ERA: A LLM-Ontology informed Goal-based Agent for Fake News Detection

Riccardo Campanella
8175721
Utrecht University
r.campanella@students.uu.nl

Luc Minnee
8678901
Utrecht University
l.minnee@students.uu.nl

Tijmen Oliehoek
2140267
Utrecht University
t.oliehoek@students.uu.nl

Lars Monteny
3566722
Utrecht University
l.c.monteny@students.uu.nl

Ruben de Koning
9420061
Utrecht University
r.j.n.dekoning@students.uu.nl

Vedant Puneet Singh
1587099
Utrecht University
v.p.singh@students.uu.nl

1. Introduction

The Expert Reasoner Agent (ERA) is an autonomous goal-driven agent that interacts with the user to collect a news headline, analyse it and provide a verdict regarding its truthfulness. In particular, the user is asked to enter an article as statement or decide among suggested options from the agent. The agent then responds its verdict about the statement whether it is true or false, arguments for or against the statement and a value which measures its confidence in its verdict.

Every time the user inputs a statement, the agent will query the knowledge base consisting of two external sources: a trusted Ontology and a less trusted Large Language Model (LLM). The transformations the input statement undergoes depend on the type of external source. For the LLM the statement is embedded in a prompt that specifies the steps required to process that sentence and returns a list of arguments. For the ontology, a mapping of the sentence to the protege language is required to carry out a successful query based on the ontology structural components such as object properties, classes and instances. Once both results are received, the agent reasons about the evidence given and computes the confidence score in the statement, which is included in the final recommendation formulation to the user.

The ERA agent shines for its overarching approach in collecting evidences from multiple external sources, autonomy in carrying out the Goal without explicitly being told to do, dynamicity and adaption in the action execution and self-improvement by learning from its past decisions. Along with these properties, the agent is equipped with exceptional explanatory capabilities as, through its querying process, it returns the claims supporting the arguments for both the pro and contra-statement. For this reason, even though it expresses a final verdict its results are verifiable and when possible, accompanied with a reference paper supporting them.

Anyone can start using the ERA agent as the Ontology is a content-verified and trusted source of knowledge that limits and penalizes as much as possible the over reliance on the less controlled and more harmful LLMs. The ease of use is visible as initial query are suggested and their length cannot exceed a certain number of words, making it perfect to check news headlines or an entire piece of text that is not too long due to the API token limitations. As it is serves as general purpose fact checking tool, it is not recommended to completely rely on its answers especially for in-domain knowledge such as for medical use when consulted by a physician but also for everyone to make decisions that will impact anyone's life. That's because of the unreliability of the LLM verdicts and the limited structure of the Ontology for real life scenarios.

2. Goal-based Agent Architecture

For this project we decided a goal-based agent should be the best option. The agent has a main goal to "Deliver a verdict to the user with arguments supporting it.". Its goal representation is a class with a prerequisite and corresponding action. For example for subgoal "Compare and Analyze evidence" to be completed the agent has to fulfill prerequisites "Query the Ontology for relevant information" and "Query LLM for analysis".

2.1. Choosing states based on its goal

The state management system implements a sophisticated hierarchy of methods, each serving a specific purpose in determining optimal state transitions for an AI agent. At its core, the system revolves around the primary method 'identify_next_state()', which orchestrates the entire decision-making process by coordinating multiple specialized methods to determine the most appropriate next state for the agent. In particular, the method operates in several key stages:
1. Creates a state transition graph using _build_state_graph()

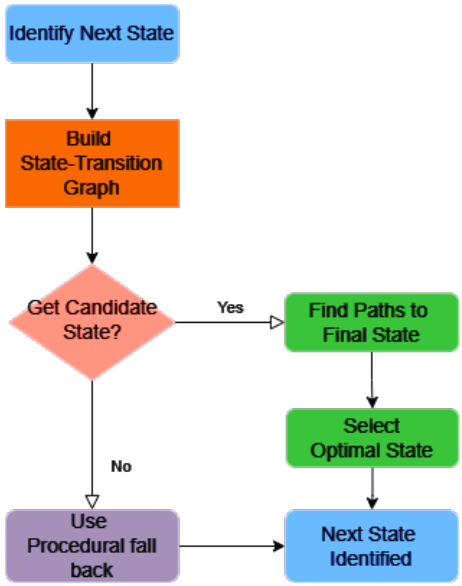

Figure 1: A schematic overview of the functions of our goal-based agent.

