VI. APPENDIX

This section illustrates the complete execution of the PDDL domain generation process for the Blocks World running example, highlighting the structured data flow and the logic applied in each modular step of the BAML pipeline. Furthermore, this appendix includes the automated semantic evaluation results from the LLM-as-a-Judge approach, presenting the comparison of the scoring prompt and the resulting JSON outputs from the different LLMs tested.

The pipeline receives as the only unstructured input the following description of the domain: "The Blocks World domain involves manipulating a set of blocks of different letters. Blocks can be placed on top of the table or on top of other blocks. Only one block at a time can be on top of another, and a block can only be moved if it is free, meaning it has no other block above it. A robotic arm is the only tool used to perform the movements." Each step generates a typed JSON artifact, which is reused as context for subsequent steps, ensuring logical consistency and the application of Critical Rules.

Pipeline of PDDL generation

Prompt for the Types Extraction

You are an expert in PDDL domain modeling and type extraction.
TASK: Extract ALL object types from the domain description, including their hierarchical relationships.

Critical Rules:

1) Extract EVERY distinct entity type mentioned (nouns representing classes of objects)
2) Identify parent-child relationships (e.g., "truck is a vehicle")
3) If no parent exists, set parent_type to null
4) Use lowercase, hyphenated names (e.g., "delivery-robot", not "DeliveryRobot")
5) Create descriptions that capture the PURPOSE of each type in planning
6) Create descriptions that capture the PURPOSE of each type in planning
7) If the description mentions abstract categories (e.g., "vehicles"), extract them even if not explicitly used
8) Handle implicit hierarchies (e.g., "robots and drones" → both could be "agent" subtypes)

Validation Checks:

- No cycles in hierarchy (a type cannot be its own ancestor)
- Parent types must be defined or be 'object'
- No duplicate type names

Few Shots Example:
Example 1 - Simple flat types: Input: "There are robots, packages, and locations. Robots move and carry packages."

```
Output:
{
  "types": [
    {"name": "robot",
    "parent_type": null,
    "description": "Mobile agent that transports items"},
    {"name": "package",
    "parent_type": null,
    "description": "Object that can be transported"},
    {"name": "location",
    "parent_type": null,
    "description": "Physical position in the environment"}
  ]
}
```

Example 2 - Complex hierarchy: Input: "Vehicles include trucks and airplanes. Trucks are ground vehicles, airplanes are aerial vehicles. Both transport cargo."

```
Output:
{
  "types": [
    {"name": "vehicle",
    "parent_type": null,
    "description": "General transport entity"},
    {"name": "ground-vehicle",
    "parent_type": "vehicle",
    "description": "Vehicle that operates on land"},
    {"name": "aerial-vehicle",
    "parent_type": "vehicle",
    "description": "Vehicle that operates in air"},
    {"name": "truck",
    "parent_type": "ground-vehicle",
    "description": "Ground vehicle for cargo transport"},
    {"name": "airplane",
    "parent_type": "aerial-vehicle",
    "description": "Aerial vehicle for cargo transport"},
    {"name": "cargo",
    "parent_type": null,
    "description": "Goods being transported"}
  ]
}
```

Example 3 - Implicit types: Input: "Agents navigate a grid world with obstacles. Some cells contain items."

```
Output:
{
  "types": [
    {"name": "agent",
    "parent_type": null,
    "description": "Entity that can navigate and interact"},
    {"name": "cell",
    "parent_type": null,
    "description": "Single position in the grid"},
    {"name": "obstacle",
    "parent_type": null,
    "description": "Blocking object in the environment"},
    {"name": "item",
    "parent_type": null,
    "description": "Object that can be collected or manipulated"}
  ]
}
```

Example 4 - Domain with roles: Input: "Players, goalies, and referees are on the field. Players and goalies are team members."

```
Output:
{
  "types": [
    {"name": "person",
    "parent_type": null,
    "description": "Human participant in the game"},
    {"name": "team-member",
    "parent_type": "person",
    "description": "Person belonging to a team"},
    {"name": "player",
    "parent_type": "team-member",
    "description": "Team member who plays the game"},
    {"name": "goalie",
    "parent_type": "team-member",
    "description": "Team member who defends the goal"},
    {"name": "referee",
    "parent_type": "person",
    "description": "Official who enforces rules"},
    {"name": "field", "parent_type": null,
    "description": "Playing area"}
  ]
}
```

Inputs:
NOW PROCESS THIS DOMAIN:
{{domain_description}}
Return ONLY valid JSON matching the schema. No markdown, no explanations.

Output of Types Extraction - BlocksWorld

```
Object type extracted:
{
  "types": [
    {
      "name": "block",
      "parent_type": null,
      "description": "Manipulable object that can be stacked or placed on surfaces"
    },
    {
      "name": "table",
      "parent_type": null,
      "description": "Fixed surface where blocks can be placed"
    },
    {
      "name": "robotic-arm",
      "parent_type": null,
      "description": "Tool used to pick up and move blocks"
    }
  ]
}
```

Prompt for the Attributes Extraction

You are an expert in PDDL predicate and fluent extraction from natural language.
TASK: Extract boolean predicates and numeric fluents for each type, following strict PDDL semantics.

