Programming Assignment 2 & 3 Report

Dom Bulone (dvbulone)

Department of Computer Science, North Carolina State University

CSC 555: Social Computing and Decentralized Artificial Intelligence

Dr. Munindar Singh

November 8, 2025

1 Introduction

This assignment was divided into two phases, each focusing on the Blindingly Simple Protocol Language (BSPL) and its functionality. These two parts serve as a follow-up to the multi-agent system constructed in phase one of this trilogy of assignments. Section 2 of this report provides a general overview of the construction and verification of a safe and functional BSPL protocol between two agents, carefully considering specific constraints. Section 3 of this report provides a breakdown of how large language models (LLMs) were integrated into the system created in phase one of this assignment to transform the agents into fully self-operating pieces of a three-way transactional relationship.

2 Phase Two

In this section, the construction of a BSPL protocol based on several business constraints is described, followed by verification of the protocol's safe functionality. This protocol was designed to be a simple, flexible transactional relationship between a customer and a vendor.

2.1 Requirements

In developing the protocol, several constraints were taken into account:

- The customer should be able to complete a purchase or cancel a transaction at any point.
- Early cancellation should only be possible prior to a payment being sent.
- A return case should exist for a customer to receive a refund after the vendor processes their return.
- Each transaction should conclude gracefully via vendor acknowledgement.
- Customers should not be able to both pay and cancel a transaction.
- A conclusion for each transaction should be able to be reached via cancellation, successful delivery, or return. This conclusion should be sent from the vendor to the customer.

In the next section, the protocol's design process will be explained with respect to each of these constraints.

2.2 Protocol Design

Logically separating the protocol into three possible scenarios was the simplest way to begin creating it. These three paths, as determined by the requirements, were:

1. Standard purchase/ship transaction.
2. A transaction resulting in a cancellation before payment.
3. A transaction resulting in a return following shipment.

Considering the protocol in this way helps better understand what the protocol design should achieve. From these three possible transactions, the protocol can be further broken down. The first steps are simple: a customer should order an item, then the vendor should confirm the order. Following these initial pieces, the first path can be drawn out by creating a “purchase” option. The second path can then be illustrated via a “cancel” option. To address the third possible path, the customer must first purchase and receive the item via “ship” and “delivery” steps. Once those two additional pieces are added, the third path can be completed via a “return” transaction. Finally, per the requirements, each possible path should result in vendor-to-customer completion messages.

The full BSPL code, which follows this logical interpretation, is shown in *Figure 1*, below.

```
protocol TransactionProtocol {
    roles Customer, Vendor
    parameters out ID key, out item, out price, out confirmation, out decision, out outcome
    private payment, paymentinfo, shipment, address, return_flag

    // Transaction initial comms.
    Customer -> Vendor : order [out ID key, out item, out price]
    Vendor -> Customer : confirm [in ID, in item, in price, out confirmation]

    // Independent comms determined by cancellation or purchase.
    Customer -> Vendor : purchase [in ID, in confirmation, nil decision, out address, out paymentinfo, out payment, out decision]
    Customer -> Vendor : cancel [in ID, in confirmation, nil payment, out decision]

    // Step following a purchase.
    Vendor -> Customer : ship [in ID, in payment, out shipment]

    // End: Delivery, return & return confirmed, and completion of return.
    Vendor -> Customer : complete_delivery [in ID, in shipment, nil return_flag, out outcome]
    Customer -> Vendor : return [in ID, in shipment, nil outcome, out return_flag]
    Vendor -> Customer : complete_cancellation [in ID, in decision, nil shipment, out outcome]
    Vendor -> Customer : complete_return [in ID, in return_flag, out outcome]
}
```

Figure 1: Completed BSPL.

