ASSIGNMENT 3:
INSURANCE POLICY & CLAIMS AGENT

Kevin Geidel
MSDS 442: AI Agent Design & Development
Northwestern University
May 18, 2025

Requirement 1: Graph the agent with LangChain/LangGraph

The construction of the agent and accompanying graph begins with the creation of the functions that serve as edges in our graph (see cell 3 in the appendix). The actual assembly of the graph itself occurs in cell 9. However, some of the components, such as the conditional edge, `verify_policy`, and **ClaimState** class, are built above. Following the logic in cell 9 we first instantiate an empty graph:

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

The `workflow` object has `add_node` and `add_edge` methods that allow us to assemble the components created in cells 3-8. The output is displayed graphically in cell 10 (reproduced in figure 1 below.)

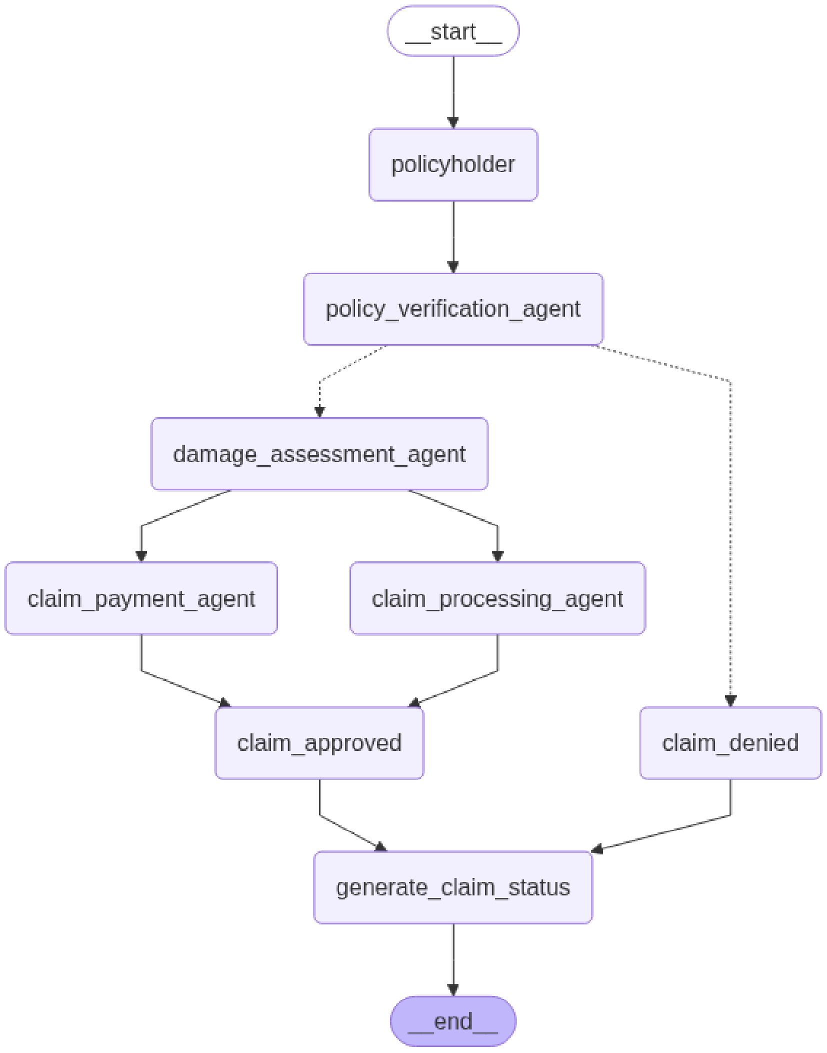


Figure 1: Graphical depiction of the Allstate insurance policy and claims agent.

Requirement 2: Establish desired process workflow

To meet the assignment objectives our agent must follow the provided workflow (shown in figure 2.) Careful comparison of figures 1 and 2 show that each agent is represented as a node in the graph, decision formulated as a conditional and each function constructed as an edge. The workflow (depicted in *Business Process Model Notion* or *BPMN*) was provided to us for this assignment. However, construction of such a graphic is key for capturing all desired functionality and provides a map for the developer as they assemble the AI agents. In the figure we see each agent as its own swim lane and place the various decisions and actions into their respective spheres. Each bullet point in the requirements appears somewhere in this visual and acts as a checklist for the developer.