Critical Rules:

1) Each predicate argument MUST have EXACTLY ONE type
2) NEVER use union types like (?x - (block | table)) - THIS IS FORBIDDEN
3) If a concept applies to multiple type combinations, create SEPARATE predicates:
   - (on-table ?b - block) for blocks on table
   - (on-block ?b1 - block ?b2 - block) for blocks on blocks
4) Predicate names should be globally unique OR have different signatures
5) Use descriptive names that clarify the relationship
6) In description, specify full signature: "Predicate: (name ?param1 - type1 ?param2 - type2)"

Critical rules for fluent (numeric):

1) Extract fluents ONLY if explicitly mentioned or strongly implied
2) Specify units and reasonable ranges in description
3) Use hyphens in multi-word names: "battery-level", not "batteryLevel"

Naming conventions:

- Use lowercase with hyphens: "is-loaded", "at-location"
- Avoid ambiguity: prefer "holding-package" over just "holding"
- For binary relations, make direction clear: "robot-at-location" vs "package-at-location"

Validation:

- No attribute name should be repeated with different types across the domain
- Every predicate MUST specify which types it applies to
- Numeric fluents should have plausible ranges

Few Shots Example:
Example 1 - Logistics domain (relationships): Types: robot, package, location Description: "Robots transport packages between locations. Robots have battery (0-100). Packages have weight."

```
Output:
{
  "types_with_attributes": [
    {
      "type_name": "robot",
      "attributes": [
        {"name": "robot-at",
        "type": "boolean",
        "description": "Predicate: (robot-at ?r - robot ?l - location)"},
        {"name": "battery-level",
        "type": "numeric",
        "description": "Battery charge percentage (0-100)"},
        {"name": "carrying-package",
        "type": "boolean",
        "description": "Predicate: (carrying-package ?r - robot ?p - package)"}
      ]
    },
    {
      "type_name": "package",
      "attributes": [
        {"name": "package-at",
        "type": "boolean",
        "description": "Predicate: (package-at ?p - package ?l - location)"},
        {"name": "weight",
        "type": "numeric",
        "description": "Weight in kilograms (0-100)"},
        {"name": "delivered",
        "type": "boolean",
        "description": "True if package reached destination"}
      ]
    },
    {
      "type_name": "location",
      "attributes": [
        {"name": "is-depot",
        "type": "boolean",
        "description": "True if location is a distribution depot"}
      ]
    }
  ]
}
```

Example 2 - Blocksworld (NO hybrid types): Types: block, table, robot-arm Description: "Blocks can be on table or on other blocks. Arm picks/puts blocks."

```
Output:
{
  "types_with_attributes": [
    {
      "type_name": "block",
      "attributes": [
        {"name": "on-table",
        "type": "boolean",
        "description": "Predicate: (on-table ?b - block)"},
        {"name": "on-block",
        "type": "boolean",
        "description": "Predicate: (on-block ?b1 - block ?b2 - block)"},
        {"name": "clear",
        "type": "boolean",
        "description": "True if nothing is on top of this block"},
        {"name": "held-by-arm",
        "type": "boolean",
        "description": "Predicate: (held-by-arm ?b - block ?a - robot-arm)"}
      ]
    },
    {
      "type_name": "table",
      "attributes": []
    },
    {
      "type_name": "robot-arm",
      "attributes": [
        {"name": "arm-empty",
        "type": "boolean",
        "description": "True if arm is not holding anything"}
      ]
    }
  ]
}
```

Example 3 - Rover domain (numeric + boolean): Types: rover, location, sample, camera Description: "Rovers navigate locations, collect samples (with mass), and take images. Rovers have fuel."

```
Output:
{
  "types_with_attributes": [
    {
      "type_name": "rover",
      "attributes": [
        {"name": "rover-at",
        "type": "boolean",
        "description": "Predicate: (rover-at ?r - rover ?l - location)"},
        {"name": "fuel",
        "type": "numeric",
        "description": "Remaining fuel in liters (0-1000)"},
        {"name": "has-sample",
        "type": "boolean",
        "description": "Predicate: (has-sample ?r - rover ?s - sample)"},
        {"name": "equipped-with-camera",
        "type": "boolean",
        "description": "Predicate: (equipped-with-camera ?r - rover ?c - camera)"}
      ]
    },
    {
      "type_name": "location",
      "attributes": [
        {"name": "visited",
        "type": "boolean",
        "description": "True if any rover has been to this location"}
      ]
    },
    {
      "type_name": "sample",
      "attributes": [
        {"name": "sample-at",
        "type": "boolean",
        "description": "Predicate: (sample-at ?s - sample ?l - location)"},
        {"name": "mass",
        "type": "numeric",
        "description": "Sample mass in grams (1-1000)"},
        {"name": "analyzed",
        "type": "boolean",
        "description": "True if sample has been analyzed"}
      ]
    },
    {
      "type_name": "camera",
      "attributes": [
        {"name": "calibrated",
        "type": "boolean",
        "description": "True if camera is ready to use"}
      ]
    }
  ]
}
```

Example 4 - Error case to AVOID:

```
WRONG:
{"name": "on",
"type": "boolean",
"description": "Predicate: (on ?x - (block | table))"}

CORRECT:
{"name": "on-table",
"type": "boolean",
"description": "Predicate: (on-table ?b - block)"}
{"name": "on-block",
"type": "boolean",
"description": "Predicate: (on-block ?b1 - block ?b2 - block)"}
```

Inputs:
NOW PROCESS:
Types (JSON): {{object_types}}
Domain Description: {{domain_description}}
Return ONLY valid JSON. NO union types. Separate predicates for different type combinations.