Importantly, following the logical flow described above is only one piece of the creation of this protocol. Starting at the top of the code, the roles are instantiated, followed by the parameters. Starting with the public parameters, “ID” is listed as the key, since it is present in every communication. “Item” and “Price” are determined by the customer’s order; “confirmation” is the output of the vendor’s “confirm” message; “decision” is used as a flag for the cancellation path to prevent the customer from making a purchase after cancelling; and “outcome” is the result from every completion message. Moving to the trickier, private parameters: “payment,” “shipment,” “address,” and “return_flag” are listed as private parameters since they are used to enable the execution of the branches of the protocol to which they belong (e.g., “payment” enables “ship” and on). The following lines incorporate each parameter to correctly transition through each possible branch, following the logical flow.

2.3 Protocol Functionality Verification

Following the careful development of the protocol, verification is necessary to determine the ability of the protocol to run safely. *Figure 2* shows the results of the verification tests.

```
(venv) PS C:\Users\Dominick2\Desktop\NCState\DAI\PA\bspl\bspl\PA3> bspl verify all dvbulone_protocol.bspl
TransactionProtocol (dvbulone_protocol.bspl):
{'elapsed': 0.010720100020989776, 'paths': 11, 'safe': True}
{'elapsed': 0.010866799857467413, 'paths': 11, 'live': True}
(venv) PS C:\Users\Dominick2\Desktop\NCState\DAI\PA\bspl\bspl\PA3> bspl verify paths dvbulone_protocol.bspl
TransactionProtocol (dvbulone_protocol.bspl):
{'elapsed': 0.013053500093519688, 'paths': 53, 'longest': (Customer!order, Vendor?order, Vendor!confirm, Customer?confirm, Customer!purchase, Vendor?purchase, Vendor!complete_cancellation, Vendor!shi
p, Customer?ship, Customer!return, Vendor?return, Customer?complete_cancellation), 'max-paths': 16}
```

Figure 2: Verification tests for dvbulone_protocol.bspl

As is seen in the figure above, the tests yielded positive results for the safety and liveness of the protocol!

3 Phase Three

Phase three further developed the system created in the first part of the assignment. As a reminder, the system developed in phase one was a three-agent transaction protocol using BSPL. These three agents (buyer, seller, and shipper) communicated to complete transactions and deliveries for items purchased by the buyer. In phase three, the context was changed, and LLMs were integrated to enable the agents to make smart decisions based on the new context and various constraints.

In this phase, the buyer is purchasing ski equipment for a family of four who are leaving for a ski trip in three weeks. The buyer carefully makes purchasing decisions based on item priorities, weather forecasts, and budget constraints. The seller prices the items based on supply, demand, and competition. The shipper makes operational considerations based on delivery locations, weather, and fleet capacity. More will be discussed about each agent in Section 3.1.

3.1 LLM Implementation

As a general overview, an independent LLM was integrated into each agent to avoid request constraints and overlaps in decision-making. These LLMs were integrated via OpenRouter, and the specific model (same model for each agent) was chosen based on which provided the best operability for the system. Each agent had its own API key read from a .env file. Completing this process was integral to the LLM integration within each agent.

To ensure careful consideration was given to the context and constraints by each agent, a holistic prompt was written as an input for each LLM, enabling the critical “thinking” necessary to meet the project goals. More fine-grained details on the LLM implementations for each agent are described in the sections below.

3.1.1 Buyer LLM Integration

In this model, the buyer is an intelligent trip planner who makes determinations based on several constraints determined from important context. There are three categories of items: essential, important, and optional. The buyer's goal is to purchase one of each essential item for each member of the family to ensure their safety. Important and optional items are not required, but may be nice to have on a ski trip. Similar to phase one, the buyer creates requests for quotes (RFQs) that are sent to the seller. The seller then responds with quotes that the buyer must evaluate and either reject or accept. The buyer has a total budget of $6000 that must be carefully rationed to ensure the family's needs are met; the leftover money can be spent on lower-priority items. Since it is important for the family to purchase essential items, the buyer shows a willingness to pay a premium for those items. Additionally, there are three delivery options for the buyer to choose from based on delivery urgency.

