Describe your algorithm for each question in the report.
• Q1 – Logic Warm-up (2%)
In Question 1, I was required to implement six functions that manipulate and assess logical expressions.
For sentence1, sentence2, and sentence3, I completed the functions according to the descriptions provided. Each function constructs a logical expression (Expr) based on specified conditions:
In sentence1, sentence2, and sentence3, the goal was to encapsulate complex logical relationships using logical operators like AND, OR, and NOT, directly reflecting the given logical statements into programmable expressions.
In the findModelUnderstandingCheck function, I created a new type of expression, named FakeExpr, to handle lowercase expressions which aren't typically supported. This function returns a dictionary that maps this expression to a boolean value, demonstrating a simplified model generation based on custom expression types.
The entails function determines logical entailment by attempting to find a model where the premise is true and the conclusion is false. If such a model exists, the function returns False, indicating the premise does not entail the conclusion. Otherwise, it returns True.
The plTrueInverse function utilizes the pl_true function to evaluate whether the negation of inverse_statement holds true given a set of assignments. This function aids in assessing conditions under negated circumstances within logical expressions.

```
Question q1
===========

*** PASS: test_cases/q1/correctSentence1.test
***     PASS
*** PASS: test_cases/q1/correctSentence2.test
***     PASS
*** PASS: test_cases/q1/correctSentence3.test
***     PASS
*** PASS: test_cases/q1/entails.test
***     PASS
*** PASS: test_cases/q1/entailsLong.test
***     PASS
*** PASS: test_cases/q1/findModelSentence1.test
***     PASS
*** PASS: test_cases/q1/findModelSentence2.test
***     PASS
*** PASS: test_cases/q1/findModelSentence3.test
***     PASS
*** PASS: test_cases/q1/findModelUnderstandingCheck.test
***     PASS
*** PASS: test_cases/q1/plTrueInverse.test
***     PASS

### Question q1: 10/10 ###
```

• Q2 – Logic Workout (2%)
In Question 2, I implemented three functions designed to express specific logical constraints using propositional logic:

atLeastOne function: This function returns disjoin(*literals), which creates a disjunction (OR operation) of all provided literals. The purpose is to ensure that at least one of the literals evaluates to true.
atMostOne function: In this function, I construct expressions to ensure that no more than one of the given literals can be true at the same time. This is achieved by creating pairwise conjunctions (AND operations) of the negations of all literals, ensuring that it is not possible for two literals to be true simultaneously.
exactlyOne function: This function combines the constraints of atLeastOne and atMostOne. It returns an expression that is true only if exactly one of the provided literals is true. This is implemented by taking the logical AND of the results from atLeastOne and atMostOne.

Together, these functions provide a robust framework for handling logical constraints in decision-making systems, where the number of true conditions needs to be carefully controlled to match specific requirements.

```
Question q2
===========

*** PASS: test_cases/q2/atLeastOne.test
***     PASS
*** PASS: test_cases/q2/atLeastOneCNF.test
***     PASS
*** PASS: test_cases/q2/atLeastOneEff.test
***     PASS
*** PASS: test_cases/q2/atMostOne.test
***     PASS
*** PASS: test_cases/q2/atMostOneCNF.test
***     PASS
*** PASS: test_cases/q2/atMostOneEff.test
***     PASS
*** PASS: test_cases/q2/exactlyOne.test
***     PASS
*** PASS: test_cases/q2/exactlyOneCNF.test
***     PASS
*** PASS: test_cases/q2/exactlyOneEff.test
***     PASS

### Question q2: 10/10 ###
```

• Q3 – Pacphysics and Satisfiability (2%)

In this question, we focused on developing logical constraints and models specific to a Pacman game. We implemented the following three functions.