Output of Attributes Extraction - BlocksWorld

```
Extracted Attributes:
{
  "types_with_attributes": [
    {
      "type_name": "block",
      "attributes": [
        {
          "name": "on-table",
          "type": "boolean",
          "description": "Predicate: (on-table ?b - block ?t - table)"
        },
        {
          "name": "on-block",
          "type": "boolean",
          "description": "Predicate: (on-block ?b1 - block ?b2 - block)"
        },
        {
          "name": "clear",
          "type": "boolean",
          "description": "True if no block is on top of this block"
        },
        {
          "name": "held-by-arm",
          "type": "boolean",
          "description": "Predicate: (held-by-arm ?b - block ?a - robotic-arm)"
        }
      ]
    },
    {
      "type_name": "table",
      "attributes": []
    },
    {
      "type_name": "robotic-arm",
      "attributes": [
        {
          "name": "arm-empty",
          "type": "boolean",
          "description": "True if arm is not holding any block"
        }
      ]
    }
  ]
}
```

Prompt for the Actions Extraction

You are an expert in PDDL action modeling and automated extraction.
TASK: Extract ALL actions with precise parameters, preconditions, and effects using PDDL semantics.

Critical Rules:

1) Parameters: List ALL parameters with ?-prefix and correct types
2) Preconditions: Use EXACT attribute names from the provided attributes
3) Effects: Clearly specify ADD/DELETE for boolean predicates, INCREASE/DECREASE for numeric
4) Be explicit about parameter bindings in conditions/effects
5) Extract both explicit actions AND implied actions (e.g., "load" implies "unload")

Parameter rules:

- Format: "name": "?param", "type": "typename"
- Always use ? prefix
- Include ALL objects involved in the action

Precondition format:

- For boolean predicates: "attribute": "pred-name", "condition": "(?p1 ?p2 ...) = true" or "= false"
- For numeric: "attribute": "fuel", "condition": "> 0" or ">= 10", "< 100", etc.
- Be specific about which parameters are used

Effect format:

- Boolean add: "attribute": "at", "change": "add (?r ?loc)"
- Boolean delete: "attribute": "at", "change": "delete (?r ?old-loc)"
- Numeric increase: "attribute": "fuel", "change": "decrease by 5"
- Numeric set: "attribute": "battery", "change": "set to 100"

Few Shots Example:
Example 1 - Move action: Domain: "Robot moves between locations, consuming 1 battery per move. Must have battery > 0." Types: robot, location Attributes: robot-at (predicate ?r ?l), battery-level (numeric 0-100)

```
Output:
{
  "actions": [
    {
      "name": "move",
      "actor": "robot",
      "parameters": [
        {"name": "?r", "type": "robot"},
        {"name": "?from", "type": "location"},
        {"name": "?to", "type": "location"}
      ],
      "preconditions": [
        {"attribute": "robot-at", "condition": "(?r ?from) = true"},
        {"attribute": "battery-level", "condition": "> 0"}
      ],
      "effects": [
        {"attribute": "robot-at", "change": "delete (?r ?from)"},
        {"attribute": "robot-at", "change": "add (?r ?to)"},
        {"attribute": "battery-level", "change": "decrease by 1"}
      ],
      "description": "Move robot from one location to another, consuming battery"
    }
  ]
}
```

Example 2 - Pickup and drop (implied pair): Domain: "Robot picks up package at location. Can carry one at a time." Types: robot, package, location Attributes: robot-at, package-at, carrying-package, robot-empty

```
Output:
{
  "actions": [
    {
      "name": "pickup",
      "actor": "robot",
      "parameters": [
        {"name": "?r", "type": "robot"},
        {"name": "?p", "type": "package"},
        {"name": "?loc", "type": "location"}
      ],
      "preconditions": [
        {"attribute": "robot-at", "condition": "(?r ?loc) = true"},
        {"attribute": "package-at", "condition": "(?p ?loc) = true"},
        {"attribute": "robot-empty", "condition": "(?r) = true"}
      ],
      "effects": [
        {"attribute": "carrying-package", "change": "add (?r ?p)"},
        {"attribute": "package-at", "change": "delete (?p ?loc)"},
        {"attribute": "robot-empty", "change": "delete (?r)"}
      ],
      "description": "Robot picks up a package from the current location"
    },
    {
      "name": "drop",
      "actor": "robot",
      "parameters": [
        {"name": "?r", "type": "robot"},
        {"name": "?p", "type": "package"},
        {"name": "?loc", "type": "location"}
      ],
      "preconditions": [
        {"attribute": "robot-at", "condition": "(?r ?loc) = true"},
        {"attribute": "carrying-package", "condition": "(?r ?p) = true"}
      ],
      "effects": [
        {"attribute": "carrying-package", "change": "delete (?r ?p)"},
        {"attribute": "package-at", "change": "add (?p ?loc)"},
        {"attribute": "robot-empty", "change": "add (?r)"}
      ],
      "description": "Robot drops a carried package at the current location"
    }
  ]
}
```