The prompt created for the buyer diligently explains these constraints.

```
# Decision prompt to the LLM
prompt = f"""
You are a buyer planning to purchase ski gear for 4 people attending a 5-day ski trip in 3 weeks. Heavy snow is expected, so essential
items must be delivered before the trip. Keep in mind, the whole family of four needs one of each essential item.
Non-essential items are optional. Here is the context:

- Item: {item}
- Offered price: ${price:.2f}
- Item budget: ${Bitem[item]:.2f}
- Remaining total budget: ${Bremaining:.2f}
- Essential items (NEED one of each for all four people): ski_boots ($400), skis ($600), ski_poles ($150)
- Important (NOT ESSENTIAL): winter_jacket, ski_goggles, gloves
- Optional: thermal_underwear, hydration_pack, ski_socks
- Addresses (priority order): 789 Pine Rd (long delivery time/blizzard), 456 Oak Ave (moderate delivery time/snowing), 123 Main St (low delivery time/clear)
- Decide on an address based on weather conditions, length of simulation run time, and urgency of receiving item.
- Weather conditions may lead to delivery delays. Be smart about choosing a delivery address.

Rules:
1. Always accept essential items if price ≤ 120% of their budget.
2. Reject non-essential items (important, optional) unless well under budget or remaining funds are high.
3. Stop buying if all essential items are acquired or remaining funds < $1000.
4. Do not stop shopping unless all essential items are secured for all 4 people, or if the budget is critically low (< lowest item cost).

Return ONLY JSON:
{{
  "decision": "ACCEPT" | "REJECT" | "STOP",
  "reason": "Brief reasoning",
  "address": "789 Pine Rd" | "456 Oak Ave" | "123 Main St"
}}
"""
```

Figure 3: Buyer LLM prompt.

3.1.2 Buyer Phase 1 vs. Buyer Phase 3

Much of the internal logic of the buyer remained the same in phase three as it was in phase two: the buyer sends RFQs, receives quotes, and makes an acceptance or rejection determination based on its constraints. The messaging between the buyer and its counterparts remained the same in this manner. The significant change between the two phases obviously comes from the integration of the LLM.

In phase one, the buyer's constraints were hardcoded. Every case that the buyer could possibly encounter had to be written in the code itself. In phase three, this is handled by the prompt seen in *Figure 3*. Every constraint is described, and the context is explained in detail. At a glance, this would make it seem like the integration of the LLM makes the code simpler. However, the LLM needs as much detail as possible to make good decisions. Several iterations of the prompt were created until the LLM's logic was sound. Additionally, careful parsing of the output was necessary to craft the messages sent to the seller.

```
cleaned = (
    response.strip()
    .removeprefix("```json")
    .removeprefix("```")
    .removesuffix("```")
    .strip()
    .replace(",}", "}")
    .replace("“", "\"")
    .replace("”", "\"")
)
result = json.loads(cleaned)
```

Figure 4: Buyer JSON parsing.

The figure above shows a new, but incredibly necessary part of the buyer agent's code. The LLM's output is strictly written in JSON, but these lines were necessary in order to grab and store important metrics that were included in the messaging.

Furthermore, in phase one, the final statistics of each agent were printed after a hardcoded delay. In phase three, the buyer is required to send an explicit "stop" message once it has either purchased all essential items or reached a minimum budget threshold. Because of this requirement, a new reaction had to be created in order to halt all agents following the buyer's completion. Only after receiving this message were agents permitted to stop and print their final statistics. The implementation of this message is shown below.

```python
async def stop_shopping(reason: str):
    global shopping_done
    shopping_done = True
    logger.info(f"LLM stopping transactions. Reason: {reason}")

    # Wait to allow deliveries to complete
    if delivered_count >= accepted:
        logger.info("All deliveries completed, sending DONE.")
        await adapter.emit_local({"type": "done", "role": "Buyer", "reason": "All deliveries completed"})

    buyer_stats()

    done_msg = {"type": "done", "role": "Buyer", "reason": reason}

    await adapter.emit_local(done_msg)

    logger.info("Emitted DONE message locally (broadcast to all roles).")
    adapter.stop()

@adapter.reaction("done")
async def on_done(_msg):
    logger.info("Received DONE signal. Printing final stats and stopping...")
    buyer_stats()
    adapter.stop()
```

Figure 5: Buyer "done" message implementation.