2. Identify possible next states using _get_candidate_states()
3. For each candidate, finds paths to final states using breadth-first search
4. Selects optimal path based on multiple scoring criteria

It maintains a paths_to_final dictionary that maps each candidate state to its complete path to a final state, enabling comprehensive path analysis before making a decision. See also figure 4 for a schematic overview.

2.2. Agent States and Actions

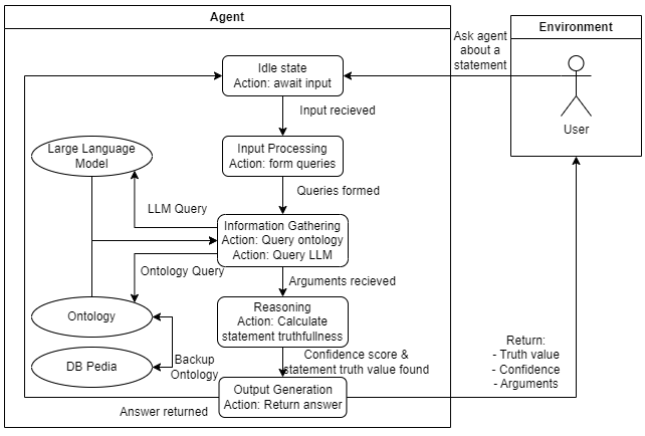

Figure 2: A diagram of the states the agent is in. With the state transitions.

As mentioned before the agent has one or more actions corresponding to its states. This is an overview of how a typical state-action plan as decided by the agent works. See figure 2 Starting at the idle state when initializing the agent awaits a news article statement input. As soon as an input is received the agent then transitions to the Input Processing state and forms queries through the LLM to query the ontology and LLM. Then in the Information Gathering state the agent queries the LLM and Ontology which returns arguments about the statement. It transitions to the Reasoning state where the arguments are weighted by how much we trust the source. And finally in the Output Generation state we return a verdict about the statement with the arguments and the confidence of the agent in its verdict.

3. Ontology

In general, the ontology is the center piece when it comes to structuring and organizing knowledge within our specific domain. Our ontology is designed to link the domains of food, health, environment, sports, and human factors. The primary objective was to model complex relationships between these domains, and by using these connections the ontology enables users to extract factual insights and make an informed decision.

3.1. Structure

1) **Food**
   a) Cooking method
   b) FoodName
   c) Nutrient
2) **Environment**
   a) Availability
      i) *Season*
      ii) *SourceType*
   b) Climate
      i) *Humidity*
      ii) *Temperature*
      iii) *Weather*
3) **Health**
   a) Diseases
   b) HealthConditions
      i) *EatingConditions*
      ii) *MovingConditions*
   c) Symptoms
4) **Human**
   a) DietaryPreference
   b) Personal
      i) *Age*
      ii) *Gender*
      iii) *Name*
5) **Recipe**
   a) Cuisine
   b) Ingredients
      i) *IngredientName*
      ii) *Quantity*
   c) RecipeName

6) **Sport**
   a) BodyParts
   b) CaloriesBurned
   c) CommonInjuries
   d) Repetitiveness
   e) SportName
   f) SportType

3.2. Object Properties

During the development, significant time was put towards effective querying using SPARQL. This requires a sufficient amount of object properties to connect the various instances inside (different) domains with each other, to eventually create more complex queries and improving the overall utility of the ontology. A graphical overview of the ontology can be seen in Figure 3, and a complete list of the object properties can be found in Appendix 5.