Example 3 - Blocksworld stack/unstack: Domain: "Stack block A on block B. A must be held, B must be clear." Types: block, robot-arm Attributes: held-by-arm, on-block, on-table, clear, arm-empty

```
Output:
{
  "actions": [
    {
      "name": "stack",
      "actor": "robot-arm",
      "parameters": [
        {"name": "?arm", "type": "robot-arm"},
        {"name": "?b1", "type": "block"},
        {"name": "?b2", "type": "block"}
      ],
      "preconditions": [
        {"attribute": "held-by-arm", "condition": "(?b1 ?arm) = true"},
        {"attribute": "clear", "condition": "(?b2) = true"}
      ],
      "effects": [
        {"attribute": "held-by-arm", "change": "delete (?b1 ?arm)"},
        {"attribute": "on-block", "change": "add (?b1 ?b2)"},
        {"attribute": "clear", "change": "delete (?b2)"},
        {"attribute": "clear", "change": "add (?b1)"},
        {"attribute": "arm-empty", "change": "add (?arm)"}
      ],
      "description": "Stack a held block on top of another clear block"
    },
    {
      "name": "unstack",
      "actor": "robot-arm",
      "parameters": [
        {"name": "?arm", "type": "robot-arm"},
        {"name": "?b1", "type": "block"},
        {"name": "?b2", "type": "block"}
      ],
      "preconditions": [
        {"attribute": "on-block", "condition": "(?b1 ?b2) = true"},
        {"attribute": "clear", "condition": "(?b1) = true"},
        {"attribute": "arm-empty", "condition": "(?arm) = true"}
      ],
      "effects": [
        {"attribute": "on-block", "change": "delete (?b1 ?b2)"},
        {"attribute": "held-by-arm", "change": "add (?b1 ?arm)"},
        {"attribute": "clear", "change": "add (?b2)"},
        {"attribute": "clear", "change": "delete (?b1)"},
        {"attribute": "arm-empty", "change": "delete (?arm)"}
      ],
      "description": "Unstack a block from another and hold it"
    }
  ]
}
```

Inputs:
NOW EXTRACT ACTIONS FROM:
Domain Description: {{domain_description}}
Types (JSON): {{object_types}}
Attributes (JSON): {{object_attributes}}
Return ONLY valid JSON with all actions, including implied inverse actions where appropriate.

Output of Actions Extraction - BlocksWorld

```
Extracted actions:
{
  "actions": [
    {
      "name": "pick-up-from-table",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "on-table",
          "condition": "(?b ?t) = true"
        },
        {
          "attribute": "clear",
          "condition": "(?b) = true"
        },
        {
          "attribute": "arm-empty",
          "condition": "(?a) = true"
        }
      ],
      "effects": [
        {
          "attribute": "on-table",
          "change": "delete (?b ?t)"
        },
        {
          "attribute": "clear",
          "change": "delete (?b)"
        },
        {
          "attribute": "held-by-arm",
          "change": "add (?b ?a)"
        },
        {
          "attribute": "arm-empty",
          "change": "delete (?a)"
        }
      ],
      "description": "Pick up a block from the table using the robotic arm"
    },
    {
      "name": "put-down-on-table",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "held-by-arm",
          "condition": "(?b ?a) = true"
        }
      ],
      "effects": [
        {
          "attribute": "on-table",
          "change": "add (?b ?t)"
        },
        {
          "attribute": "clear",
          "change": "add (?b)"
        },
        {
          "attribute": "held-by-arm",
          "change": "delete (?b ?a)"
        },
        {
          "attribute": "arm-empty",
          "change": "add (?a)"
        }
      ],
      "description": "Put down a held block on the table using the robotic arm"
    },
    {
      "name": "stack",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "held-by-arm",
          "condition": "(?b1 ?a) = true"
        },
        {
          "attribute": "clear",
          "condition": "(?b2) = true"
        }
      ],
      "effects": [
        {
          "attribute": "held-by-arm",
          "change": "delete (?b1 ?a)"
        },
        {
          "attribute": "on-block",
          "change": "add (?b1 ?b2)"
        },
        {
          "attribute": "clear",
          "change": "delete (?b2)"
        },
        {
          "attribute": "clear",
          "change": "add (?b1)"
        },
        {
          "attribute": "arm-empty",
          "change": "add (?a)"
        }
      ],
      "description": "Stack a held block on top of another clear block"
    },
    {
      "name": "unstack",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "on-block",
          "condition": "(?b1 ?b2) = true"
        },
        {
          "attribute": "clear",
          "condition": "(?b1) = true"
        },
        {
          "attribute": "arm-empty",
          "condition": "(?a) = true"
        }
      ],
      "effects": [
        {
          "attribute": "on-block",
          "change": "delete (?b1 ?b2)"
        },
        {
          "attribute": "held-by-arm",
          "change": "add (?b1 ?a)"
        },
        {
          "attribute": "clear",
          "change": "delete (?b1)"
        },
        {
          "attribute": "clear",
          "change": "add (?b2)"
        },
        {
          "attribute": "arm-empty",
          "change": "delete (?a)"
        }
      ],
      "description": "Unstack a clear block from on top of another block"
    }
  ]
}
```

Prompt for the Actions Refinement

You are an expert PDDL domain engineer specializing in semantic inference and action refinement.

Critical Rules:

1) **PRESERVE ALL ACTIONS**: Output count MUST equal or exceed input count. NEVER remove actions.
2) **INFER IMPLICIT CONSTRAINTS**: Add missing preconditions based on domain semantics
3) **COMPLETE EFFECTS**: Ensure all state changes are explicitly listed
4) **CONSISTENT NAMING**: Use EXACT attribute names from the provided attributes

Refinement rules:

1) Precondition Inference
    - If domain mentions "obstacles", "blocked", "occupied" → add clearance checks
    - If domain mentions "capacity", "limit", "full" → add resource constraints
    - If action modifies object state → check current state first
    - If action needs empty/available resource → add availability check
    - If locations/positions matter → add location preconditions
2) Effect completeness
    - For movement: delete old location, add new location
    - For pickup: change holder, remove from previous location
    - For resource consumption: decrease/increase numeric fluents
    - For state changes: add new state, delete old state
3) ADL detection
   Set requires_adl = "true" if:
    - Action affects ALL objects of a type (e.g., "unload all packages")
    - Effects depend on conditions (e.g., "damage if collision")
    - Quantifiers needed (forall, exists)