This message is sent by the buyer to the seller and shipper, signaling the completion of the transaction. There is a check here to ensure that all delivery notifications are made to the buyer by the shipper prior to sending the message, preventing completion of the transactions before all messages are received.

As a final note, the requirements of the assignment stated that the buyer's total budget should be $1800. However, this would prevent the buyer from meeting the requirement to purchase one of each essential item for every member of the family since the combination of those items would be well over budget. $6000 was chosen as a safe budget based on the total sum of the buyer's item-specific budgets.

3.1.3 Seller LLM Integration

Much of the integration of the LLM into the seller agent matches what was seen in the buyer. As a brief overview, the seller was to determine appropriate pricing based on seasonal demand, inventory, competition, and profit margin. The LLM was left with the freedom to choose what it should price each item at, so long as it considered these constraints. The prompt for the seller's LLM is shown below.

```
prompt = f"""
    You are a seller of ski equipment currently operating at the season's peak demand. You must determine the optimal
    quote price for {item} based on the following context:

    - Seasonal demand is high due to this being the peak part of the ski season.
    - Inventory status: {stock_level} units of {item} remaining.
    - Competition: This is a competitive market, so pricing must be attractive yet profitable.
    - Cost structure: you acquire the goods at 60% of the retail price and aim for a 40% profit margin.
    - Current retail baseline price for {item} is {base_prices[item]:.2f}.
    - Cost basis: ${base_prices[item] * cost_factor:.2f}.
    - Baseline profit price: ${base:.2f}.

    - Price items reasonably based on stock and demand:
        - Essential items: ski_boots, skis, ski_poles -> higher demand, lower stock, can command premium (higher) pricing.
            - Never price essential items below the baseline unless the stock for those items is very high.
        - Non-essential items: winter_jacket, ski_goggles, gloves -> moderate demand, higher stock, competitive pricing.
        - Optional items: thermal_underwear, hydration_pack, ski_socks -> lower demand, ample stock, budget-friendly pricing.

    Remember to:
    1. Offer competitive yet profitable prices.
    2. Price aggressively vs competitors while maintaining profitability.
    3. Consider stock levels in pricing strategy.
    4. Avoid pricing below cost basis (60% of retail).
    5. Return reasoning that shows understanding of market dynamics.

    Return your decision in valid JSON format:
    {{
        "decision": "QUOTE" | "DECLINE",
        "price": <float>,
        "reason": "Short explanation of pricing decision considering stock, demand, competition, and profitability."
    }}
"""
```

Figure 6: Seller LLM prompt.

As is seen in the figure above, all the seller had to do was respond to the buyer's RFQs with a price that meets supply, demand, and competition constraints. The agent's primary goal was to ensure a profit of 40% was met in all sales.

3.1.4 Seller Phase 1 vs. Seller Phase 3

Like the buyer, the seller agent in phase three has several differences from its phase one counterpart. In phase three, all hardcoded constraints were removed, and the determination of prices was left to the LLM. The seller still receives RFQs from the buyer and responds with quotes. Once a quote is accepted, the seller sends a ship message to the shipper. One major difference is the signaling of the "done" message sent by the buyer. This is simply solved by creating another adapter reaction for the seller to print its final statistics and halt once a "stop" message is received.

```
@adapter.reaction("done")
async def on_done(_msg):
    logger.info("Received DONE signal. Printing final stats and stopping...")
    await seller_stats()
    adapter.stop()
```

Figure 7: Seller reaction to buyer "done."