- pacmanSuccessorAxiomSingle: It determines Pacman's current state and possible successor states based on adjacent coordinates and potential directions of movement. The logic checks for wall obstacles and calculates permissible movements into adjacent squares. If movement in a particular direction is possible (i.e., no wall blocks the path), that direction and the resultant new position become part of the potential successors.
- pacphysicsAxioms: This function constructs the overall game logic at a given time step t. It involves:
    - Ensuring Pacman cannot occupy a space with a wall.
    - Asserting that Pacman must be in exactly one position per time step.
    - Ensuring exactly one movement action is taken by Pacman at each step.
    - Integrating sensor model information if available, to reflect perceptions in the logic.
    - If provided, adding successor axioms to describe transitions between states based on Pacman's actions and the physical layout of the maze.
- checkLocationSatisfiability: This function aims to establish the feasibility of Pacman being at a certain position (x1, y1) at time t=1 while considering his previous position (x0, y0) and actions taken. It includes:
    - Adding knowledge about wall locations to the knowledge base (KB).
    - Encoding Pacman's movements from his initial to the potential new position.
    - Incorporating the necessary pacphysics axioms to enforce game rules and logic.
    - Querying the model to confirm whether it's possible (or not) for Pacman to be in the proposed position at the next time step.

```
Question q3
===========

*** Testing checkLocationSatisfiability
[LogicAgent] using problem type LocMapProblem
Ending game
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** PASS: test_cases/q3/location_satisfiability1.test
*** Testing checkLocationSatisfiability
[LogicAgent] using problem type LocMapProblem
Ending game
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** PASS: test_cases/q3/location_satisfiability2.test
*** Testing pacphysicsAxioms
*** PASS: test_cases/q3/pacphysics1.test
*** Testing pacphysicsAxioms
*** PASS: test_cases/q3/pacphysics2.test
*** PASS: test_cases/q3/pacphysics_transition.test
***     PASS

### Question q3: 10/10 ###
```

• Q4 – Path Planning with Logic (2%)

In this question, the task was to utilize logical planning to determine a viable path for Pacman from a specified starting point to a designated goal position, taking into account the presence of walls and adhering to previously established game rules. Here's a detailed breakdown of the algorithm used:

- Initial Setup:
- Location Constraint: At each timestep t, ensure Pacman occupies exactly one of the possible coordinates that are not blocked by walls. This is encoded using the exactlyOne function, which ensures a unique position for Pacman at each timestep.
- Model Checking:
- Goal Check: At each timestep t, the model is queried to check if it satisfies the condition of Pacman being at the goal coordinate. If such a model is found, it signifies that a path leading to the goal within t timesteps exists.
- Extract Actions: If a model is found, the sequence of actions leading to this model is extracted and returned. This sequence represents the path that Pacman must follow to reach the goal.
- Action and Successor Constraints:
- Action Constraint: For each timestep t, ensure that Pacman takes exactly one of the possible movement actions (North, South, East, West). This is also managed by the exactlyOne function, applied to the set of actions.
- Successor State Axiom: For each possible location and for each timestep, the successors for the next timestep are computed using the pacmanSuccessorAxiomSingle function. This function calculates the next possible positions based on the current position and the absence of walls blocking potential movements.
- Termination:
- If no satisfying model is found within the predefined number of timesteps (often set as a reasonable upper limit like 50 to prevent infinite computation), the function returns None, indicating that no path could be established under the given constraints.

```
Question q4
===========

[LogicAgent] using problem type PositionPlanningProblem
Path found with total cost of 2 in 0.0 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 508
Average Score: 508.0
Scores:        508.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases/q4/positionLogicPlan1.test
***     pacman layout:          maze2x2
***     solution score:         508
***     solution path:          West South
[LogicAgent] using problem type PositionPlanningProblem
Path found with total cost of 8 in 0.2 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases/q4/positionLogicPlan2.test
***     pacman layout:          tinyMaze
***     solution score:         502
***     solution path:          South South West South West West South West
[LogicAgent] using problem type PositionPlanningProblem
Path found with total cost of 19 in 18.5 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases/q4/positionLogicPlan3.test
***     pacman layout:          smallMaze
***     solution score:         491
***     solution path:          East East South South West South South West West South
 West West West West West West West West West

### Question q4: 10/10 ###
```

• Q5 – Eating All the Food (2%)

In this question, the challenge was to develop a logical strategy for Pacman to eat all the food in the maze using propositional logic.

Initial Setup:

Food State: At the beginning (time=0), initialize the state of each food item. For each food location (x, y), create a propositional variable indicating the presence of food. These variables are set to true initially.

Incremental Search:

For each timestep t up to a maximum (e.g., 50 timesteps), perform the following:

Location Constraint: Ensure that Pacman can only be at one non-wall location at each timestep. This constraint is managed by the exactlyOne function applied to all possible coordinates where Pacman can legally move.

Eat Action: If Pacman is at a coordinate with food, the state of that food item should be changed to false (indicating it has been eaten). This is modeled by conditional expressions that tie Pacman's presence at a food location to the disappearance of the food from that location.