   Otherwise: requires_adl = "false"
4) Parameter format
    - Always include ? prefix: "?r", "?from", "?to"
    - Match types exactly from object_types input
    - Include ALL objects involved in the action
5) Condition/Effect format
   Precondition:
    - Predicates: "(?param1 ?param2) = true" or "(?param) = false"
    - Numeric: "> 0", ">= 5", "< 100", "!= 0"

   Effect:
    - Add predicate: "add (?p1 ?p2)"
    - Delete predicate: "delete (?p1 ?p2)"
    - Increase: "increase by 5"
    - Decrease: "decrease by 3"
    - Set: "set to 100"

Few Shots Example:
Example 1: Simple movement with inferred clearance

```
Input Action:
{
  "name": "move",
  "actor": "robot",
  "parameters": [{"name": "?r", "type": "robot"},
  {"name": "?from", "type": "location"},
  {"name": "?to", "type": "location"}],
  "preconditions": [{"attribute": "at", "condition": "(?r ?from) = true"}],
  "effects": [{"attribute": "at", "change": "delete (?r ?from)"},
  {"attribute": "at", "change": "add (?r ?to)"}],
  "description": "Move robot"
}
```

Domain mentions: "Some locations are blocked by obstacles"

```
Expected Output:
{
  "name": "move",
  "description": "Move robot from one location to another if destination is
  clear",
  "actor": "robot",
  "parameters": [
    {"name": "?r", "type": "robot"},
    {"name": "?from", "type": "location"},
    {"name": "?to", "type": "location"}
  ],
  "preconditions": [
    {"attribute": "at", "condition": "(?r ?from) = true"},
    {"attribute": "clear", "condition": "(?to) = true"}
  ],
  "effects": [
    {"attribute": "at", "change": "delete (?r ?from)"},
    {"attribute": "at", "change": "add (?r ?to)"}
  ],
  "requires_adl": "false"
}
```

—

Example 2: Pickup with implicit preconditions Input Action:

```
{
  "name": "pickup",
  "actor": "robot",
  "parameters": [{"name": "?r", "type": "robot"},
  {"name": "?p", "type": "package"}],
  "preconditions": [],
  "effects": [{"attribute": "holding", "change": "add (?r ?p)"}],
  "description": "Pick up package"
}
```

Domain mentions: "Robot can carry one package at a time. Robot and package must be at same location."

```
Expected Output:
{
  "name": "pickup",
  "description": "Robot picks up a package from the same location if hands are
  empty",
  "actor": "robot",
  "parameters": [
    {"name": "?r", "type": "robot"},
    {"name": "?p", "type": "package"},
    {"name": "?loc", "type": "location"}
  ],
  "preconditions": [
    {"attribute": "at", "condition": "(?r ?loc) = true"},
    {"attribute": "at", "condition": "(?p ?loc) = true"},
    {"attribute": "hands-empty", "condition": "(?r) = true"}
  ],
  "effects": [
    {"attribute": "holding", "change": "add (?r ?p)"},
    {"attribute": "at", "change": "delete (?p ?loc)"},
    {"attribute": "hands-empty", "change": "delete (?r)"}
  ],
  "requires_adl": "false"
}
```

—

Example 3: Action with numeric fluent

```
Input Action:
{
  "name": "drive",
  "actor": "truck",
  "parameters": [{"name": "?t", "type": "truck"},
  {"name": "?from", "type": "city"},
  {"name": "?to", "type": "city"}],
  "preconditions": [{"attribute": "truck-at", "condition": "(?t ?from) = true"}],
  "effects": [{"attribute": "truck-at", "change": "delete (?t ?from)"},
  {"attribute": "truck-at", "change": "add (?t ?to)"}],
  "description": "Drive truck"
}
```

Domain mentions: "Trucks consume 10 fuel per trip. Fuel ranges 0-100."

```
Expected Output:
{
  "name": "drive",
  "description": "Drive truck between cities consuming fuel",
  "actor": "truck",
  "parameters": [
    {"name": "?t", "type": "truck"},
    {"name": "?from", "type": "city"},
    {"name": "?to", "type": "city"}
  ],
  "preconditions": [
    {"attribute": "truck-at", "condition": "(?t ?from) = true"},
    {"attribute": "fuel", "condition": ">= 10"}
  ],
  "effects": [
    {"attribute": "truck-at", "change": "delete (?t ?from)"},
    {"attribute": "truck-at", "change": "add (?t ?to)"},
    {"attribute": "fuel", "change": "decrease by 10"}
  ],
  "requires_adl": "false"
}
```

—

Example 4: ADL action (affects multiple objects)

```
Input Action:
{
  "name": "unload-all",
  "actor": "truck",
  "parameters": [{"name": "?t", "type": "truck"},
  {"name": "?loc", "type": "location"}],
  "preconditions": [{"attribute": "truck-at", "condition": "(?t ?loc) = true"}],
  "effects": [{"attribute": "in-truck", "change": "delete all packages from ?t"}],
  "description": "Unload all packages"
}
```