3.1.5 Shipper LLM Integration

The shipper is the simplest of the three agents: it must patiently wait for a “ship” message from the seller, then develop a delivery process and message to be sent to the buyer. The shipper had several constraints to enable its shipping process: weather conditions, delivery options (standard, express, overnight), fleet capacity, and seasonal demand. Similar to the buyer and seller, these constraints were described within a prompt that was used as input for the LLM.

```
prompt = f"""
    You are a winter logistics coordinator. Decide how to handle the shipment below given weather conditions, capacity, and service options:
        Shipment details:
            - ID: {shortID}
            - Item: {item}
            - Address: {addr} ({zone_name} zone)
            - Base ETA: {z['min']}-{z['max']} days
        Operational context:
            - Weather at destination: {z['weather']}
            - Route risk: {z['risk']}
            - Peak Season: {"Yes" if capacity["peak_season"] else "No"}
            - Fleet Capacity: {capacity['active_trucks']} of {capacity['max_trucks']} trucks currently in use
            - Available Trucks: {capacity['max_trucks'] - capacity['active_trucks']}
            - If there are 0 active trucks, that means the full capacity is available for delivery.
                -Likewise, if there are 6 active trucks, that means no trucks are available for delivery.
            - Service types:
                - standard: ~{max(z['min'],1)}-{z['max']} days
                - express: ~max(3, {z['min']}) days
                - overnight: 1 day (low/medium risk only and capacity permitting)
        Determination:
            - Choose to SHIP, DELAY (weather/capacity), or REJECT (impossible, bad address) the shipment.
            - Pick the best service type available (standard, express, overnight) based on conditions.
            - Provide estimated delivery days (eta_days) within the chosen service and base window.
            - If you cannot safely ship now, set an appropriate outcome like "weather_delay" or "capacity_exceeded".
            - Never set a price. Only worry about making logistical decisions.

    Return your decision in valid JSON format:
    {{
        "decision": "SHIP" | "DELAY" | "REJECT",
        "service": "standard" | "express" | "overnight",
        "eta_days": <int>,
        "outcome": "delivered" | "address_not_found" | "recipient_unavailable" |
                   "damaged_in_transit" | "weather_delay" | "capacity_exceeded",
        "reason": "<brief reasoning>"
    }}
"""
```

Figure 8: Shipper LLM prompt.

As can be seen in the figure above, the prompt instructs the LLM to carefully consider all of the constraints in making its delivery decision. The shipper had to determine the best course of action given road conditions and fleet capacity. A message is crafted from the shipper’s decision and sent to the buyer.

3.1.6 Shipper Phase 1 vs. Shipper Phase 3

Similar to the buyer and seller, the shipper in phase three had no hard-coded decision-making. Much of the shipper remained the same outside of this key difference, but the inclusion of fleet capacity is a significant addition in this phase. Phase three sought to integrate realism in shipping processes by including a finite number of company vehicles. To implement this requirement, an “active trucks” counter was created to be incremented on every active delivery. Upon receipt of a “ship” message from the seller, a truck is reserved. If the shipper decides to ship an item, “active

trucks" is incremented if the shipment went through (no weather delay or capacity issues). This counter is tracked to ensure deliveries aren't attempted if the fleet capacity is at zero (all fleet vehicles are out for delivery).

4 Execution and Evaluation

Brief snippets of each agent's execution windows are shown below with evaluations. One important overarching issue in the system's execution is the time requirement. The transaction protocol proceeds incredibly slowly due to the safeguards in place to avoid LLM rate-limiting. This is unfortunate, as any attempts to lower the run time result in errors with the LLM rate limits. Additionally, as a final note, the outputs are lengthy, preventing full execution results from being shown in this report.