Model Checking:

Completion Check: After setting the conditions for each timestep, query the model to check if it satisfies the condition where all food items are marked as false (eaten). If such a model is found, it indicates that a viable sequence of movements for Pacman exists that results in all food being eaten within the given timesteps.

Extract Actions: If a satisfying model is found, extract the sequence of actions from this model that leads to the successful completion of the goal (eating all food).

Action Constraints:

Movement Constraints: At each timestep, enforce that Pacman takes exactly one of the possible directions (North, South, East, West). This is implemented using the exactlyOne function on the set of actions, ensuring that Pacman's movement is logically consistent and follows one direction per timestep.

No Solution Scenario:

If no satisfying model is found within the predetermined number of timesteps, conclude that it is not possible to eat all the food within those limits and return None.

```
Question q5
===========

[LogicAgent] using problem type FoodPlanningProblem
Path found with total cost of 8 in 0.1 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases/q5/foodLogicPlan1.test
***     pacman layout:          testSearch
***     solution score:         513
***     solution path:          West East East South South West West East
[LogicAgent] using problem type FoodPlanningProblem
Path found with total cost of 28 in 3.7 seconds
Nodes expanded: 0
Pacman emerges victorious! Score: 573
Average Score: 573.0
Scores:        573.0
Win Rate:      1/1 (1.00)
Record:        Win
*** PASS: test_cases/q5/foodLogicPlan2.test
***     pacman layout:          tinySearch
***     solution score:         573
***     solution path:          South South West East East East East North North North
 North West West West West West East East East South South West West West South S
outh West

### Question q5: 10/10 ###


Finished at 15:12:31

Provisional grades
==================
Question q1: 10/10
Question q2: 10/10
Question q3: 10/10
Question q4: 10/10
Question q5: 10/10
------------------
Total: 50/50
```

Q1(8%). According to AI Weekly in the lecture, some experts and scholars such as Karl Friston and Yann LeCun believe: "You can't get to AGI with LLMs .” Nowadays, the prospects of LLM are so optimistic. Why do you think these experts have such ideas? Please elaborate on your views.

The skepticism expressed by experts like Karl Friston and Yann LeCun regarding the capacity of Large Language Models to achieve Artificial General Intelligence is rooted in several foundational differences between the capabilities of current LLMs and the broader requirements for AGI. Here’s a breakdown of the key reasons for their perspective, along with my views on the matter:

1.LLMs are specialized in processing and generating text based on patterns and data they were trained on. They excel in tasks that involve language understanding, translation, answering questions, and similar text-based processing. However, AGI encompasses a much broader spectrum of cognitive abilities, including reasoning, problem-solving, perception, and manipulation of the physical environment—capabilities that go far beyond text manipulation.

2. AGI implies a level of understanding and potentially consciousness that LLMs do not possess. LLMs operate through statistical inference and pattern recognition, lacking an actual understanding of content in the way humans do. They do not have awareness or conscious experiences and do not genuinely 'understand' the text they generate or process but rather mimic a form of understanding by statistically predicting word sequences. AGI, in contrast, would require a genuine comprehension of the world, possibly akin to human consciousness or awareness.

3. AGI is expected to learn from new experiences and adapt to new environments in real-time, a trait that mirrors human cognitive flexibility. LLMs, however, are typically not designed for such on-the-fly learning and often require extensive retraining with new data to adapt to new circumstances. Their learning is mostly static and confined to the data they were trained on during their initial training phase.

4. AGI should ideally be able to operate across various domains and tasks without needing task-specific data to perform effectively. Current LLMs, however, often require fine-tuning for specific tasks and are limited by the scope of their training data. Their performance can degrade when introduced to scenarios or contexts that are too far removed from their training environments.

5. The development of AGI would also necessitate the integration of ethical reasoning and social norms, areas where LLMs currently do not operate independently. They can mimic ethical reasoning based on their training data but do not possess an intrinsic sense of ethics or morality.

I partially agree with these experts. For example, in my experience with building a Large Language Model (LLM), researchers often exclude offensive language to prevent the model

from generating inappropriate responses. This situation illustrates the fifth perspective mentioned earlier about ethical and societal understanding. It's challenging to find a way for the model to comprehend the context and implications of such language soon because it's difficult to quantify these nuances. While LLMs may not be capable of achieving Artificial General Intelligence (AGI) on their own, they undoubtedly contribute to the development and implementation of AGI systems.

Q2(8%). According to the paper ”CLIP-Event: Connecting Text and Images with Event Structures,” in CVPR 2022, after the process of generating the event-centric structured data, how does this work implement contrastive learning? Specifically, how does this work choose the positive and negative samples for contrastive learning?

Positive Samples: In the context of this paper, positive samples are pairs of text and images that refer to the same event. The alignment is based on the event structure extracted from both modalities. If the textual description and the image depict the same actors performing the same actions within the same context, they are considered positive samples. These samples help the model learn to recognize and reinforce the connections between similar event descriptions across text and images.
Negative Samples: Negative samples, conversely, consist of pairs where the text and image do not correspond to the same event. This could mean different actors, actions, or contexts. By contrasting these negative examples against the positive ones, the model learns to distinguish between differing events and to accurately align only those text and image pairs that truly depict the same scene.
Learning Process
The training process involves minimizing a contrastive loss, such as a triplet loss or a noise-contrastive estimation loss, which encourages the model to minimize the distance between embeddings of positive pairs while maximizing the distance between embeddings of negative pairs. Through iterative training on batches of positive and negative samples, the model fine-tunes its parameters to better align text-image pairs based on their underlying event structures.

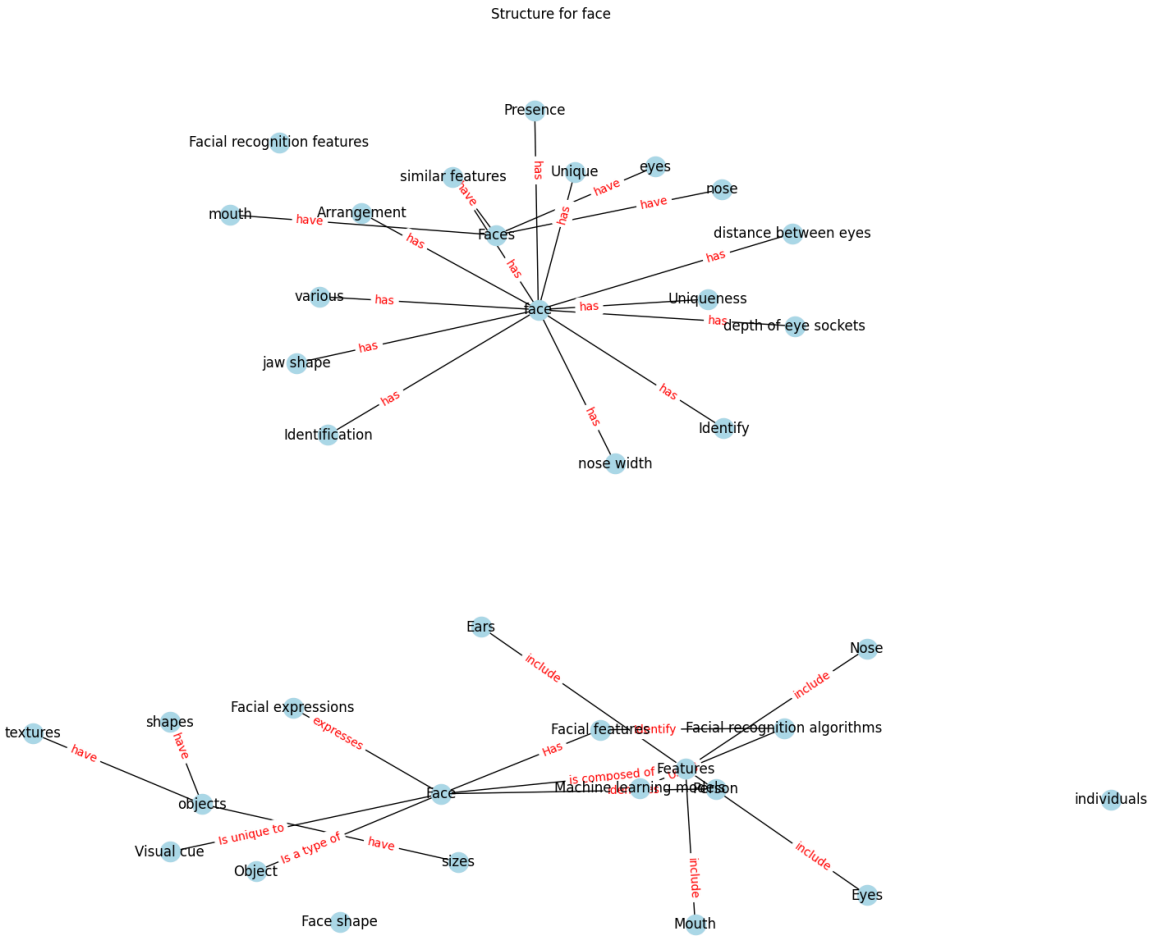
Q3(8%). Referring to the paper, what is the main problem with the current description of the VLM pre-training process? Please describe the steps to generate the structured graph-based data in this work.