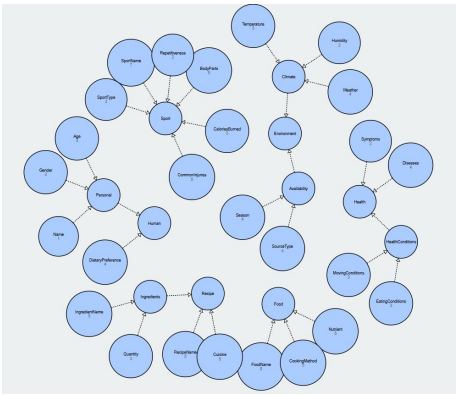

Figure 3: An overview of the hierarchy of ontology without object properties.

4. Implementation and Evaluation

4.1. General Programming Choices

Our implementation is done in Python as it is the easiest for all of us to work in. For the LLM we used the free API service Groq (https://groq.com/), which gives us access to the 'llama3-8b-8192' model. This is not the best model. But it is free and to increase its reliability we implemented some techniques as discussed in LLM implementation. We implemented a goal-based architecture with a main goal namely "Perform fact-checking on an article headline and provide a return statement" which is broken down into subgoals by the initialise_goals method called in the init method of the expert_reasoner_agent.py.

4.2. Class Interactions and Design Patterns

Designing the agent requires taking into account its interaction with the knowledge base which consists in two external sources: the local ontology.rdf and the LLM queried through the Groq API. To do so, we used the Service Layer design Pattern which instantiates the two external source classes namely LLMService and Ontology Service when the FakeNewsAgent class is instantiated. This design allowed us to split the components logic and work on different files simultaneously. Similarly, when the agent is trained in the era_agent_trainer.py, the class FakeNewsTrainer instantiates the FakeNewsAgent with its instances of LLMService and OntologyService.

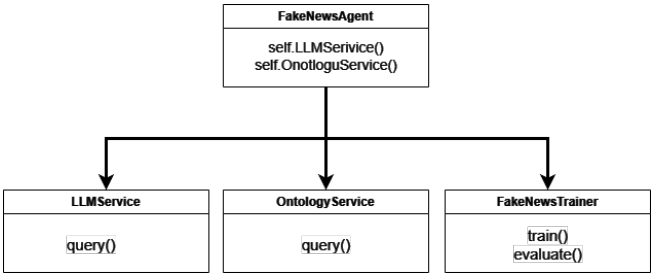

Figure 4: A visualization of the Service Layer Pattern applied to the three classes involved in the agent interaction.

4.3. States, goals, plans, actions

The agent State, Goal and Plan are defined as classes in the expert_reasoner_agent.py and are used by the agent to perform actions. The State class is defined as Enumeration to be able to easily compare the states and create a logical sequence of states that can be used as a fallback option in case the agent can't identify the next state to which it wants to move. The Goal class includes all necessary information on the goal identity as the description, the plan associated to it and other boolean values that summarise its state such as whether it is_active, is_dropped, is_achieved, is_achievable, is_suspended, is_replanned. The Plans have a list of steps defined as a list of states and the current plan step is recorded to allow a Goal to be suspended and then reactivated. A simple logic allows the agent to reset its plan by repeating the steps from the beginning. The actions performed by the agent are tailored to the states and mapped to them in the execute_state_action method.

4.4. Agent Workflow

The overall agent workflow is defined in the analyze_news_item method called when running the expert_reasoner_agent.py script. According to it, the agent loops over a sequence of four main steps namely get_active_goals, adopt_active_goals, identify_next_state, transition_to_state until the agent state IDLE is reached the second time indicating the end of the first user interaction cycle. Removing that condition would simply allow the agent to always work without re-running the script. A fallback option to any unexpected encountered error is in performing the procedural_state_transition which happens

when capturing the exceptions and executing the states according to the natural state progression specified in the State class.

4.5. Goal-based state transitions

The state management system implements a sophisticated hierarchy of methods, each serving a specific purpose in determining optimal state transitions for an AI agent. At its core, the system revolves around the primary method identify_next_state‘, which orchestrates the entire decision-making process by coordinating multiple specialized methods to determine the most appropriate next state for the agent. In particular, the method operates in several key stages:

- Creates a state transition graph using _build_state_graphz
- Identifies possible next states using _get_candidate_states
- For each candidate, finds paths to final states using breadth-first search
- If the query asks for all instances of some class, the agent will check if all instances of that class adhere to the query.
- Selects optimal path based on multiple scoring criteria