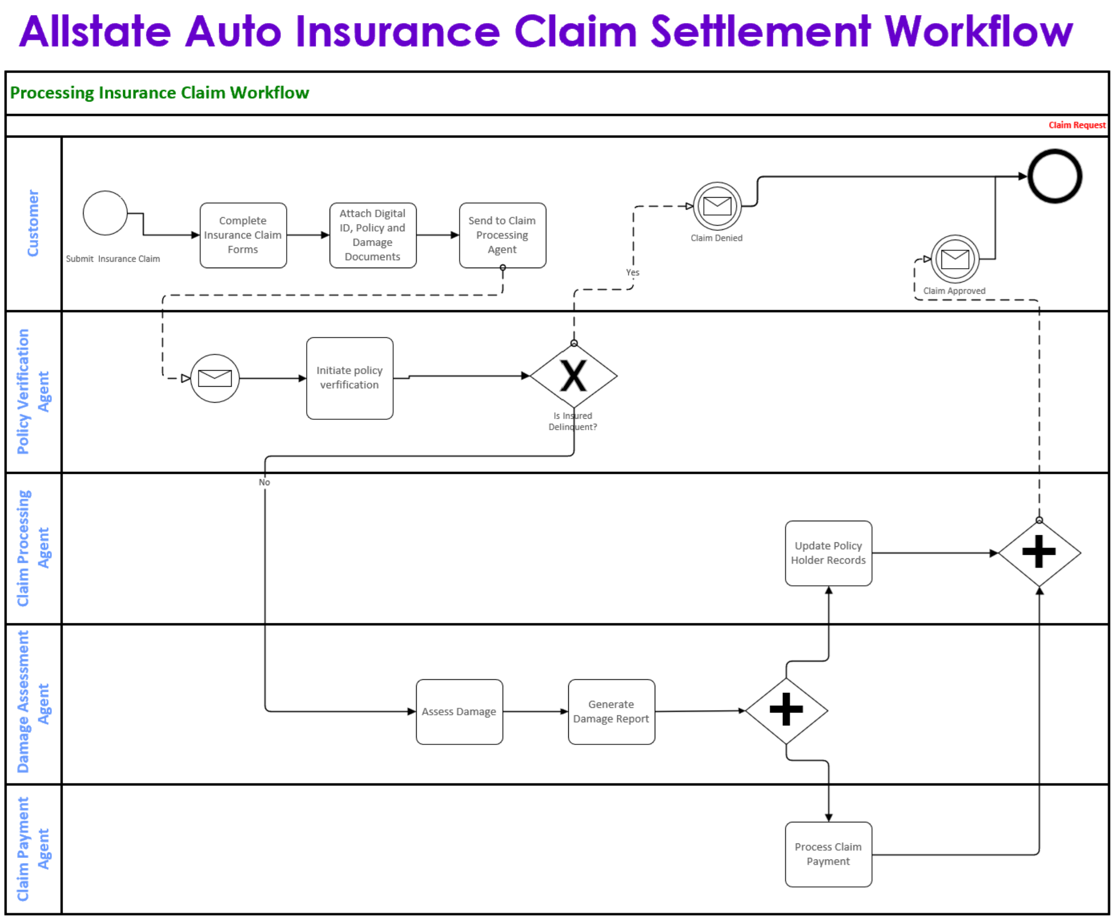

Allstate Auto Insurance Claim Settlement Workflow
Processing Insurance Claim Workflow
Claim Request
Customer
Submit Insurance Claim
Complete Insurance Claim Forms
Attach Digital ID, Policy and Damage Documents
Send to Claim Processing Agent
Claim Denied
Yes
Claim Approved
Policy Verification Agent
Initiate policy verification
Is Insured Delinquent?
No
Claim Processing Agent
Update Policy Holder Records
Damage Assessment Agent
Assess Damage
Generate Damage Report
Claim Payment Agent
Process Claim Payment

