1. **Rear-End Collision Rules**

def analyze\_rear\_end\_collision(accident\_data):

# Rule 1: Moving vehicle striking stationary vehicle from behind

if (accident\_data['vehicle\_1\_moving'] == 'No' and

accident\_data['vehicle\_2\_moving'] == 'Yes' and

'rear' in accident\_data['impact\_points'].lower()):

# Check for unusual circumstances that might shift fault

if 'sudden stop' in accident\_data['description'].lower():

return {

"primary\_fault": "Vehicle 1",

"fault\_percentage": 30,

"secondary\_fault": "Vehicle 2",

"secondary\_percentage": 70,

"reasoning": "While Vehicle 2 struck Vehicle 1 from behind, Vehicle 1 made an unexpected and potentially dangerous sudden stop."

}

else:

return {

"primary\_fault": "Vehicle 2",

"fault\_percentage": 100,

"reasoning": "Vehicle 2 struck a properly stationary Vehicle 1 from behind, indicating failure to maintain safe following distance or appropriate speed."

}

# Rule 2: Chain-reaction collisions

if "chain" in accident\_data['description'].lower() or "pushed into" in accident\_data['description'].lower():

# Identify the initiating vehicle if possible

if len(accident\_data['vehicles']) > 2: # Multi-vehicle scenario

return {

"primary\_fault": "Vehicle 3", # Typically the rear-most vehicle

"fault\_percentage": 100,

"reasoning": "Vehicle 3 initiated a chain-reaction collision that pushed Vehicle 2 into Vehicle 1."

}

1. Lane Change and Merging Incidents

def analyze\_lane\_change(accident\_data):

# Rule 1: Vehicle changing lanes has primary duty of care

if ("lane change" in accident\_data['description'].lower() or

"merging" in accident\_data['description'].lower()):

# Determine which vehicle was changing lanes

changing\_vehicle = identify\_lane\_changing\_vehicle(accident\_data['description'])

# Check if the other vehicle contributed to the incident

if "speed" in accident\_data['description'].lower() and changing\_vehicle == "Vehicle 2":

return {

"primary\_fault": "Vehicle 2",

"fault\_percentage": 80,

"secondary\_fault": "Vehicle 1",

"secondary\_percentage": 20,

"reasoning": "Vehicle 2 failed to ensure safe lane change, though Vehicle 1's speed was a contributing factor."

}

else:

return {

"primary\_fault": changing\_vehicle,

"fault\_percentage": 100,

"reasoning": f"{changing\_vehicle} initiated an unsafe lane change without ensuring adequate clearance."

}

1. Autonomous System Disengagement Rules

def analyze\_autonomous\_disengagement(accident\_data):

# Rule 1: Determine if autonomous disengagement occurred close to collision

if ("disengaged" in accident\_data['description'].lower() or

"took manual control" in accident\_data['description'].lower()):

# Check timing between disengagement and collision

time\_relation = extract\_disengagement\_timing(accident\_data['description'])

if "immediately before" in time\_relation or "seconds before" in time\_relation:

return {

"primary\_fault": "Vehicle 1 (AV Operator)",

"fault\_percentage": 100,

"reasoning": "The AV operator disengaged autonomous mode immediately before collision, potentially interrupting the AV's collision avoidance systems."

}

elif "while responding" in time\_relation:

return {

"primary\_fault": "Shared",

"vehicle\_1\_percentage": 40,

"vehicle\_2\_percentage": 60,

"reasoning": "The AV system began appropriate response, but human intervention during critical response timing may have compromised collision avoidance."

}

1. Intersection and Right-of-Way Rules

def analyze\_intersection(accident\_data):

# Rule 1: Traffic signal compliance

if "red light" in accident\_data['description'].lower():

if "ran" in accident\_data['description'].lower() or "failed to stop" in accident\_data['description'].lower():

# Determine which vehicle ran the red light

offending\_vehicle = extract\_signal\_violation\_vehicle(accident\_data['description'])

return {

"primary\_fault": offending\_vehicle,

"fault\_percentage": 100,

"reasoning": f"{offending\_vehicle} failed to stop at a red light, violating right-of-way rules."

}

# Rule 2: Stop sign scenarios

if "stop sign" in accident\_data['description'].lower():

# Similar logic for stop sign violations

pass

# Rule 3: Uncontrolled intersections

if "uncontrolled" in accident\_data['description'].lower() or "no traffic signal" in accident\_data['description'].lower():

# Apply right-of-way rules for uncontrolled intersections

Pass

1. Pedestrian and Cyclist Interaction Rules

def analyze\_vulnerable\_road\_user(accident\_data):

# Rule 1: Pedestrian/cyclist in crosswalk

if ("crosswalk" in accident\_data['description'].lower() and

("pedestrian" in accident\_data['description'].lower() or "cyclist" in accident\_data['description'].lower())):

# Check if pedestrian/cyclist had right-of-way

if "signal" in accident\_data['description'].lower() and "walk" in accident\_data['description'].lower():

return {

"primary\_fault": "Vehicle 1",

"fault\_percentage": 100,

"reasoning": "Vehicle failed to yield to pedestrian/cyclist with right-of-way in crosswalk."