Expected Output:

```
{
  "name": "unload-all",
  "description": "Unload all packages from truck to current location",
  "actor": "truck",
  "parameters": [
    {"name": "?t", "type": "truck"},
    {"name": "?loc", "type": "location"}
  ],
  "preconditions": [
    {"attribute": "truck-at", "condition": "(?t ?loc) = true"}
  ],
  "effects": [
    {"attribute": "in-truck", "change": "delete (?p ?t) for all ?p"},
    {"attribute": "package-at", "change": "add (?p ?loc) for all ?p in ?t"}
  ],
  "requires_adl": "true"
}
```

— **Validation Checklist:**

- Every parameter has ? prefix and valid type
- Every attribute name exists in the attributes input
- Preconditions use correct format: "(?p1 ?p2) = true/false" or "> N"
- Effects use correct format: "add/delete (?p1 ?p2)" or "increase/decrease by N"
- ADL flag is "true" or "false" (string, not boolean)

—
NOW REFINE THE ACTIONS given:
Domain Description: {{ domain_description }}
Valid Types (JSON): {{ object_types }}
Valid Attributes/Predicates (JSON): {{ attributes }}
Actions to Refine (JSON): {{ actions_list }}
Step 1: Count input actions Step 2: For EACH action, infer missing preconditions from domain semantics Step 3: Complete all effects Step 4: Determine if ADL is needed Step 5: Verify output count >= input count
Return ONLY valid JSON matching the RefinedActionsResult schema:

```
{
  "actions": [
    {
      "name": "action-name",
      "description": "what it does",
      "actor": "type-name",
      "parameters": [{"name": "?param", "type": "type"}],
      "preconditions": [{"attribute": "pred-name", "condition": "(?p) = true"}],
      "effects": [{"attribute": "pred-name", "change": "add (?p1 ?p2)"}],
      "requires_adl": "false"
    }
  ]
}
```

Output of Actions Refinement - BlocksWorld

As anticipated, this step is not necessary for simple, well-defined domains like BlocksWorld, in fact in this case the output of the refined actions is the same as step C.

```
Extracted actions:
{
  "actions": [
    {
      "name": "pick-up-from-table",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "on-table",
          "condition": "(?b ?t) = true"
        },
        {
          "attribute": "clear",
          "condition": "(?b) = true"
        },
        {
          "attribute": "arm-empty",
          "condition": "(?a) = true"
        }
      ],
      "effects": [
        {
          "attribute": "on-table",
          "change": "delete (?b ?t)"
        },
        {
          "attribute": "clear",
          "change": "delete (?b)"
        },
        {
          "attribute": "held-by-arm",
          "change": "add (?b ?a)"
        },
        {
          "attribute": "arm-empty",
          "change": "delete (?a)"
        }
      ],
      "description": "Pick up a block from the table using the robotic arm"
    },
    {
      "name": "put-down-on-table",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "held-by-arm",
          "condition": "(?b ?a) = true"
        }
      ],
      "effects": [
        {
          "attribute": "on-table",
          "change": "add (?b ?t)"
        },
        {
          "attribute": "clear",
          "change": "add (?b)"
        },
        {
          "attribute": "held-by-arm",
          "change": "delete (?b ?a)"
        },
        {
          "attribute": "arm-empty",
          "change": "add (?a)"
        }
      ],
      "description": "Put down a held block on the table using the robotic arm"
    },
    {
      "name": "stack",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "held-by-arm",
          "condition": "(?b1 ?a) = true"
        },
        {
          "attribute": "clear",
          "condition": "(?b2) = true"
        }
      ],
      "effects": [
        {
          "attribute": "held-by-arm",
          "change": "delete (?b1 ?a)"
        },
        {
          "attribute": "on-block",
          "change": "add (?b1 ?b2)"
        },
        {
          "attribute": "clear",
          "change": "delete (?b2)"
        },
        {
          "attribute": "clear",
          "change": "add (?b1)"
        },
        {
          "attribute": "arm-empty",
          "change": "add (?a)"
        }
      ],
      "description": "Stack a held block on top of another clear block"
    },
    {
      "name": "unstack",
      "actor": "robotic-arm",
      "preconditions": [
        {
          "attribute": "on-block",
          "condition": "(?b1 ?b2) = true"
        },
        {
          "attribute": "clear",
          "condition": "(?b1) = true"
        },
        {
          "attribute": "arm-empty",
          "condition": "(?a) = true"
        }
      ],
      "effects": [
        {
          "attribute": "on-block",
          "change": "delete (?b1 ?b2)"
        },
        {
          "attribute": "held-by-arm",
          "change": "add (?b1 ?a)"
        },
        {
          "attribute": "clear",
          "change": "delete (?b1)"
        },
        {
          "attribute": "clear",
          "change": "add (?b2)"
        },
        {
          "attribute": "arm-empty",
          "change": "delete (?a)"
        }
      ],
      "description": "Unstack a clear block from on top of another block"
    }
  ]
}
```

Prompt for the PDDL Domain Generation

You are a PDDL domain code generator with perfect syntax knowledge.
TASK: Generate a complete, valid, and executable PDDL domain file.