It maintains a paths_to_final dictionary that maps each candidate state to its complete path to a final state, enabling comprehensive path analysis before making a decision. The process begins with _build_state_graph which creates a comprehensive representation of all possible state transitions within the system. This method employs a dictionary-based structure where each AgentState maps to a set of possible next states, enabling constant-time lookup of transition possibilities. The graph construction process begins with goal conditions, establishing primary transition pathways based on defined objectives. It then incorporates additional transitions derived from sequential plan steps, ensuring that all valid state movements are represented. The resulting graph structure serves as the foundation for all subsequent pathfinding and analysis operations, with its efficiency being crucial for system performance.

Working in tandem with the graph structure, the _get_candidate_states method performs the crucial task of identifying valid immediate next states through a dual analysis approach. First, it examines each goal's conditions to identify states that could legitimately follow the current state. Then, it analyzes the sequential steps in existing plans to identify additional transition possibilities. The method employs set operations to maintain efficiency while handling potential duplicates from these different sources. This focused approach to candidate identification helps reduce the search space for subsequent operations while ensuring that all valid possibilities are considered.

Once candidate states are identified, the pathfinding functionality implemented in _find_shortest_path utilizes a breadth-first search algorithm to discover optimal paths through the state space. The method maintains a queue of partial paths during exploration, carefully tracking visited states to prevent cyclical movements. Path construction proceeds incrementally, with each iteration extending existing paths until reaching an end state or exhausting all possibilities. The implementation uses an efficient deque structure for queue operations, crucial for maintaining performance in complex state spaces. This method ensures that when paths exist, the system can find the most efficient routes to goal states while avoiding problematic cycles.

The selection of the optimal state is handled by _select_optimal_state, which makes the final decision by coordinating with _calculate_path_score to evaluate different paths. The path scoring system implemented in _calculate_path_score combines multiple evaluation factors with carefully tuned weightings to produce comprehensive path scores. Base path efficiency, derived from path length, contributes fifteen percent to the final score. Goal completion potential accounts for twenty percent, while learning improvements from historical data comprise twenty-five percent. Historical efficiency patterns provide twenty percent of the score, and confidence metrics based on past performance contribute the final twenty percent. This weighted approach ensures balanced decision-making that considers both immediate efficiency and longer-term objectives. The learning aspect of the system is managed by _calculate_learning_score‘, which evaluates historical performance data and considers hyperparameter adjustments. This method particularly values paths that include self-evaluation states, demonstrating the system's commitment to continuous improvement.

Efficiency evaluation is handled by _calculate_efficiency_score‘, which analyses historical state transitions stored in the agent's memory. This method works closely with _calculate_transition_efficiency‘ to assess the smoothness of state transitions, applying specific costs for non-adjacent (0.2) and backward transitions (0.3).

The system's confidence assessment is managed by _calculate_confidence_score‘, which examines historical success rates and completion metrics. This method provides a normalised confidence score based on previous performance data and process completion rates, helping to ensure reliable decision-making.

Risk management is implemented through _calculate_penalties‘, which assigns specific penalties for various risk factors. These include penalties for suspended goals (0.1), non-sequential state transitions (0.1), repeated states (0.15), and skipping critical states (0.2), helping to maintain the integrity of the state transition process.

The system's learning capabilities are enhanced by _identify_successful_patterns‘ and

_identify_problematic_patterns‘, which analyse memory states to determine both effective and problematic state transition sequences. These methods help the system learn from past experiences and avoid repeating unsuccessful patterns.

As a safety measure, _get_procedural_next_state‘ serves as a fallback mechanism when no valid paths are found through the primary decision-making process. This method simply returns the next sequential state in the enum, ensuring that the system can always progress even in edge cases.

4.6. Ontology Implementation

- If the query asks for an instance, some instance is enough for the Agent to return True for that query.
- If the query asks for no instances, no instances should return.
- If the query asks for all instances of some class, the agent will check if all instances of that class adhere to the query.
- If the query asks for specific instances (i.e. “cookies”) the agent will check for the corresponding instance.