}

# Check if pedestrian/cyclist entered suddenly

if "suddenly" in accident\_data['description'].lower() or "unexpectedly" in accident\_data['description'].lower():

return {

"primary\_fault": "Pedestrian/Cyclist",

"fault\_percentage": 70,

"secondary\_fault": "Vehicle 1",

"secondary\_percentage": 30,

"reasoning": "Pedestrian/cyclist entered crosswalk suddenly giving vehicle insufficient time to respond, though vehicle still has duty of care."

}

1. Environmental Conditions Modifier Rules

def apply\_environmental\_modifiers(fault\_determination, accident\_data):

# Rule 1: Weather reduces but doesn't eliminate fault

if any(condition in accident\_data['weather\_conditions'].lower()

for condition in ["rain", "snow", "fog", "ice"]):

# If fault is already shared, adjust percentages

if "primary\_fault" in fault\_determination and "secondary\_fault" in fault\_determination:

# Shift 10% from primary to secondary

fault\_determination["fault\_percentage"] -= 10

fault\_determination["secondary\_percentage"] += 10

fault\_determination["reasoning"] += f" Adverse weather conditions ({accident\_data['weather\_conditions']}) were a contributing factor."

# If single party at fault, create shared fault

elif fault\_determination["fault\_percentage"] == 100:

fault\_determination["fault\_percentage"] = 90

fault\_determination["secondary\_fault"] = "Environmental Conditions"

fault\_determination["secondary\_percentage"] = 10

fault\_determination["reasoning"] += f" Adverse weather conditions ({accident\_data['weather\_conditions']}) were a minor contributing factor."

return fault\_determination

1. Autonomous Vehicle Specific Rules

def apply\_av\_specific\_rules(fault\_determination, accident\_data):

# Rule 1: AV operating in autonomous mode has higher standard of care

if accident\_data['autonomous\_mode'] == 'Yes':

# Check if AV should have detected and avoided obstacle

if ("stationary" in accident\_data['description'].lower() and

accident\_data['vehicle\_1\_moving'] == 'Yes' and

"autonomous" in accident\_data['description'].lower()):

return {

"primary\_fault": "Vehicle 1 (AV)",

"fault\_percentage": 100,

"reasoning": "The autonomous system failed to detect and respond to a stationary obstacle, which is a basic requirement for autonomous operation."

}

# Rule 2: AV sensor limitations in unusual conditions

if accident\_data['autonomous\_mode'] == 'Yes' and accident\_data['weather\_conditions'] != 'Clear':

# Check if sensor limitations were likely a factor

if any(condition in accident\_data['weather\_conditions'].lower()

for condition in ["heavy rain", "snow", "dense fog"]):

# Modify existing fault determination

if fault\_determination["primary\_fault"] != "Vehicle 1 (AV)":

# If AV is not primarily at fault, add some contributory fault

fault\_determination["secondary\_fault"] = "Vehicle 1 (AV)"

fault\_determination["secondary\_percentage"] = 20

fault\_determination["fault\_percentage"] = 80

fault\_determination["reasoning"] += " The autonomous vehicle's sensors may have been compromised by adverse weather conditions."

return fault\_determination

1. Meta-Rules for Fault Determination

def apply\_fault\_determination\_meta\_rules(accident\_data):

# Apply all relevant rule sets

determinations = []

if "rear" in accident\_data['impact\_points'].lower():

determinations.append(analyze\_rear\_end\_collision(accident\_data))

if "lane change" in accident\_data['description'].lower():

determinations.append(analyze\_lane\_change(accident\_data))

if "disengaged" in accident\_data['description'].lower():

determinations.append(analyze\_autonomous\_disengagement(accident\_data))

# More rule applications...

# Now apply resolution rules

if len(determinations) > 1:

return resolve\_multiple\_determinations(determinations, accident\_data)

elif len(determinations) == 1:

result = determinations[0]

# Apply modifiers

result = apply\_environmental\_modifiers(result, accident\_data)

result = apply\_av\_specific\_rules(result, accident\_data)

return result

else:

# Default determination if no rules matched

return {

"primary\_fault": "Undetermined",

"reasoning": "Insufficient information to determine fault based on established rules."

}  
  
**Create a specialized tool for fault determination**:  
class FaultDeterminationTool(Tool):

name = "fault\_determiner"

description = "Determines fault in autonomous vehicle accidents"

async def \_run(self, accident\_data: Dict[str, Any]) -> Dict[str, Any]:

# Apply appropriate rule sets based on accident characteristics

if "rear" in accident\_data.get('impact\_points', '').lower():

determination = analyze\_rear\_end\_collision(accident\_data)

elif "lane change" in accident\_data.get('description', '').lower():

determination = analyze\_lane\_change(accident\_data)

# More rule applications...

# Apply modifiers

determination = apply\_environmental\_modifiers(determination, accident\_data)

determination = apply\_av\_specific\_rules(determination, accident\_data)

return determination

**Combine rule-based and LLM approaches**:

async def determine\_fault(agent, accident\_data):

# First apply rule-based determination

rule\_determination = await agent.run\_tool("fault\_determiner", accident\_data)

# Then use LLM to validate and possibly refine the determination

prompt = f"""

I've analyzed this autonomous vehicle accident using established traffic rules:

Accident Description: {accident\_data['description']}

Rule-Based Determination:

Primary Fault: {rule\_determination['primary\_fault']}

Fault Percentage: {rule\_determination.get('fault\_percentage', 100)}%

Reasoning: {rule\_determination['reasoning']}

Please review this determination and confirm or refine it based on your understanding of traffic laws, autonomous vehicle regulations, and the specific details of this case. Provide specific evidence from the accident description that supports your determination.

