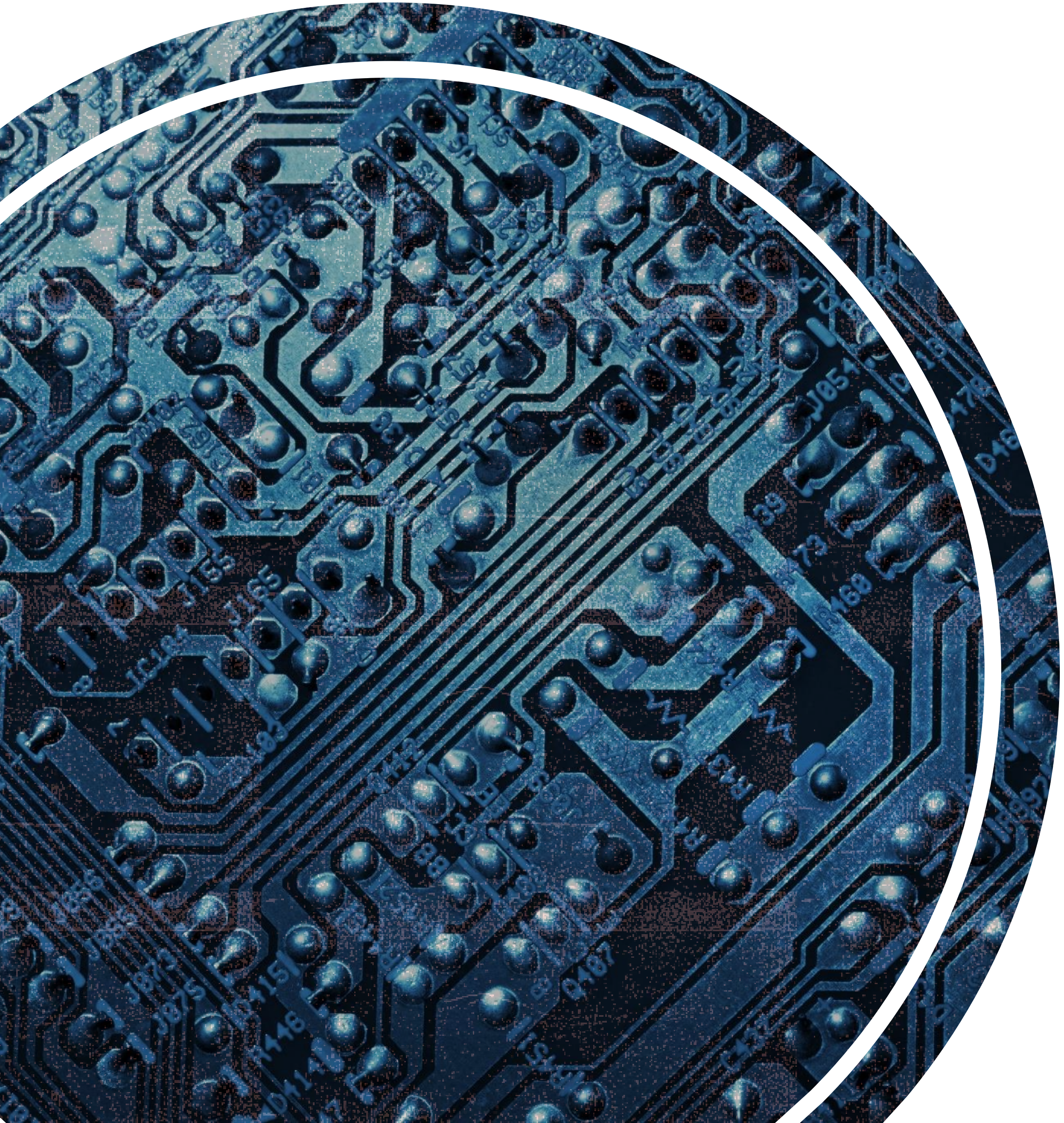
LOGICAL INFERENCE IN ARTIFICIAL INTELLIGENCE



INFERENCE:

- In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.

INFERENCE RULE:

- In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:
- **Implication:** It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.
- **Inverse:** The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.

EXAMPLE INFERENCE RULE:

- From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:
-

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

- Hence from the above truth table, we can prove that P → Q is equivalent to ¬ Q → ¬ P, and Q→ P is equivalent to ¬ P → ¬ Q.

TYPES OF INFERENCE RULE:

1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\dfrac{P \rightarrow Q, \quad P}{\therefore Q}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I am sleepy" ==> P

Conclusion: "I go to bed." ==> Q.

Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

P	Q	P→Q
0	0	1
0	1	1
1	0	0
1	1	1

EXAMPLE CODE:

```
[51] class MotionSensor:
         def __init__(self):
             self.motion_detected = False

         def detect_motion(self):
             # Simulate motion detection
             self.motion_detected = True

     class LightController:
         def __init__(self):
             self.lights_on = False

         def turn_on_lights(self):
             # Simulate turning on the lights
             self.lights_on = True

     # Define modus ponens inference function
     def modus_ponens_inference(motion_detected, light_controller):
         # If motion is detected, then turn on the lights
         if motion_detected:
             light_controller.turn_on_lights()

     # Instantiate objects
     motion_sensor = MotionSensor()
     light_controller = LightController()

     # Simulate motion detection
     motion_sensor.detect_motion()

     # Apply modus ponens inference
     modus_ponens_inference(motion_sensor.motion_detected, light_controller)

     # Check if lights are turned on
     if light_controller.lights_on:
         print("Lights are turned on.")
     else:
         print("No motion detected or lights are not turned on.")
```

```
Lights are turned on.
```

```
[52] class MotionSensor:
         def __init__(self):
             self.motion_detected = False

         def detect_motion(self):
             # Simulate motion detection
             self.motion_detected = False

     class LightController:
         def __init__(self):
             self.lights_on = False

         def turn_on_lights(self):
             # Simulate turning on the lights
             self.lights_on = True

     # Define modus ponens inference function
     def modus_ponens_inference(motion_detected, light_controller):
         # If motion is detected, then turn on the lights
         if motion_detected:
             light_controller.turn_on_lights()

     # Instantiate objects
     motion_sensor = MotionSensor()
     light_controller = LightController()

     # Simulate motion detection
     motion_sensor.detect_motion()

     # Apply modus ponens inference
     modus_ponens_inference(motion_sensor.motion_detected, light_controller)

     # Check if lights are turned on
     if light_controller.lights_on:
         print("Lights are turned on.")
     else:
         print("No motion detected or lights are not turned on.")
```

```
No motion detected or lights are not turned on.
```

2. Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ **Q is true, then** ¬ **P** will also true. It can be represented as:

Notation for Modus Tollens: $\dfrac{P \rightarrow Q, \quad \sim Q}{\sim P}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I do not go to the bed."==> ~Q

Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:

P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

TYPES OF INFERENCE RULE:

EXAMPLE CODE:

[59]

```python
class UserLogin:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def authenticate(self, entered_username, entered_password):
        # Simulate authentication process
        if entered_username == self.username and entered_password == self.password:
            return True  # Authentication successful
        else:
            return False  # Authentication failed

# Function to apply modus tollens inference
def modus_tollens_inference(authentication_result):
    # If authentication fails, the entered credentials must be incorrect
    if not authentication_result:
        print("Incorrect username or password.")
    else:
        print("Authentication successful.")


# Assuming correct username and password
correct_username = "user123"
correct_password = "pass456"

# Simulating user input (incorrect username or password)
entered_username = "abc"
entered_password = "bcd"

# Create UserLogin object
user_login = UserLogin(correct_username, correct_password)

# Authenticate user
authentication_result = user_login.authenticate(entered_username, entered_password)

# Apply modus tollens inference
modus_tollens_inference(authentication_result)
```

```
Incorrect username or password.
```

[60]

```python
class UserLogin:
    def __init__(self, username, password):
        self.username = username
        self.password = password

    def authenticate(self, entered_username, entered_password):
        # Simulate authentication process
        if entered_username == self.username and entered_password == self.password:
            return True  # Authentication successful
        else:
            return False  # Authentication failed

# Function to apply modus tollens inference
def modus_tollens_inference(authentication_result):
    # If authentication fails, the entered credentials must be incorrect
    if not authentication_result:
        print("Incorrect username or password.")
    else:
        print("Authentication successful.")


# Assuming correct username and password
correct_username = "user123"
correct_password = "pass456"

# Simulating user input (incorrect username or password)
entered_username = "user123"
entered_password = "pass456"

# Create UserLogin object
user_login = UserLogin(correct_username, correct_password)

# Authenticate user
authentication_result = user_login.authenticate(entered_username, entered_password)

# Apply modus tollens inference
modus_tollens_inference(authentication_result)
```

```
Authentication successful.
```