The main problem with the current description of the Vision-Language Model (VLM) pre-training process is the lack of structured knowledge that effectively represents the interconnections among entities or attributes linked to a particular category. Conventional descriptions tend to be unstructured and do not capture the nuanced relationships and attributes that are essential for creating robust models capable of understanding and interacting with visual and textual data effectively. This limitation can lead to underwhelming performance, especially in scenarios where the category names are ambiguous or where the model needs to understand complex relationships within the data.

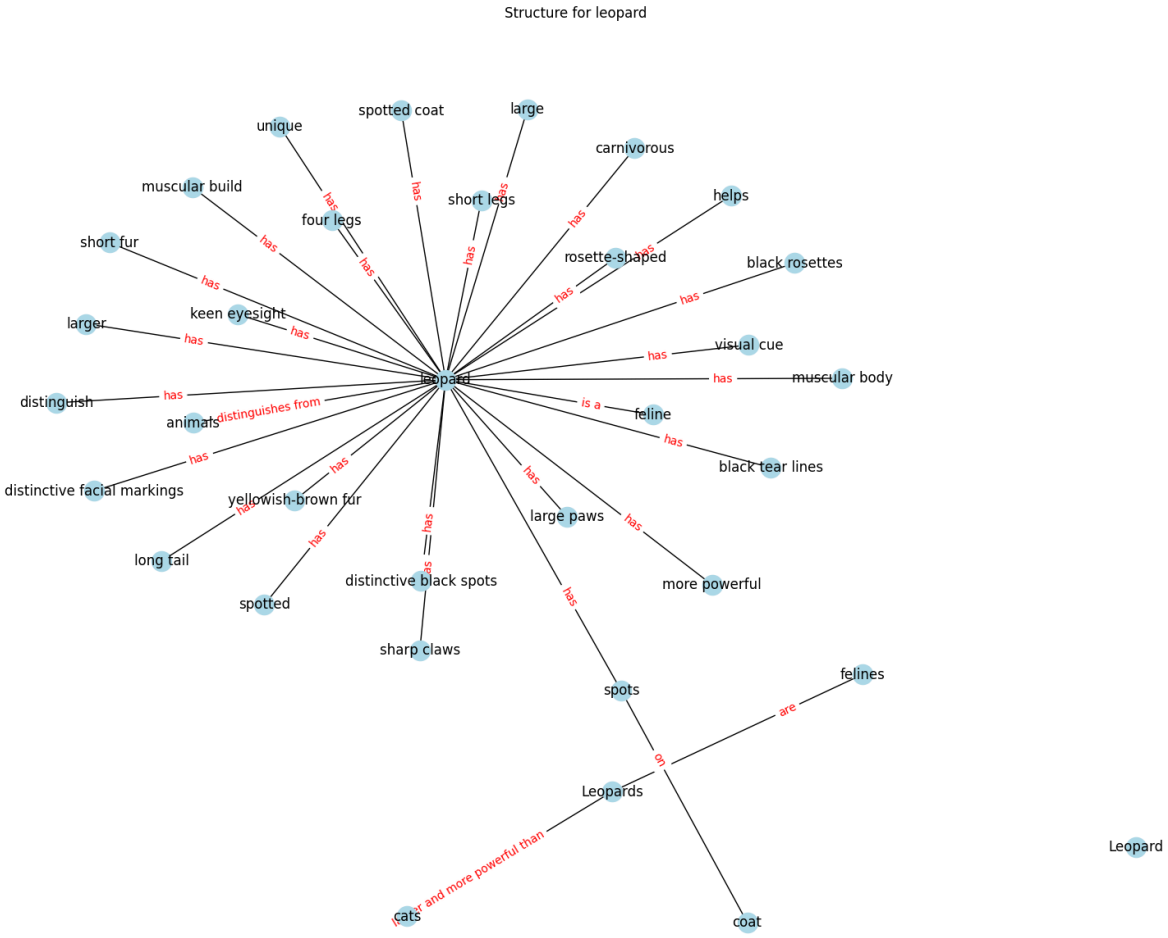
Steps to Generate Structured Graph-based Data:

1. Hand-crafted instructions are input into a Large Language Model (LLM) to generate human-like category-related descriptions. These descriptions are detailed and specific, encompassing entities and attributes that define the category.
2. These generated descriptions are then reconstructed into a structured format. This is done by dividing the content into four main parts:
   Entities: Identifying all the entities mentioned in the description.
   Attributes: Delineating all the attributes associated with these entities.
   Relationships between Entities: Establishing the connections or relationships that exist between different entities.
   Relationships between Entity and Attribute: Linking each attribute to its corresponding entity to create a clear mapping of how attributes modify or describe entities.
3. Using the structured data from the previous step, a graph is constructed for each description. This graph models the entities as nodes and the relationships (both inter-entity and entity-attribute) as edges. This graph-based representation captures the structured knowledge in a visual and interconnected format, which is more organized and facilitates the discovery of implicit connections that may not be evident in the original text descriptions.

Q4(8%). Please select one of the datasets provided in this work and visualize two categories.
• Path to the gpt-generated data: ./data/gpt_data
• You must show the corresponding description and two graph-based structured data components for each category as shown on the right.



Description for face: ['A face is a unique arrangement of features that includes eyes, nose, mouth, ears and facial expressions that identifies a person.', 'Facial recognition features include distance between eyes, nose width, jaw shape, and depth of eye sockets. These are unique to individuals.', 'Facial recognition algorithms identify unique facial features, such as eyes, nose, mouth, and face shape, using machine learning models.', 'Faces have similar features such as eyes, nose, and mouth, while other objects can have various shapes, sizes, and textures.', 'The visual cue that is unique to face among all objects is the presence of facial features such as eyes, nose, and mouth.']



Description for leopard: ['A leopard is a carnivorous, spotted feline with a muscular body, four legs, short fur, sharp claws, and keen eyesight.', 'The distinct features of leopard are its spotted coat, muscular body, large paws, and distinctive facial markings including its black "tear" lines.', 'Leopards are large felines with yellowish-brown fur covered in black rosettes, a long tail, and short legs.', 'Leopards have distinctive black spots on their fur, a muscular build, and a long tail. They are larger and more powerful than most cats.', "The rosette-shaped spots on a leopard's coat are a unique visual cue that helps to distinguish them from other animals."]

Q5(8%). Based on current VLM auxiliary data improvement methods, such as the event-centric structure data in the lecture and the structured linguistic knowledge in this paper, what other deep semantic knowledge do you think humans possess that can be provided to VLM for learning?

Integrating fundamental principles of physics, specifically gravity, can significantly enhance a VLM's understanding of the world. Teaching VLMs about gravity involves several structured steps:

- Start by defining gravity in the training data, explaining it conceptually as the force that attracts objects toward each other, which causes objects on Earth to fall towards the ground.
- Utilize simulations to generate visual data showing various scenarios influenced by gravity, such as objects falling, rolling, or being thrown. These visuals should be annotated with descriptions that clearly connect the depicted actions to the effects of gravity.
- Design specific tasks for the VLM that involve predicting the outcomes of physical interactions under gravity's influence. These tasks can include predicting object trajectories or outcomes of collisions, enhancing the VLM's predictive and interpretative capabilities in physics-based scenarios.
- Along with qualitative descriptions, introduce the quantitative aspects of gravity, including the gravitational constant and related physics equations. This will enable the VLM to engage in problem-solving that requires numerical calculations, such as determining the time it takes for an object to hit the ground from a given height.
- To deepen understanding, use contrastive learning approaches by presenting scenarios with different forces, such as electromagnetic forces or conditions of microgravity. This helps the VLM distinguish the specific effects of gravity from other forces.
- Apply the trained model to analyze real-world images and videos to identify and describe gravitational effects accurately, continuously refining the model's performance based on real-world data and feedback.

Integrating this deep semantic knowledge of physical principles like gravity would enable VLMs not only to better understand and predict physical phenomena but also to contextualize and reason about everyday scenes and events more effectively. This approach aims to enrich the VLM's toolkit by grounding it in fundamental scientific knowledge, thereby expanding its utility across various applications that require an understanding of real-world dynamics.