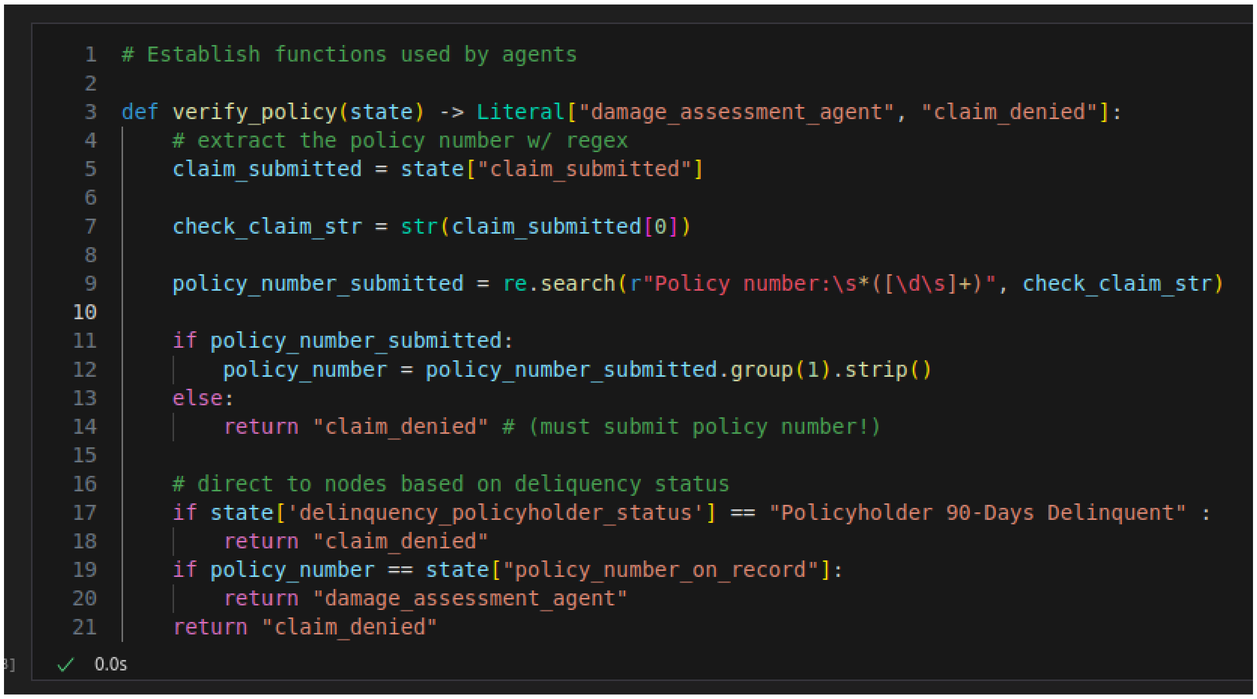
Figure 2: Desired agent workflow in *Business Process Model Notation* (BPMN).

Requirement 3: Prepare for expected user input

The user will provide their policy number and a photograph of the damage to their car. In preparation for this multi-modal inputs we build utilities for handling and displaying files in pdf format (cell 2.) A user's policy is loaded in cell 4 for use by the `policy_verification_agent`. In cell 5 we demonstrate the loading and display of an example vehicle damage photo. The trials in requirement 7 demonstrate the agent's ability to receive these inputs and perform the desired analysis.

Requirement 4: The policy verification agent

The details on the policy pdf must be extracted and analyzed to ensure the customer had a valid policy that is current. The `policy_verification_agent` will examine these files and locate information required to make this decision. The core logic in the verification agent begins in cell 3 (reproduced below in cell 3).

