UNIVERSITY OF TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Artificial Intelligence Systems

Final Dissertation

Exploring the Use of LLMs for Agent Planning: Strengths and Weaknesses

Supervisor
Paolo Giorgini

Student
Davide Modolo
229297

Academic year 2023/2024

Contents

Abstract

1 Introduction

This thesis explores the capabilities of Large Language Models (LLMs) in the context of a logistics problem. Artificial intelligence has made significant strides in generative systems, particularly with the advent of LLMs, which are capable of producing coherent and contextually relevant text based on input prompts. However, their ability to autonomously plan and achieve goals without additional external structures remains a topic of investigation. The primary aim of this work is to assess whether LLMs can be effectively utilized as agents in dynamic environments without leveraging predefined frameworks or knowledge bases.

To achieve this, a thorough analysis of existing methodologies is necessary. Traditional approaches such as PDDL and Reinforcement Learning provide structured and systematic ways to tackle planning problems. PDDL offers explainability and efficiency in constrained environments but lacks adaptability, making it impractical for real-time applications. On the other hand, Reinforcement Learning is highly adaptable and effective in changing environments but suffers from issues such as convergence to local minima and lack of explainability. Recent research has also explored planning capabilities of LLMs, with several studies investigating their reasoning abilities and limitations. The specific problem addressed in this thesis involves evaluating the capacity of an LLM, devoid of external structures, to solve logistics problems in dynamic settings.

[TODO]

2 Background

In this thesis, we will analyze in detail the behavior of an LLM as an agent within a controlled environment.

Before presenting all the work carried out in detail, this chapter aims to provide a comprehensive explanation of all the theoretical foundations necessary to understand the steps presented in the following chapters. Starting from a brief introduction of Artificial Intelligence just to define the boundaries in which we are working, we will move to the core concepts. In particular, we want to highlight what an LLM is and how it works, with a special focus on the Attention mechanism and how the uncertainty of an LLM can be calculated. This will serve as a basis for correctly interpreting the results analyzed in Section 6.

There will also be a broader discussion on agents in a strict sense and "LLM agents" to better show the difference between our implementation and what is currently being discussed in the literature.

To better define the context of this thesis, we will also examine the main alternative approaches to solving a logistical problem currently studied in the literature.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is a very wide field, that can be resumes as systems designed to perform tasks traditionally intelligence, such as *natural language understanding*, visual perception, decision-making, and problem-solving.

In recent years, AI has rapidly evolved, driven by advances in deep learning[1], increased computational power, and the easy availability of massive datasets (models are even trained on the entirety of the internet). Early AI systems, including expert systems and early machine learning models, relied on manually crafted rules or statistical techniques. However, with the rise of neural networks, particularly deep learning models, AI has shifted toward self-learning systems capable of extracting complex patterns from raw data.

One of the key breakthroughs in this evolution was the development of deep neural networks (DNNs), particularly Convolutional Neural Networks (CNNs) for image processing, introduced by Krizhevsky et al. [11] and Recurrent Neural Networks (RNNs) for sequential data, including language modeling, introduced by Hochreiter et al. [7].

Despite their success, RNNs struggled with long-term dependencies due to vanishing gradients[2], leading to the development of the *Transformer architecture* (from Vaswani et al., Attention Is All You Need [20]), which eliminated recurrence in favor of self-attention mechanisms, significantly improving efficiency and scalability in natural language processing. This shift enabled the emergence of large-scale AI models, particularly in NLP, where two main categories dominate: discriminative models and generative models.

Discriminative models are a class of machine learning models that aim to directly model the decision boundary between different classes in a dataset. Unlike generative models, which learn the underlying distribution of the data, discriminative models focus on learning the conditional probability of a target class given the input features. