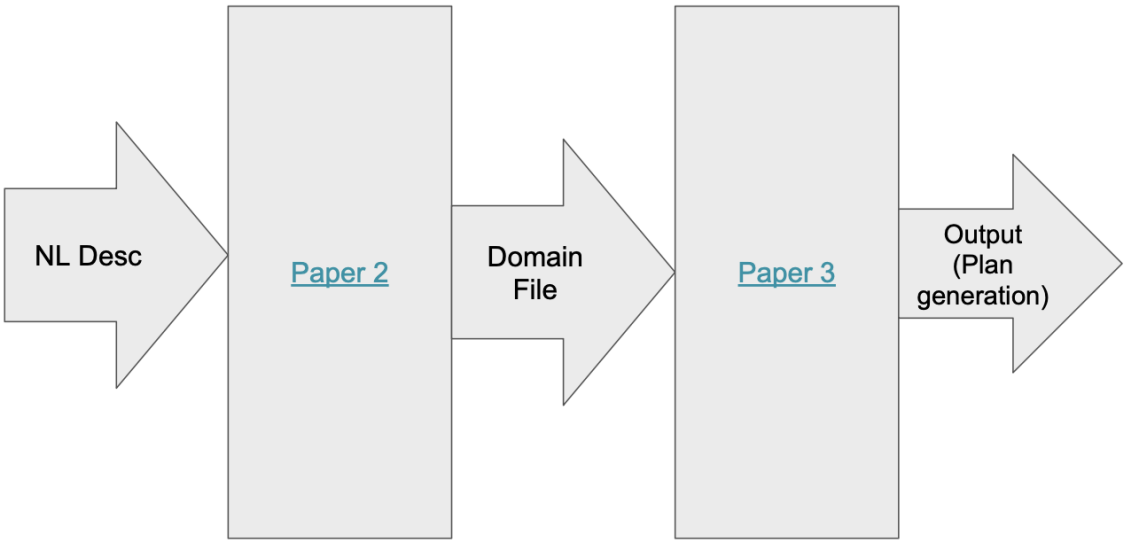
I am exploring the feasibility of predicting both PDDL files from natural language inputs by leveraging insights from the following two papers (named them as Paper 2 and Paper 3) . My approach involves combining their proposed pipelines and further refining the methodology based on the results:

- https://arxiv.org/pdf/2305.14909 [Paper 2]
- https://arxiv.org/pdf/2405.19793 [Paper 3]



- Created a new domain (coin collector) for Paper 2 to test the output (TextExpress game simulated domain of Paper 3)
- The domain is generated using the inverse process by taking the domain file from Paper 3 and with the help of LLM.
- Three files needed for new domains
  - Action model JSON
    - Natural language description of actions in domain
  - Domain description (text)
    - Comprehensive description of domain
  - Hierarchy requirements JSON
    - Requirements for PDDL planner and objects in environment
- combining the pipelines of Paper 2 and 3

- Now, we have a pipeline for plan generation from an input NLP text description

Domain file for ‘coin collector’

```
(define (domain environment)
  (:requirements :strips :typing :negative-preconditions :disjunctive-
        preconditions)
  (:types
    location
    direction
  )
  (:predicates
    (at ?loc - location)
    (visited ?loc - location)
    (connected ?loc1 - location ?loc2 - location ?dir - direction)
    (closed_door ?loc1 - location ?loc2 - location)
  )

  (:action move
    :parameters (?loc1 - location ?loc2 - location ?dir - direction)
    :precondition (and (at ?loc1) (connected ?loc1 ?loc2 ?dir) (not (
          closed_door ?loc1 ?loc2)))
    :effect (and (not (at ?loc1)) (at ?loc2))
  )

  (:action open_door
    :parameters (?loc1 - location ?loc2 - location)
    :precondition (and (at ?loc1) (closed_door ?loc1 ?loc2))
    :effect (not (closed_door ?loc1 ?loc2))
  )
)
```

Figure 5: Annotated domain file for Coin Collector.