Critical Syntax Rules:

1) Perfect parenthesis balance - every (must have matching)
2) Proper spacing: (define (domain name)) NOT (define(domain name))
3) Types section: use " - " (space-dash-space) for inheritance
4) Predicate signatures: EXACTLY ONE type per parameter, NO union types
5) Action parameters: MUST have " - " before type
6) Effects: use (not (pred)) for deletion, NOT (delete (pred))

Structure requirements:

```
(define (domain DOMAIN-NAME)
  (:requirements :strips :typing :negative-preconditions :equality)

  (:types
    type1 type2 - parent-type
    type3 - object
  )

  (:predicates
    (pred-name ?param - type)
    (binary-pred ?p1 - type1 ?p2 - type2)
  )

  (:functions
    (func-name ?param - type) - number
  )  ;; ONLY if numeric fluents exist

  (:action action-name
    :parameters (?p1 - type1 ?p2 - type2)
    :precondition (and
      (pred1 ?p1)
      (pred2 ?p1 ?p2)
      (> (func ?p1) 0)  ;; for numeric
    )
    :effect (and
      (pred3 ?p2)
      (not (pred1 ?p1))
      (increase (func ?p1) 5)  ;; for numeric
    )
  )
)
```

Predicate conversion rules
Input attribute format: "Predicate: (robot-at ?r - robot ?l - location)"
Output PDDL: (robot-at ?r - robot ?l - location)

Effect conversion

- "add (?r ?loc)" → (robot-at ?r ?loc)
- "delete (?r ?from)" → (not (robot-at ?r ?from))
- "increase by 5" → (increase (attribute-name ?param) 5)
- "decrease by 3" → (decrease (attribute-name ?param) 3)

Common Errors to avoid

- (on ?b - (block | table)) → Use separate predicates
- (on-table ?b - block) → Ensure consistent with attributes (on ?b1 ?b2 - block)
- :parameters (?r robot) → :parameters (?r - robot)
- (delete (at ?r ?l)) → (not (at ?r ?l))

Requirements section:

- Always: :strips :typing
- If negated preconditions: :negative-preconditions
- If numeric fluents: :fluents :numeric-fluents
- If conditional effects: :conditional-effects
- If equality checks: :equality
- If requires_adl=true: :adl

Few Shots Example:

EXAMPLE 1 - Blocksworld: Input Types: block, robot-arm Input Predicates: on-table(?b - block), on-block(?b1 - block ?b2 - block), clear(?b - block), arm-empty(?arm - robot-arm), held-by-arm(?b - block ?arm - robot-arm) Input Actions: pickup(from table), stack, unstack, putdown
Output:

```
(define (domain blocksworld)
  (:requirements :strips :typing :negative-preconditions)

  (:types
    block
    robot-arm
  )

  (:predicates
    (on-table ?b - block)
    (on-block ?b1 - block ?b2 - block)
    (clear ?b - block)
    (arm-empty ?arm - robot-arm)
    (held-by-arm ?b - block ?arm - robot-arm)
  )

  (:action pickup-from-table
    :parameters (?arm - robot-arm ?b - block)
    :precondition (and
      (on-table ?b)
      (clear ?b)
      (arm-empty ?arm)
    )
    :effect (and
      (not (on-table ?b))
      (not (arm-empty ?arm))
      (held-by-arm ?b ?arm)
    )
  )

  (:action putdown-on-table
    :parameters (?arm - robot-arm ?b - block)
    :precondition (held-by-arm ?b ?arm)
    :effect (and
      (on-table ?b)
      (clear ?b)
      (arm-empty ?arm)
      (not (held-by-arm ?b ?arm))
    )
  )

  (:action stack
    :parameters (?arm - robot-arm ?b1 - block ?b2 - block)
    :precondition (and
      (held-by-arm ?b1 ?arm)
      (clear ?b2)
    )
    :effect (and
      (on-block ?b1 ?b2)
      (clear ?b1)
      (arm-empty ?arm)
      (not (held-by-arm ?b1 ?arm))
      (not (clear ?b2))
    )
  )

  (:action unstack
    :parameters (?arm - robot-arm ?b1 - block ?b2 - block)
    :precondition (and
      (on-block ?b1 ?b2)
      (clear ?b1)
      (arm-empty ?arm)
    )
    :effect (and
      (held-by-arm ?b1 ?arm)
      (clear ?b2)
      (not (on-block ?b1 ?b2))
      (not (clear ?b1))
      (not (arm-empty ?arm))
    )
  )
)
```

EXAMPLE 2 - Logistics with numeric (fuel): Input includes: battery-level (numeric 0-100)
Output:

```
(define (domain logistics)
  (:requirements :strips :typing :fluents :numeric-fluents)

  (:types
    robot package location
  )

  (:predicates
    (robot-at ?r - robot ?l - location)
    (package-at ?p - package ?l - location)
    (carrying ?r - robot ?p - package)
  )

  (:functions
    (battery-level ?r - robot) - number
  )

  (:action move
    :parameters (?r - robot ?from - location ?to - location)
    :precondition (and
      (robot-at ?r ?from)
      (> (battery-level ?r) 0)
    )
    :effect (and
      (not (robot-at ?r ?from))
      (robot-at ?r ?to)
      (decrease (battery-level ?r) 1)
    )
  )

  (:action pickup
    :parameters (?r - robot ?p - package ?loc - location)
    :precondition (and
      (robot-at ?r ?loc)
      (package-at ?p ?loc)
    )
    :effect (and
      (carrying ?r ?p)
      (not (package-at ?p ?loc))
    )
  )
)
```

Inputs:
NOW GENERATE PDDL FROM:
Domain Description: {{domain_description}}
Types (NL): {{objects}}
Predicates (NL): {{attributes}}
Actions (NL): {{actions}}
CRITICAL: Return ONLY valid JSON: "domain_name": "lowercase-name", "pddl_code": "complete PDDL code as single string with
n for newlines", "explanation": "brief explanation of key design decisions"
Verify parenthesis balance before returning. Ensure types match across predicates and actions.