3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

Statement-1: If you have my home key then you can unlock my home. **P→Q**
Statement-2: If you can unlock my home then you can take my money. **Q→R**
Conclusion: If you have my home key then you can take my money. **P→R**

Proof by truth table:

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

TYPES OF INFERENCE RULE:

EXAMPLE CODE:

```python
class AutonomousVehicle:
    def __init__(self):
        self.obstacle_detected = False
        self.brakes_applied = False
        self.vehicle_stopped = False

    def detect_obstacle(self):
        # Simulate obstacle detection
        self.obstacle_detected = True

    def apply_brakes(self):
        # Simulate applying brakes
        self.brakes_applied = True

    def stop_vehicle(self):
        # Simulate stopping the vehicle
        self.vehicle_stopped = True

# Function to apply hypothetical syllogism inference
def hypothetical_syllogism_inference(vehicle):
    # If an obstacle is detected, then apply brakes
    if vehicle.obstacle_detected:
        vehicle.apply_brakes()

    # If brakes are applied, then stop the vehicle
    if vehicle.brakes_applied:
        vehicle.stop_vehicle()

# Create AutonomousVehicle object
autonomous_vehicle = AutonomousVehicle()

# Simulate obstacle detection
autonomous_vehicle.detect_obstacle()

# Apply hypothetical syllogism inference
hypothetical_syllogism_inference(autonomous_vehicle)

# Check if the vehicle has stopped
if autonomous_vehicle.vehicle_stopped:
    print("Vehicle stopped successfully.")
else:
    print("Vehicle did not stop.")
```

```
Vehicle stopped successfully.
```

```python
[62] class AutonomousVehicle:
    def __init__(self):
        self.obstacle_detected = False
        self.brakes_applied = False
        self.vehicle_stopped = False

    def detect_obstacle(self):
        # Simulate obstacle detection
        self.obstacle_detected = False

    def apply_brakes(self):
        # Simulate applying brakes
        self.brakes_applied = True

    def stop_vehicle(self):
        # Simulate stopping the vehicle
        self.vehicle_stopped = True

# Function to apply hypothetical syllogism inference
def hypothetical_syllogism_inference(vehicle):
    # If an obstacle is detected, then apply brakes
    if vehicle.obstacle_detected:
        vehicle.apply_brakes()

    # If brakes are applied, then stop the vehicle
    if vehicle.brakes_applied:
        vehicle.stop_vehicle()

# Create AutonomousVehicle object
autonomous_vehicle = AutonomousVehicle()

# Simulate obstacle detection
autonomous_vehicle.detect_obstacle()

# Apply hypothetical syllogism inference
hypothetical_syllogism_inference(autonomous_vehicle)

# Check if the vehicle has stopped
if autonomous_vehicle.vehicle_stopped:
    print("Vehicle stopped successfully.")
else:
    print("Vehicle did not stop.")
```

```
Vehicle did not stop.
```

TYPES OF INFERENCE RULE:

4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>P∨Q

Statement-2: Today is not Sunday. ==> ¬P

Conclusion: Today is Monday. ==> Q

Proof by truth-table:

P	Q	$\neg P$	$P \vee Q$
0	0	1	0
0	1	1	1 ←
1	0	0	1
1	1	0	1

EXAMPLE CODE:

```
[73] def plan_activity(weather_forecast):
         if weather_forecast == "sunny":
             print("Let's go for a picnic!")
         elif weather_forecast == "rainy":
             print("It's raining, let's stay indoors.")
         else:
             print("Weather forecast is uncertain. Use your discretion.")

     # Simulated weather forecast
     weather_forecast = input()

     # Apply disjunctive syllogism
     if weather_forecast != "rainy":
         plan_activity("sunny")
     else:
         plan_activity("rainy")

     sunny
     Let's go for a picnic!

     Start coding or generate with AI.
```

```
[74] def plan_activity(weather_forecast):
         if weather_forecast == "sunny":
             print("Let's go for a picnic!")
         elif weather_forecast == "rainy":
             print("It's raining, let's stay indoors.")
         else:
             print("Weather forecast is uncertain. Use your discretion.")

     # Simulated weather forecast
     weather_forecast = input()

     # Apply disjunctive syllogism
     if weather_forecast != "rainy":
         plan_activity("sunny")
     else:
         plan_activity("rainy")

     rainy
     It's raining, let's stay indoors.
```

TYPES OF INFERENCE RULE:

5. Addition:

The Addition rule is one the common inference rule, and it states that If P is true, then PVQ will be true.