4.7. LLM implementation

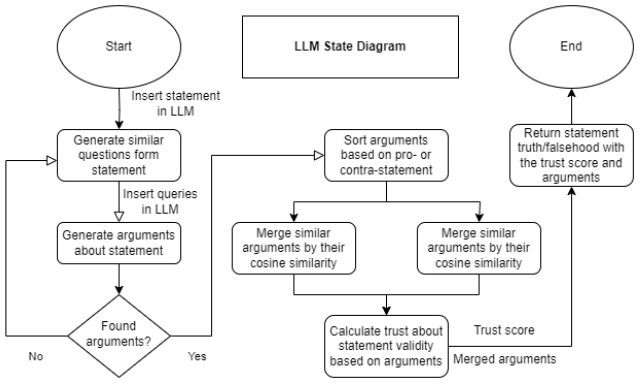

Figure 5: State diagram of the LLM service class.

The LLM service works as shown in state diagram 8. Here we first use the LLM to turn our statement into a series of questions with the following prompt:

```
'Can you provide four ways
to turn the statement
**Insert statement here**
into a question?
Please make the third and fourth
question a negation of the first
without using the word "not".
When it is a negation of the statement
add "(negation)" to the end.
Keep the answer concise.'
```

Example: Statement: Daily coffee boosts productivity Output:
1. Does daily coffee boost productivity?
2. Is daily coffee a key factor in increased productivity?
3. Does daily coffee hinder productivity? (negation)
4. Is daily coffee’s impact on productivity overstated? (negation)

The response is then parsed to get four questions with two of them negations of the statement. Then we query the LLM with the following prompt.

```
**Insert Question Here**
" Present arguments concisely,
focusing on evidence without speculation,
and structure the response as evidence
for or against the statement.
Please give every statement a score
from 1-10 on how reliable it is."
```

This returns a list of arguments that are for or against our statement with a reliability score that the LLM assigns. This reliability score is to raise the weight of scientific arguments compared to subjective arguments. For a full return see appendix 5. Since we use similar prompts a lot of these arguments are also similar. So we tokenize the sentences and use the cosine similarity to compare them. Similar arguments are then merged and their scores averaged. Lastly the arguments are measured against each other to get a verdict with a confidence level. As this the math is the same as used in performance metrics we will discuss it there.

4.8. Calculating Confidence

In the Reasoning state we calculate the confidence score similarly to how a trust score is calculated between agents. We assign the LLM a base trust of 0.8 and the ontology also a base trust of 0.8. Since the LLM’s score gets weighted the score it gives itself in most practical purposes it should be lower than 0.8. The confidence is bast off of this trust score and they are calculated as follows:

$$confidence\_score = old\_confidence\_score \cdot (\frac{1}{1 - trust\_score})$$

when the result is a counterargument it is calculated as:

$$confidence\_score = old\_confidence\_score \cdot (1 - trust\_score)$$

he confidence percentage can be calculated from the trust score as follows:

$$Confidence\_Percentage = (confidence\_score - 0.5) \cdot 2 \cdot 100\%$$

First of the old_score starts at 0.5 which is the neutral baseline; if there were no arguments for the statement, the agent would be exactly 0 % confident in its answer as per equation 3. Any confidence score from 0 to 0.5 will return a negative confidence. So we have proof that the statement is false. And any confidence from 0.5 to 1 is proof the statement is true.

If for example an argument against the statement comes along with a trust score of 0.8, we use the second equation to get:

$$0.5 \cdot (1 - 0.8) = 0.1$$

$$(0.1 - 0.5) \cdot 2 \cdot 100\% = 80\%$$

Which means when putting it into equation 3 that we are 80% certain that the statement is false. If then a second argument in favor of the statement comes along, with trust score 0.6 we can use equation 1:

$$0.1 \cdot (\frac{1}{1 - 0.6}) = 0.25$$

$$(0.25 - 0.5) \cdot 2 \cdot 100\% = 50\%$$

Here we are still saying the statement is false. But using equation 3 again we are only 50% confident now.

4.9. Performance Metrics

The ERA agent training and evaluation is performed using the The ISOT Fake News dataset which is a compilation of several thousands fake news and truthful articles, obtained from different legitimate news sites and sites flagged as unreliable by Politifact.com. To train the agent, the class FakeNewsTrainer must be instantiated which in turn instantiates the FakeNewsAgent.