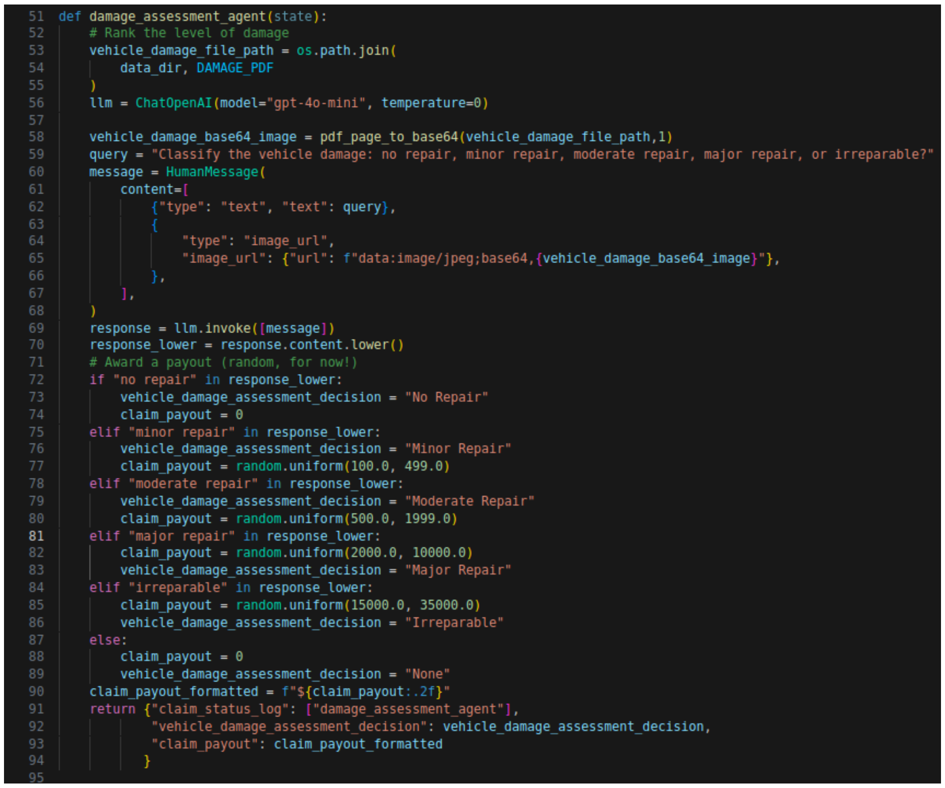


```python
# Establish functions used by agents

def verify_policy(state) -> Literal["damage_assessment_agent", "claim_denied"]:
    # extract the policy number w/ regex
    claim_submitted = state["claim_submitted"]

    check_claim_str = str(claim_submitted[0])

    policy_number_submitted = re.search(r"Policy number:\s*([\d\s]+)", check_claim_str)

    if policy_number_submitted:
        policy_number = policy_number_submitted.group(1).strip()
    else:
        return "claim_denied" # (must submit policy number!)

    # direct to nodes based on deliquency status
    if state['delinquency_policyholder_status'] == "Policyholder 90-Days Delinquent" :
        return "claim_denied"
    if policy_number == state["policy_number_on_record"]:
        return "damage_assessment_agent"
    return "claim_denied"
```
✓ 0.0s

Figure 3: The `policy_verification_agent`.

Requirement 5: Process damage photo

Processing and analyzing the user's photo of vehicle damage is the job of the `damage_assessment_agent`. The logic used within the node occurs in cell 8. Many nodes and edges are instantiated in this cell but in figure 4 we focus on the query that will have the LLM rank the level of damage shown in the photo.



```python
def damage_assessment_agent(state):
    # Rank the level of damage
    vehicle_damage_file_path = os.path.join(
        data_dir, DAMAGE_PDF
    )
    llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

    vehicle_damage_base64_image = pdf_page_to_base64(vehicle_damage_file_path,1)
    query = "Classify the vehicle damage: no repair, minor repair, moderate repair, major repair, or irreparable?"
    message = HumanMessage(
        content=[
            {"type": "text", "text": query},
            {
                "type": "image_url",
                "image_url": {"url": f"data:image/jpeg;base64,{vehicle_damage_base64_image}"},
            },
        ],
    )
    response = llm.invoke([message])
    response_lower = response.content.lower()
    # Award a payout (random, for now!)
    if "no repair" in response_lower:
        vehicle_damage_assessment_decision = "No Repair"
        claim_payout = 0
    elif "minor repair" in response_lower:
        vehicle_damage_assessment_decision = "Minor Repair"
        claim_payout = random.uniform(100.0, 499.0)
    elif "moderate repair" in response_lower:
        vehicle_damage_assessment_decision = "Moderate Repair"
        claim_payout = random.uniform(500.0, 1999.0)
    elif "major repair" in response_lower:
        claim_payout = random.uniform(2000.0, 10000.0)
        vehicle_damage_assessment_decision = "Major Repair"
    elif "irreparable" in response_lower:
        claim_payout = random.uniform(15000.0, 35000.0)
        vehicle_damage_assessment_decision = "Irreparable"
    else:
        claim_payout = 0
        vehicle_damage_assessment_decision = "None"
    claim_payout_formatted = f"${claim_payout:.2f}"
    return {"claim_status_log": ["damage_assessment_agent"],
            "vehicle_damage_assessment_decision": vehicle_damage_assessment_decision,
            "claim_payout": claim_payout_formatted
           }
```

Figure 4: The `damage_assessment_agent`.

Requirement 6: The claim processing agent

The `claim_processing_agent` ensures that the decision letter is stamped with the proper dates. Assignment requirements described a seven day delay between approval and disbursement (see cell 8.)

Requirement 7: Trial runs

The agents were tested under several conditions. Each scenario is described with a comment at the top of their respective cell. The output is the Markdown formatted decision letter that is returned to the user. The queries and their outputs are shown in cells 12 through 16.