Notation of Addition: $\frac{P}{P \vee Q}$

Statement: I have a vanilla ice-cream. ==> P

Statement-2: I have Chocolate ice-cream.

Conclusion: I have vanilla or chocolate ice-cream. ==> (PVQ)

Proof by Truth-Table:

P	Q	$P \vee Q$
0	0	0
1	0	1 ←
0	1	1
1	1	1 ←

EXAMPLE CODE:

```python
def check_transaction(transaction, suspicious_transactions):
    if transaction in suspicious_transactions:
        print("Transaction flagged as suspicious. Further investigation warranted.")
    else:
        print("Transaction is not flagged as suspicious.")

# List of suspicious transactions (initial rule)
suspicious_transactions = ["transaction1", "transaction2", "transaction3"]

# Additional information indicating a flagged transaction
transaction_to_check = "transaction6"

# Applying addition inference logic
check_transaction(transaction_to_check, suspicious_transactions)
```

```
Transaction is not flagged as suspicious.
```

```python
[82] def check_transaction(transaction, suspicious_transactions):
    if transaction in suspicious_transactions:
        print("Transaction flagged as suspicious. Further investigation warranted.")
    else:
        print("Transaction is not flagged as suspicious.")

# List of suspicious transactions (initial rule)
suspicious_transactions = ["transaction1", "transaction2", "transaction3"]

# Additional information indicating a flagged transaction
transaction_to_check = "transaction2"

# Applying addition inference logic
check_transaction(transaction_to_check, suspicious_transactions)
```

```
Transaction flagged as suspicious. Further investigation warranted.
```

TYPES OF INFERENCE RULE:

6. Simplification:

The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

P	Q	$P \wedge Q$
0	**0**	**0**
1	**0**	**0**
0	**1**	**0**
1	**1**	**1**

EXAMPLE CODE:

[83]

```python
def process_package(heavy, destination):
    if heavy and destination:
        print("Package requires special handling.")
    else:
        print("Package does not require special handling.")

# Simulated package attributes
is_heavy = True
is_destination_specific = True

# Applying the simplification rule
process_package(is_heavy, is_destination_specific)
```

```
Package requires special handling.
```

[84]

```python
def process_package(heavy, destination):
    if heavy and destination:
        print("Package requires special handling.")
    else:
        print("Package does not require special handling.")

# Simulated package attributes
is_heavy = True
is_destination_specific = False

# Applying the simplification rule
process_package(is_heavy, is_destination_specific)
```

```
Package does not require special handling.
```

```python
def check_course_pass(midterm_grade, final_grade, passing_threshold):
    if midterm_grade >= passing_threshold and final_grade >= passing_threshold:
        print("Congratulations! You passed the course.")
    else:
        print("Sorry, you did not pass the course.")

# Simulated student grades and passing threshold
midterm_grade = 75
final_grade = 80
passing_threshold = 60

# Applying the simplification rule
check_course_pass(midterm_grade, final_grade, passing_threshold)
```

```
Congratulations! You passed the course.
```

[86]

```python
def check_course_pass(midterm_grade, final_grade, passing_threshold):
    if midterm_grade >= passing_threshold and final_grade >= passing_threshold:
        print("Congratulations! You passed the course.")
    else:
        print("Sorry, you did not pass the course.")

# Simulated student grades and passing threshold
midterm_grade = 55
final_grade = 80
passing_threshold = 60

# Applying the simplification rule
check_course_pass(midterm_grade, final_grade, passing_threshold)
```

```
Sorry, you did not pass the course.
```