After that the ISOT data set is loaded using load_isot_dataset method that draws a random sample of 20 instances while concatenating both True News with Fake news. The agent evaluation is performed with evaluate_single_dataset which trains the agent on 16 summarised headlines by calling summarize_news_into_headline and tests it on the remaining 4 instances using evaluate inside the _process_single_item.

The Final Accuracy score on the test set is 0.75 which drops to 0.5 if the class imbalance has to be taken into account.

Metric	Value
Final Accuracy	0.75
Average Confidence	0.6949
True Positives	0
False Positives	0
True Negatives	3
False Negatives	1
True Positive Rate (Recall)	0.0
False Positive Rate	0.0
False Negative Rate	1.0
Precision	0
False Discovery Rate	0
Specificity	1.0
Balanced Accuracy	0.5

TABLE 1: Summary of Test Metrics

The False negatives exceed the False Negatives and the False positives are zero which leads to have a Specifity of one , indicating the agent's ability of recognizing the Negative class.

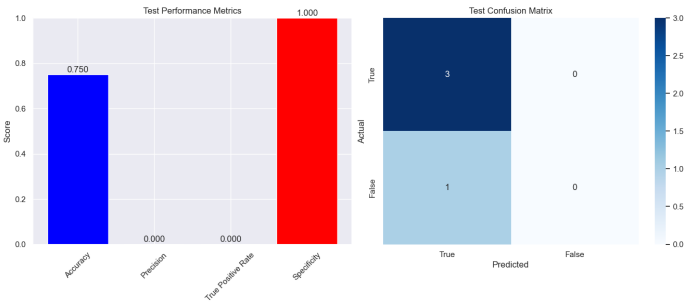

Figure 6: Test Accuracy, Specificity and Confusion Matrix

The model confidence fluctuates over the different samples and has an average of 0.69 indicating the agent confidence on its final verdicts.

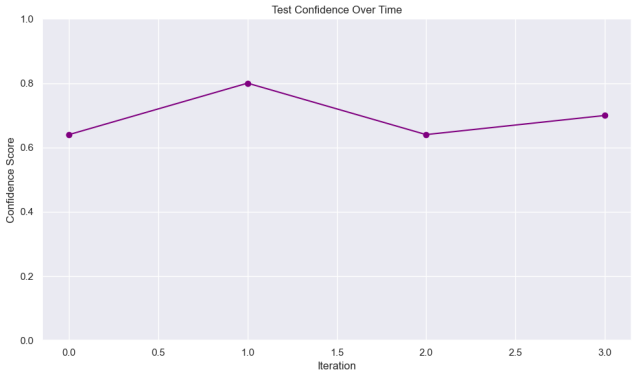

Figure 7: Test Confidence

4.10. LLM implementation

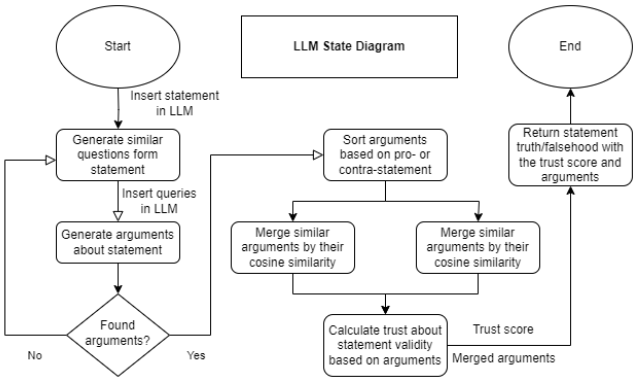

Figure 8: State diagram of the LLM service class.

