arranging blocks for readability. **Solution:** A five-phase graph layout algorithm optimized for control system conventions. **Implementation (`SimulinkLayoutEngine.m`, 807 lines):**
Phase 1: Graph Extraction

```
function buildGraph(obj)
    blocks = find_system(obj.Model, 'SearchDepth', 1, 'Type', 'block');
    lines = find_system(obj.Model, 'FindAll', 'on', 'Type', 'line');

    for i = 1:length(lines)
        srcHandle = get_param(lines(i), 'SrcBlockHandle');
        dstHandle = get_param(lines(i), 'DstBlockHandle');
        obj.AdjacencyList{srcIdx}(end+1) = dstIdx;
    end
end
```

Phase 2: Layer Assignment (Longest-Path Algorithm)

```
function assignLayers(obj)
    sources = obj.findSourceBlocks();  % Blocks with no inputs
    obj.LayerAssignment = zeros(1, obj.NumBlocks);

    queue = sources;
    while ~isempty(queue)
        current = queue(1); queue(1) = [];
        currentLayer = obj.LayerAssignment(current);

        for succ = obj.AdjacencyList{current}
            newLayer = currentLayer + 1;
            if newLayer > obj.LayerAssignment(succ)
                obj.LayerAssignment(succ) = newLayer;
                queue(end+1) = succ;
            end
        end
    end
end
```

Phase 3: Crossing Minimization (Barycenter Heuristic)

```
function minimizeCrossings(obj)
    for sweep = 1:obj.MaxSweeps
        % Forward sweep
        for layer = 2:obj.NumLayers
            for block = obj.BlocksInLayer{layer}
                predecessors = obj.getPredecessors(block);
                if ~isempty(predecessors)
                    obj.Barycenter(block) = mean(obj.Position(predecessors));
                end
            end
            obj.sortLayerByBarycenter(layer);
        end

        % Backward sweep (similar, using successors)
    end
end
```

Phase 4: Coordinate Assignment

```
function assignCoordinates(obj)
    x = obj.Margin;
    for layer = 1:obj.NumLayers
        y = obj.Margin;
        for block = obj.BlocksInLayer{layer}
            obj.BlockPosition(block, :) = [x, y, x+obj.BlockWidth, y+obj.BlockHeight];
            y = y + obj.BlockHeight + obj.VerticalSpacing;
        end
        x = x + obj.BlockWidth + obj.HorizontalSpacing;
    end
end
```

Phase 5: Orthogonal Wire Routing

```
function routeWires(obj)
    for edge = obj.Edges
        srcPort = obj.getOutputPortPosition(edge.src);
        dstPort = obj.getInputPortPosition(edge.dst);

        % Create orthogonal route with 90-degree angles
        midX = (srcPort(1) + dstPort(1)) / 2;
        points = [srcPort; midX, srcPort(2); midX, dstPort(2); dstPort];

        set_param(edge.line, 'Points', points);
    end
end
```

Key innovation: The algorithm respects control system conventions (inputs left, outputs right, signal flow left-to-right) and produces publication-quality diagrams with minimal crossings and clean 90-degree wire angles.

Claim 5: Multi-Session Context Isolation

Problem: Multiple chat tabs should maintain independent conversation contexts. Existing tools either share context (cross-contamination) or spawn separate processes (high resource usage).
Solution: Per-session context storage within a single process, with efficient session switching.
Implementation (`bridge.py:1414-1516`):

```
@dataclass
class SessionContext:
    tab_id: str
    messages: List[Dict[str, Any]] = field(default_factory=list)
    created_at: float = field(default_factory=time.time)
    last_active_at: float = field(default_factory=time.time)
    agent_config: Optional[str] = None

class MatlabBridge:
    def __init__(self):
        self._session_contexts: Dict[str, SessionContext] = {}
        self._active_session_id: Optional[str] = None

    def switch_session_context(self, tab_id: str) -> None:
        # Save current session state
        if self._active_session_id:
            old = self._session_contexts[self._active_session_id]
            old.last_active_at = time.time()

        # Create or activate target session
        if tab_id not in self._session_contexts:
            self._session_contexts[tab_id] = SessionContext(tab_id=tab_id)

        self._active_session_id = tab_id

    def get_session_messages(self, tab_id: str) -> List[Dict]:
        if tab_id in self._session_contexts:
            return self._session_contexts[tab_id].messages
        return []
```

Key innovation: Single-process efficiency with full context isolation. Sessions can be suspended and resumed without losing state, and inactive sessions can be cleaned up based on age.

Claim 6: Pattern-Based Agent Routing

Problem: Different tasks (git operations, code review, Simulink work) benefit from specialized agents with focused prompts and restricted tools. Users shouldn't need to manually select agents.
Solution: Confidence-based routing that analyzes user messages to select the best-fit agent.
Implementation (`specialized_agent.py:61-96`):

```
@dataclass
class AgentConfig:
    name: str
    command_prefix: str  # e.g., "/git"
    system_prompt: str
    allowed_tools: List[str]
    auto_detect_patterns: List[str] = field(default_factory=list)
    priority: int = 100

    def calculate_confidence(self, message: str) -> float:
        if not self.auto_detect_patterns:
            return 0.0

        message_lower = message.lower()
        matches = 0

        for pattern in self.auto_detect_patterns:
            if re.search(pattern, message_lower, re.IGNORECASE):
                matches += 1

        if matches == 0:
            return 0.0

        # Scoring: first match = 0.5, each additional = +0.15
        score = 0.5 + (matches - 1) * 0.15
        return min(1.0, score)
```

Routing logic:

```
def route_message(self, message: str) -> AgentConfig:
    # 1. Check explicit commands first
    for agent in self.agents:
        if message.startswith(agent.command_prefix):
            return agent

    # 2. Auto-detect based on confidence
    best_agent = None
    best_score = 0.0
    for agent in self.agents:
        score = agent.calculate_confidence(message)
        if score > best_score and score >= 0.6:
            best_score = score
            best_agent = agent

    return best_agent or self.default_agent
```

Key innovation: Explicit commands (`/git status`) override auto-detection. Confidence threshold (0.6) prevents false routing. Priority field breaks ties between equally confident agents.

---

5. Prior Art Differentiation

Feature	MATLAB Copilot	General AI (ChatGPT)	**Derivux**
Workspace awareness	Limited	None	Full real-time
Code execution	None	None	Secure sandbox
Simulink manipulation	None	None	Programmatic
Inline figures	None	None	Yes
Session persistence	N/A	Web-based	MATLAB-integrated
Agent specialization	None	None	Pattern-based routing

---

6. Key Implementation Files

Component	File	Lines
Orchestrator	`toolbox/+derivux/DerivuxApp.m`	385
Security	`toolbox/+derivux/CodeExecutor.m`	256
Simulink Bridge	`toolbox/+derivux/SimulinkBridge.m`	537
Layout Engine	`toolbox/+derivux/SimulinkLayoutEngine.m`	807
Workspace Context	`toolbox/+derivux/WorkspaceContextProvider.m`	380
Python Bridge	`python/derivux/bridge.py`	1516
AI Agent	`python/derivux/agent.py`	590
Agent Routing	`python/derivux/agents/specialized_agent.py`	265
MATLAB Tools	`python/derivux/matlab_tools.py`	451
Simulink Tools	`python/derivux/simulink_tools.py`	469

Total: ~8,000 lines across MATLAB, Python, and JavaScript

Version 3.0 — January 27, 2026