4.1 Buyer Execution and Evaluation

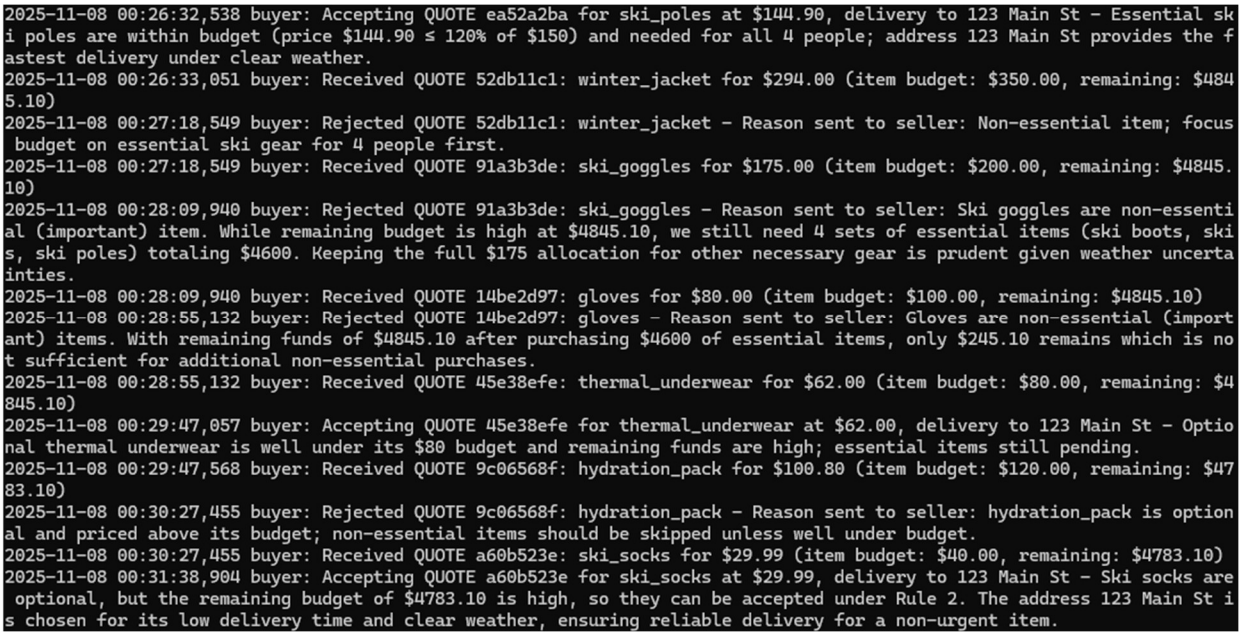

Figure 9: Buyer execution snippet.

The figure above shows a brief snippet of the buyer's execution window. The output in this figure shows the buyer is receiving quotes from the seller and carefully considering its requirements to decide whether or not to purchase an item per a given quote. Several rejections are shown, and the buyer provides reasoning for each that pertains to the family's requirements. Furthermore, several quote acceptances are shown where the buyer rationalizes why it would accept the quote given all external context and constraints. This figure does not display the sending of the RFQs or the receipt of delivery messages from the shipper. However, the full execution window shows both message types received by the buyer, indicating successful integration of the LLM per project requirements.