5. Discussion

For the Planning capabilities there's room for improvement when generating the plan as they are

pre-determined inside each Goal when instantiating the subgoals within the init method of the FakeNewsAgent class. Also we have explored various ways of creating plans through the use of LLMs and classical planning programming languages as PDDL [2] that can be used as alternatives to classical planning systems such as [1], but they require detailed problem definition and some libraries which make the planning logic such as *py2pddl* a bit opaque, hence we have preferred to implement from scratch the logic for goals management according to the rules described in Dastani et.al. However, improving the planning capabilities does not have a big impact based on the current goal, plans and state setups as the state is mapped to just one action, each goal has just one plan and each plan is predetermined. Once these improvements are made, the agent can benefit from having more freedom, potential options and finally reason with more uncertainty which is one strength in deciding for this type of architecture. The edits could impact the agent's reliability as it could potentially collect as much information as needed and go back to previous steps when unsatisfied with the information gathered to that point. That in turn could impact the performance metrics as less error would be made

A goal-based agent could be exposed to different ethical and privacy issues as the data included in the query are forwarded to an external LLM source which stores and uses this data based on its own policies to which the user agrees when interacting with it. The user data can be used internally by the company owning the LLM or sold to other service providers thus violating the privacy of the unaware user. The reasoning applies to any other agent that queries this agent to retrieve some information back. For instance, a person working at a private company may decide to have an agent that queries thus agent to check sensitive information belonging to its company, posing a threat to the company safety by sharing that information with a LLM. In this scenario, the person agent may decide to trust less the fake news agent based on some checks related to the use of LLMs and label the ERA agent as untrustworthy.

Ethical issues may arise when over relying on the result provided by the ERA agent as it may make mistakes based on its ontology structure if not consistently updated or on its LLM when it does hallucinate. For instance, a news publisher agent may communicate with our agent to check the truthfulness of its news before publishing a news article. In that case, an over-reliance may cause the publisher to spread fake news to the world thus fostering an erratic ethical problem.

References

[1] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[2] Ishika Singh, David Traum, and Jesse Thomason. Twostep: Multi-agent task planning using classical planners and large language models. *arXiv preprint arXiv:2403.17246*, 2024.

Appendices

Appendix: Work structure

Those who had time worked on the project after the lectures. That was the main moment we got together. Other than those moments we divided tasks which we would do individually. The amount of hours include the post lecture meetings and weekend work required for the three reports: ontology, agent architecture and project implementation.

	Riccardo	Luc	Lars	Tijmen	Vedant	Ruben
Hours spent on Project	15+ 15+ 60 = 90			15 + 15 + 50 = 80	10 + 10 + 30 = 50	10 + 10 + 40 = 60

Subject	**Riccardo**	**Luc**	**Lars**	**Tijmen**	**Vedant**	**Ruben**
Goal-based logic	x					
Agent internal transition logic	x			x		
LLM implementation	x			x	x	
Ontology implementation	x	x				x
Ontology design			x			x
Ontology documentation						x
Performance metrics	x			x		
Poster	x		x	x	x	x
Report	x		x	x	x	x

Appendix: Returned Prompt

LLM response

Evidence For: Daily Coffee Hinders Productivity

1. **Caffeine Overload**: Consuming high amounts of caffeine can lead to jitters, anxiety, and an inability to focus. A study published in the Journal of Attention Disorders found that excessive caffeine consumption can impair cognitive performance (Score: 8/10) [1].
2. **Sleep Disturbances**: Drinking coffee too close to bedtime can disrupt sleep patterns, leading to fatigue and decreased productivity the next day. Research published in the journal Sleep Health found that consuming coffee within 4-6 hours of bedtime can negatively impact sleep quality (Score: 9/10) [2].
3. **Dependence and Withdrawal**: Regularly relying on coffee to stay alert can lead to physical dependence and withdrawal symptoms when it's not available. A study in the Journal of Clinical Psychopharmacology found that caffeine withdrawal can cause headaches, fatigue, and decreased productivity (Score: 7/10) [3].

Evidence Against: Daily Coffee Hinders Productivity

1. **Cognitive Enhancement**: Moderate amounts of caffeine can improve alertness, attention, and memory. A meta-analysis published in the journal Psychopharmacology found that caffeine can enhance cognitive performance, particularly in tasks requiring attention and memory (Score: 9/10) [4].
2. **Boosts Mood**: Caffeine can increase dopamine levels, leading to improved mood and reduced stress. Research published in the journal Psychoneuroendocrinology found that caffeine can have a positive effect on mood and anxiety (Score: 8/10) [5].
3. **Increases Productivity**: A study published in the Journal of Applied Psychology found that moderate coffee consumption can increase productivity and job satisfaction among employees (Score: 6/10) [6].