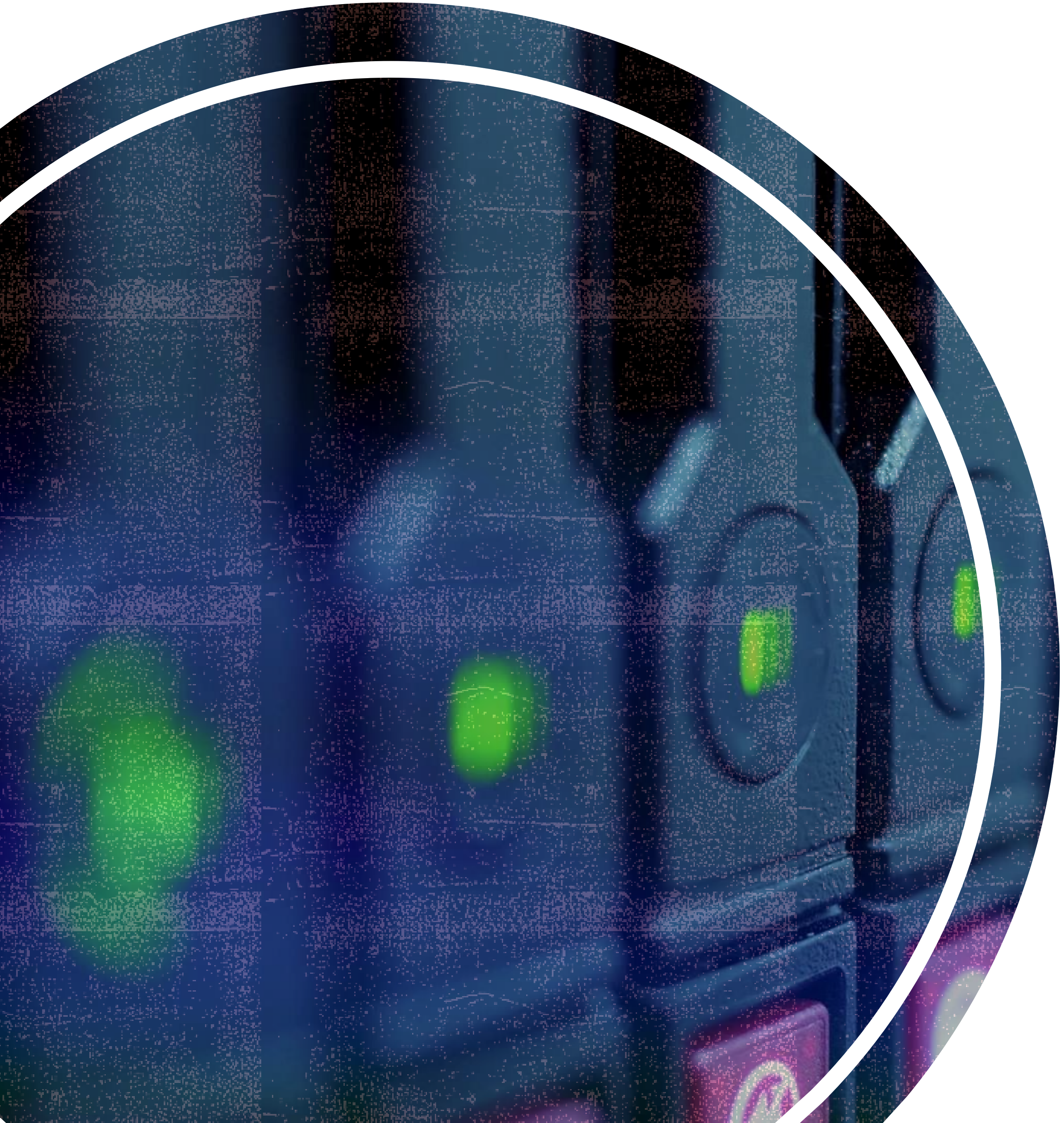
TASKS:

TASKS:

1. **Modus Ponens Task:**

Scenario: You're developing a security system for a bank vault. Implement a function that checks if the security camera detects unauthorized access. If the camera detects unauthorized access, trigger the alarm system.

Objective: Use modus ponens to activate the alarm system when unauthorized access is detected, demonstrating how logical inference triggers actions in a security system.



TASKS:

1. **Modus Tollens Task:**

Scenario: You're creating a temperature monitoring system for a server room. Develop a function that checks if the temperature sensor indicates a temperature above the threshold. If the temperature is not above the threshold, ensure the cooling system remains off.

Objective: Apply modus tollens to keep the cooling system off when the temperature is not above the threshold, showcasing how logical inference prevents unnecessary actions based on negated conditions.

TASKS:

Hypothetical Syllogism Task:

Scenario: You're building a navigation app for drivers. Write a function that determines if the GPS signal is available. If the GPS signal is available, calculate the route to the destination.