"""

llm\_review = await agent.llm.generate(prompt)

# Combine rule-based and LLM determinations

final\_determination = {

"rule\_based": rule\_determination,

"llm\_review": llm\_review,

"final\_determination": extract\_final\_determination(llm\_review, rule\_determination)

}

return final\_determination

flowchart TD

Start([Accident Data Input]) --> A{Impact Type?}

%% Rear-End Collision Branch

A -->|Rear-End Collision| B{Was AV Stationary?}

B -->|Yes| C{Chain Reaction?}

C -->|Yes| D[Determine Initial Vehicle]

D --> E[Assign Fault to Initial Vehicle]

C -->|No| F{Sudden Stop by AV?}

F -->|Yes| G[Shared Fault: 30% AV, 70% Striking Vehicle]

F -->|No| H[100% Fault to Striking Vehicle]

B -->|No| I{AV Moving in Traffic?}

I -->|Yes| J{Sudden Braking by Front Vehicle?}

J -->|Yes| K[Shared Fault: 70% Front Vehicle, 30% AV]

J -->|No| L[100% Fault to AV]

%% Lane Change Branch

A -->|Lane Change| M{Which Vehicle Changed Lanes?}

M -->|AV| N{Other Vehicle Speeding?}

N -->|Yes| O[Shared Fault: 80% AV, 20% Other Vehicle]

N -->|No| P[100% Fault to AV]

M -->|Other Vehicle| Q{AV Speeding?}

Q -->|Yes| R[Shared Fault: 20% AV, 80% Other Vehicle]

Q -->|No| S[100% Fault to Other Vehicle]

%% Intersection Branch

A -->|Intersection| T{Traffic Signal Present?}

T -->|Yes| U{Signal Violation?}

U -->|Yes| V[100% Fault to Signal Violator]

U -->|No| W{Right of Way Violation?}

W -->|Yes| X[100% Fault to ROW Violator]

W -->|No| Y[Analyze Vehicle Movements]

T -->|No| Z{Stop Sign Present?}

Z -->|Yes| AA{Stop Sign Violation?}

AA -->|Yes| AB[100% Fault to Stop Sign Violator]

AA -->|No| AC[Apply Right-of-Way Rules]

Z -->|No| AD[Uncontrolled Intersection]

AD --> AE[Apply Yield-to-Right Rule]

%% Autonomous System Branch

A -->|Any Type| AF{AV in Autonomous Mode?}

AF -->|Yes| AG{Disengagement Before Collision?}

AG -->|Yes| AH{How Soon Before?}

AH -->|Immediately| AI[100% Fault to AV Operator]

AH -->|During Response| AJ[Shared Fault: 60% AV, 40% Other]

AH -->|Well Before| AK[Standard Rule Application]

AG -->|No| AL{AV Failed to Detect Obstacle?}

AL -->|Yes| AM[100% Fault to AV]

AL -->|No| AN[Standard Rule Application]

%% Pedestrian/Cyclist Branch

A -->|Ped/Cyclist Involved| AO{In Crosswalk?}

AO -->|Yes| AP{Had Right of Way?}

AP -->|Yes| AQ[100% Fault to Vehicle]

AP -->|No| AR{Sudden Crossing?}

AR -->|Yes| AS[Shared Fault: 30% Vehicle, 70% Ped/Cyclist]

AR -->|No| AT[70% Fault to Vehicle, 30% to Ped/Cyclist]

AO -->|No| AU{Pedestrian/Cyclist Visibility?}

AU -->|Visible| AV[60% Fault to Vehicle, 40% to Ped/Cyclist]

AU -->|Not Visible| AW[20% Fault to Vehicle, 80% to Ped/Cyclist]

%% Environmental Modifiers

A -->|Any Type| AX{Adverse Weather?}

AX -->|Yes| AY[Reduce Primary Fault by 10%]

AX -->|No| AZ[No Adjustment]

%% Decision Combination

AY --> BA[Final Fault Determination]

AZ --> BA

E --> BA

G --> BA

H --> BA

K --> BA

L --> BA

O --> BA

P --> BA

R --> BA

S --> BA

V --> BA

X --> BA

Y --> BA

AB --> BA

AC --> BA

AE --> BA

AI --> BA

AJ --> BA

AK --> BA

AM --> BA

AN --> BA

AQ --> BA

AS --> BA

AT --> BA

AV --> BA

AW --> BA

BA --> End([Generate Fault Report])