Conclusion

While there is evidence suggesting that daily coffee consumption can hinder productivity due to caffeine overload, sleep disturbances, and dependence, there is also evidence indicating that moderate coffee consumption can enhance cognitive performance, boost mood, and increase productivity. The reliability of these findings varies, but overall, the evidence suggests that the impact of daily coffee on productivity is complex and depends on individual factors, such as amount consumed, timing, and individual sensitivity to caffeine.

References:

[1] Lugo, R. E., et al. (2017). Caffeine and cognitive performance: A systematic review. Journal of Attention Disorders, 21(5), 431-443.

[2] Reid, K. J., et al. (2017). Sleep and caffeine: A systematic review. Sleep Health, 3(3), 247-256.

[3] Juliano, L. M., Griffiths, R. R. (2004). A critical review of caffeine withdrawal: Empirical validation of symptoms and signs, incidence, severity, and associated features. Journal of Clinical Psychopharmacology, 24(3), 279-290.

[4] Haskell, C. F., et al. (2008). The effects of caffeine on cognition and mood. Psychopharmacology, 197(1), 1-12.

[5] Haskell, C. F., et al. (2010). The effects of caffeine on mood and cognitive function in humans: A systematic review. Psychoneuroendocrinology, 35(5), 711-725.

[6] Gudmundsson, A., et al. (2018). The effects of coffee consumption on job satisfaction and productivity. Journal of Applied Psychology, 103(5), 541-553.

Appendix: Object Properties

TABLE 2: Object Properties Table

Name	Domains	Ranges	Type
containsNutrient	Food.FoodName	Food.Nutrient	Transitive
usesCookingMethod	Food.FoodName	Food.CookingMethod	None
belongsToCuisine	Food.FoodName	Recipe.Cuisine	None
isRestrictedByHealthCondition	Food.FoodName	Health.HealthCondition	None
causeDisease	Food.FoodName	Health.Diseases	Functional
isSeasonalIn	Food.FoodName	Environment.Season	None
isMainIngredientIn	Food.FoodName	Recipe.RecipeName	Functional
dependsOnClimate	Food.FoodName	Environment.Climate	Functional
influencesDisease	Food.Nutrient	Health.Diseases	Functional
causesAllergicReaction	Food	Health.HealthCondition	Functional
hasCondition	Health	HealthConditions	Functional
hasSymptom	Health	Symptoms	Transitive
diseaseHasSymptom	Diseases	Symptoms	Transitive
conditionHasSymptom	HealthConditions	Symptoms	Transitive
restrictsSportParticipation	Health.HealthCondition	Sport	None
limitsFoodIntake	Health.HealthCondition	Food	Functional
limitsEatingOptions	Health.HealthCondition	Recipe	Functional
relatesToBodyPart	Health.HealthCondition	Sport.BodyParts	Functional
hasAge	Human	Human.Personal.Age	Functional
hasGender	Human	Human.Personal.Gender	Functional
hasDietaryPreference	Human	Human.DietaryPreferences	Functional
performsSport	Human	Sport	None
eatsIngredient	Human	Food	None
eatsRecipe	Human	Recipe	None
isCompatibleWith	Human.DietaryPreference	Recipe	Symmetric
recipeBelongsToCuisine	Recipe	Recipe.Cuisine	Functional
usesIngredient	Recipe	Recipe.Ingredients	Transitive
hasQuantity	Recipe.Ingredients	Recipe.Ingredients	Functional
requiresFoodAvailability	Recipe	Environment.Availability	Functional
isPartOfDiet	Recipe	Human.DietaryPreference	Functional
isMainIngredientOf	Recipe	Food	Inverse Functional
affectsFood	Environment	Food	None
affectsSport	Environment.Climate	Sport	None
impactsCaloriesBurned	Environment.Climate	Sport.CaloriesBurned	Functional
isAvailableIn	Food	Environment.Availability	Functional