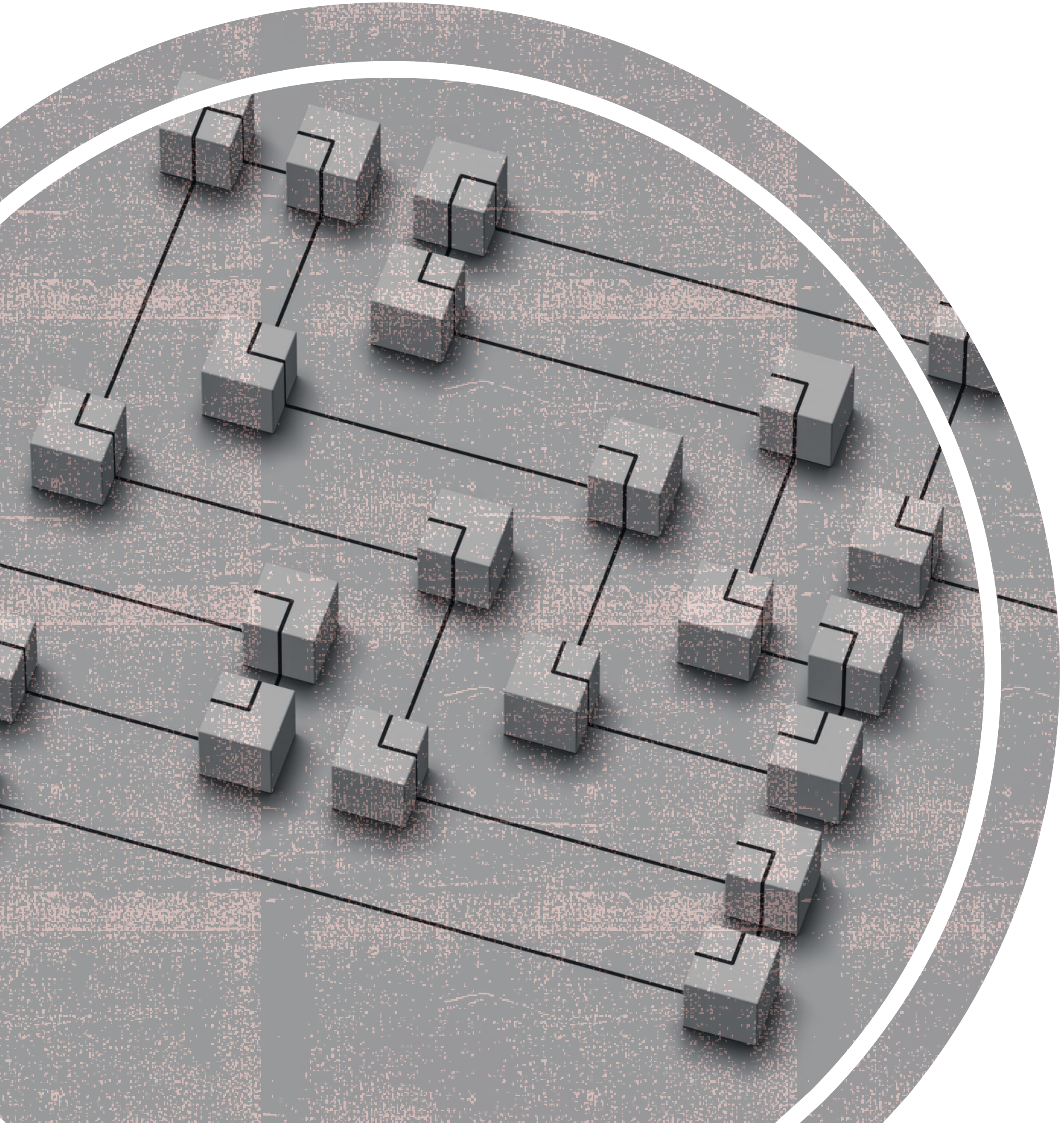
Objective: Utilize hypothetical syllogism to calculate the route to the destination when the GPS signal is available, illustrating how logical inference guides actions in a navigation system.

TASKS:

Disjunctive Syllogism Task:

Scenario: You're developing a scheduling app for students. Implement a function that checks if the user has selected either a morning or evening class. If the user hasn't selected a morning class, assume they've chosen an evening class.

Objective: Apply disjunctive syllogism to infer the user's class preference when a choice is not explicitly provided, demonstrating how logical inference handles alternative options.



TASKS:

Simplification Task:

Scenario: You're developing a game with power-up mechanics. Write a function that simplifies the logic for activating a power-up, considering factors such as player level and available resources.

Objective: Use simplification to streamline the activation logic for power-ups, demonstrating how logical inference simplifies complex decision-making in game development.

TASKS:

Addition Task:

Scenario: You're creating a reservation system for a restaurant. Develop a function that adds a new reservation to the system based on the available time slots and seating capacity.

Objective: Apply the addition rule to incorporate new reservations into the system, showcasing how logical inference updates data based on incoming information in a reservation system.