4.2 Seller Execution and Evaluation

```
2025-11-08 00:11:26,358 seller: Starting Seller...
2025-11-08 00:11:26,359 seller: Awaiting messages...
2025-11-08 00:11:53,458 seller: RFQ 3a1e5840: quoting ski_boots at $405.00 - Peak-season demand and very low stock (5 un
its) justify a premium on essential ski_boots. Pricing at 405 stays attractive against competitors while keeping a healt
hy 41% margin on retail (69% on cost) and well above the 336 baseline. This balances profitability with aggressiveness w
ithout dropping below cost. (Stock: 5)
2025-11-08 00:12:16,271 seller: RFQ 8af6939c: quoting skis at $648.00 - Peak-season demand with only 5 units left; must
remain above cost ($360) and the 40% margin target. Price slightly above retail baseline ($600) to capture scarcity prem
ium while staying competitive and profitable. (Stock: 5)
2025-11-08 00:12:39,618 seller: RFQ 9995c875: quoting ski_poles at $149.00 - High seasonal demand and low remaining stoc
k (8 units) support a premium for this essential item. Quoted at $149.00—slightly below the $150 baseline but well above
 the $126 baseline profit—keeps the offer attractive against competitors while securing a strong margin (39.3%) and avoi
ding pricing below the $90 cost basis. (Stock: 8)
2025-11-08 00:13:01,583 seller: RFQ a3a1b48f: quoting winter_jacket at $320.00 - Pricing winter_jacket at $320.00 balanc
es peak season demand with competitive positioning. At 20 units remaining during high season demand, I can command premi
um pricing while staying attractive vs the $350 baseline. The price maintains 40% profit margin above the $294 baseline
and cost basis protection, while being competitive in the non-essential item category with moderate demand and higher st
ock. (Stock: 20)
2025-11-08 00:13:30,077 seller: RFQ c617b012: quoting ski_goggles at $180.00 - Set price at $180, just under the $200 re
tail baseline but well above the $120 cost basis, yielding a healthy margin while staying competitive. With 10 units lef
t and peak seasonal demand, this aggressive price balances urgency and profitability without resorting to discounts. (St
ock: 10)
2025-11-08 00:13:52,554 seller: RFQ 717c4893: quoting gloves at $84.00 - Pricing at baseline profit margin of $84.00 dur
ing peak season demand. With 12 units remaining (moderate stock) and high seasonal demand, this competitive yet profitab
le price maintains the 40% profit margin while staying attractive in the competitive market. The non-essential nature of
 gloves justifies avoiding premium pricing, but peak season conditions support maintaining standard profitability rather
 than discounting. (Stock: 12)
```

Figure 10: Seller execution snippet.

The figure above is a display of the first few quotes sent from the seller to the buyer. The figure shows the RFQs being received from the buyer, and the seller's succinct explanation of how the sale price was determined. The output shows that the seller is making careful determinations based on stock, demand, and competition. The careful thinking from the LLM results in fair prices that still ensure that its profit margins are being met. Not shown in this image is the seller sending "ship" messages to the shipper or receiving "Reject" messages from the buyer. However, these messages can be seen when running a full test of the system, showing full functionality of the seller per the project requirements.

4.3 Shipper Execution and Evaluation

```
2025-11-08 00:23:21,998 shipper: Starting Shipper...
2025-11-08 00:23:21,999 shipper: Awaiting shipping messages...
2025-11-08 00:27:22,567 shipper: Received SHIP dda0f08c from Seller for ski_boots destined to 123 Main St
2025-11-08 00:28:09,702 shipper: Shipment dda0f08c to 123 Main St [Local] | Decision: SHIP | Service: standard | ETA: 1
days | Outcome: delivered | Reason: Clear weather, low route risk, full fleet capacity available, base ETA within 1-2 da
ys
2025-11-08 00:28:09,704 shipper: Delivery SUCCESS dda0f08c: ski_boots delivered to 123 Main St in 1 days
2025-11-08 00:28:09,707 shipper: Received SHIP 01091a91 from Seller for skis destined to 123 Main St
2025-11-08 00:28:52,878 shipper: Shipment 01091a91 to 123 Main St [Local] | Decision: SHIP | Service: overnight | ETA: 1
 days | Outcome: delivered | Reason: Weather is clear, route risk is low, and all 6 trucks are available, so we can use
overnight service to meet the 1-day delivery window.
2025-11-08 00:28:52,879 shipper: Delivery SUCCESS 01091a91: skis delivered to 123 Main St in 1 days
2025-11-08 00:28:52,880 shipper: Received SHIP ea52a2ba from Seller for ski_poles destined to 123 Main St
2025-11-08 00:29:43,120 shipper: Shipment ea52a2ba to 123 Main St [Local] | Decision: SHIP | Service: overnight | ETA: 1
 days | Outcome: delivered | Reason: Clear weather, low route risk, and full fleet capacity allow expedited overnight de
livery.
2025-11-08 00:29:43,120 shipper: Delivery SUCCESS ea52a2ba: ski_poles delivered to 123 Main St in 1 days
2025-11-08 00:29:47,058 shipper: Received SHIP 45e38efe from Seller for thermal_underwear destined to 123 Main St
2025-11-08 00:30:31,852 shipper: Shipment 45e38efe to 123 Main St [Local] | Decision: SHIP | Service: overnight | ETA: 1
 days | Outcome: delivered | Reason: Clear weather, low route risk, and full capacity available allow for overnight serv
ice to thermal_underwear shipment in local zone
2025-11-08 00:30:31,852 shipper: Delivery SUCCESS 45e38efe: thermal_underwear delivered to 123 Main St in 1 days
```