Output of PDDL Domain Generation - BlocksWorld

```
(define (domain blocksworld)
  (:requirements :strips :typing :negative-preconditions)
  (:types
    block
    table
    robotic-arm
  )
  (:predicates
    (on-table ?b - block ?t - table)
    (on-block ?b1 - block ?b2 - block)
    (clear ?b - block)
    (held-by-arm ?b - block ?a - robotic-arm)
    (arm-empty ?a - robotic-arm)
  )
  (:action pick-up-from-table
    :parameters (?b - block ?t - table ?a - robotic-arm)
    :precondition (and
      (on-table ?b ?t)
      (clear ?b)
      (arm-empty ?a)
    )
    :effect (and
      (not (on-table ?b ?t))
      (not (clear ?b))
      (held-by-arm ?b ?a)
      (not (arm-empty ?a))
    )
  )
  (:action put-down-on-table
    :parameters (?b - block ?t - table ?a - robotic-arm)
    :precondition (and
      (held-by-arm ?b ?a)
    )
    :effect (and
      (on-table ?b ?t)
      (clear ?b)
      (arm-empty ?a)
      (not (held-by-arm ?b ?a))
    )
  )
  (:action stack
    :parameters (?b1 - block ?b2 - block ?a - robotic-arm)
    :precondition (and
      (held-by-arm ?b1 ?a)
      (clear ?b2)
    )
    :effect (and
      (on-block ?b1 ?b2)
      (clear ?b1)
      (arm-empty ?a)
      (not (held-by-arm ?b1 ?a))
      (not (clear ?b2))
    )
  )
  (:action unstack
    :parameters (?b1 - block ?b2 - block ?a - robotic-arm)
    :precondition (and
      (on-block ?b1 ?b2)
      (clear ?b1)
      (arm-empty ?a)
    )
    :effect (and
      (held-by-arm ?b1 ?a)
      (clear ?b2)
      (not (arm-empty ?a))
      (not (on-block ?b1 ?b2))
      (not (clear ?b1))
    )
  )
)
```

LLM-as-a-Judge

Prompt for evaluation with LLM-as-a-judge

You are an expert in PDDL and natural language understanding, acting as a rigorous evaluator. Your task is to provide a quantitative assessment (0-100 points) of how well the GENERATED PDDL DOMAIN file captures the generalized rules, entities, and actions from the provided natural language (NL) description. Carefully analyze the provided materials and return ONLY a single JSON object matching this schema: {schema_format}.
Use the following rules to determine the scores:
1. Domain Modeling Score (key: "Domain_Modeling_Score", 0-100 points): This score assesses the quality and semantic correctness of the PDDL domain file based on the NL description.
- (30 pts) Predicates: Are the predicates accurately, completely, and non-redundantly representing the described properties and relations (e.g., avoiding highly generic predicates or using correct arity)? - (35 pts) Actions Logic: Do the action schemas (preconditions and effects) make perfect logical and functional sense for the domain? Do they comply with implicit common sense constraints implied by the NL description? - (35 pts) Generality and Sufficiency: Are the defined actions and predicates sufficiently generalized to represent ALL capabilities mentioned or implied by the domain description, ensuring the domain is actionable (solvable) for typical tasks derived from the description?
2. Rationale (key: "Rationale", string): Provide a concise but detailed explanation, justifying all deductions, and referencing specific issues in predicate definitions, action preconditions, or missing inverse actions.
The inputs for your evaluation are as follows:
1. Natural Language Environment Scenario: {domain_description}
2. Generated PDDL Domain: {pddl_domain_code}

Output of Evaluation with GPT 4.1 - BlocksWorld

```
{ "Domain_Modeling_Score": 92,
  "Rationale": "The domain file models the classic Blocks World scenario
  with high fidelity. Predicates are well-chosen: ’on-table’, ’on-block’, ’clear’,
  ’held-by-arm’, and ’arm-empty’ accurately capture the necessary state
  information without redundancy, and arities are correct (30/30).
  The action schemas (’pick-up-from-table’, ’put-down-on-table’, ’stack’,
  ’unstack’) have logically sound preconditions and effects, enforcing
  constraints such as only moving clear blocks and requiring the arm to be empty
  before picking up (34/35). The domain is sufficiently general for all
  standard Blocks World tasks, with all necessary actions and no missing inverse
  actions. The only minor deduction is for the explicit modeling of ’table’
  and ’robotic-arm’ as types and parameters, which is not strictly necessary in
  the classic formulation (usually a single table and arm are assumed), but this
  does not affect correctness or solvability (28/35). Overall, the domain is
  highly accurate and actionable." }
```

Output of Evaluation with Gemini 2.5 PRO - BlocksWorld

```
{ "Domain_Modeling_Score": 100,
  "Rationale": "The generated PDDL domain provides a semantically perfect
  representation of the Blocks World scenario described.
  Predicates & Types: The domain correctly types `table` as an object, which
  generalizes the scenario to potentially allow multiple tables, while
  maintaining correct state tracking with `(on-table ?b ?t)`. The predicates
  cover all necessary states (clear, held, on-block). Action Logic: The logic is
  sound. The `stack` and `unstack` actions correctly manage the `clear` predicate
  to enforce the ’only one block on top’ rule. The `pick-up` and `put-down`
  actions correctly handle transitions between the table and the arm.
  Completeness: The actions cover all described manipulations. The explicit
  parameterization of the table in `put-down-on-table` ensures the robot
  can place blocks on any valid table object defined in a problem file." }
```