Natural language description generated:


```
{
    "Move": {
        "desc": "This action enables the agent to move from one location to another in the specified direction. For example, the agent can move from location A to location B if they are connected in that direction and there is no closed door between them.",
        "extra_info": []
    },

    "Open a Door": {
        "desc": "This action enables the agent to open a closed door between two locations, allowing movement between them. For example, if there is a closed door between location A and location B, the agent at location A can open it, making it possible to move to location B.",
        "extra_info": []
    },

    "Pick": {
        "desc": "This action enables the agent to pick up a coin once it is in the same room where the coin is located. For example, if the agent is in location C, and a coin is found in the same location, the agent can pick up the coin, achieving the goal of collecting it.",
        "extra_info": []
    }
}
```

Domain description:

The AI agent in this domain is tasked with exploring a series of connected rooms, each represented as a unique location. The ultimate goal of the agent is to locate and collect a coin hidden in one of these rooms. The agent does not initially know where the coin is located and must systematically explore the rooms to find it. Rooms may be connected by doors, some of which are locked and need to be opened to allow passage. The agent can only move between rooms if a door is open, and if the rooms are connected in the desired direction. The agent can open a closed door if it is in the current location of that door. The agent is designed to remember which rooms it has already visited to avoid revisiting them unnecessarily. By continuously exploring unvisited rooms, the agent progresses toward its main goal of discovering the room containing the coin.

Initial output generated by Paper 2:

"Move": "llm_output": "1. ?from - location: the current location of the agent\n2. ?to - location: the destination location where the agent wants to move\n3. ?dir - direction: the direction in which the agent wants to move\n\n

Preconditions:\n```\n(and\n (agent-at ?from)\n (connected ?from ?to ?dir)\n (door-open ?from ?to)\n)\n```\n\n

Effects:\n```\n(and\n (not (agent-at ?from))\n (agent-at ?to)\n (visited ?to)\n)\n```\n\n

Formatted Domain file output from Paper 2 (as input to Paper 3)

```
(define (domain environment)
  (:requirements :strips :typing :negative-preconditions :disjunctive-preconditions)
  (:types
    location
    direction
  )
  (:predicates
    (at ?loc - location)
    (visited ?loc - location)
    (connected ?loc1 - location ?loc2 - location ?dir - direction)
    (closed_door ?loc1 - location ?loc2 - location)
  )
  (:action move
    :parameters (?loc1 - location ?loc2 - location ?dir - direction)
    :precondition (and (at ?loc1) (connected ?loc1 ?loc2 ?dir) (not (closed_door ?loc1
?loc2)))
    :effect (and (not (at ?loc1)) (at ?loc2))
  )
  (:action open_door
    :parameters (?loc1 - location ?loc2 - location)
    :precondition (and (at ?loc1) (closed_door ?loc1 ?loc2))
    :effect (not (closed_door ?loc1 ?loc2))
  )
)
```

PDDL problem file generated by Paper 3 at last step (iterative)

```
(define (problem exploration)
  (:domain environment)
  (:objects
    kitchen corridor bedroom living_room backyard street l7 - location
    east west - direction
  )
  (:init
    (at street)
    (visited kitchen)
    (visited corridor)
    (visited bedroom)
    (visited living_room)
    (visited backyard)
    (visited street)
    (connected backyard street east)
    (connected street backyard west)
    (connected street l7 west)
    (closed_door street l7)
  )
  (:goal
    (exists (?x - location)
      (and
        (not (visited ?x))
        (at ?x)
      )
    )
  )
)
```

Plan found at the last step

Found Plan (output)

(open_door street l7)

(move street l7 west)

```
(:action move
  :parameters (street l7 west)
  :precondition
    (and
      (at street)
      (connected street l7 west)
      (not
        (closed_door street l7)
      )
    )
  :effect
    (and
      (not
        (at street)
      )
      (at l7)
    )
)
```

Combined Pipeline Output WorkFlow (iterative)