Figure 11: Shipper execution snippet.

The figure above shows a brief snippet of the shipper's execution window. The figure shows the shipper waiting for messages, then sending shipment and delivery messages as the ship messages arrive from the seller. The output of the shipment messages shows the shipper carefully considering the route risk and fleet capacity, with a delivery decision based on both. The outputs show the LLM is functioning as intended, as its considerations are printed succinctly, supporting its decisions. Similar to the seller, however, the cases in which the shipper rejects a shipment are not shown due to the lengthy output. These message types can be seen in full testing, verifying the functionality of the shipper.

5 Challenges

5.1 Phase 2 Challenge

One significant challenge faced in creating the BSPL for phase two was the determination of private vs. public parameters. My initial understanding of the parameter types was that a private parameter was an output that was never taken as an input for another part of the protocol sequence. While this is partially correct, the error in the logic was quickly shown in the development of the more complicated BSPL required in phase two. Given the requirements, the protocol should be able to conclude at the end of multiple "paths." After completing the initial draft of the protocol, the verification checks showed successful results in safety, but not in liveness. After digging further, I gained a new understanding that private parameters are actually those that are used to enable specific portions of the protocol, rather than being globally impactful.

5.2 Phase 3 Challenges

There were many issues faced in the integration of LLMs into the phase one system; however, only the two most significant issues are addressed below.

5.2.1 LLM Rate Limiting

After implementing all of the requirements and making the necessary changes to each agent's code, testing began. However, a significant challenge was immediately faced due to rate limiting from the integrated LLMs. Understandably, free LLMs have low rate limits to deter organizations from using them in operational instances. Initially, only one LLM was used for all three agents - this quickly led to the rate limit being hit on every test due to the number of requests being made across all three agents. Several iterations of the system were developed from these errors. Notably, several different models were used for testing, and lengthy hard-coded sleeps were added to try to avoid the rate limit. After several attempts to alleviate this issue, it was determined that the three agents should not be using a single model, but each should have their own. After this determination, two new API keys were generated, the .env was changed to include a key per agent, and llm.py was altered to assign one model to one agent per the assigned keys. This immediately alleviated the rate-limiting issues.

5.2.2 LLM Prompt Creation

The second major issue faced in the integration of LLM into the system was with creating a prompt for each agent that allowed them to be independently deterministic while still meeting critical objectives. There are several examples of where the descriptiveness of the prompt led to significant errors, but a general focus on how the LLM interprets the prompts would more succinctly describe the challenge. The initial assumption was that the LLM could make assumptions based on the provided situational context. However, this was not a correct understanding of how an LLM operates. Generally, they are not able to independently make accurate decisions because of their lack of experience. This is why significant time is spent training AI/ML models to achieve a single goal. The prompts had to be granular in their explanations to ensure no gaps were left for the LLM to fill. Once this was realized, each prompt was updated to include as much detail as possible, while still explaining the situation and constraints succinctly to ensure the LLMs were able to efficiently make good decisions.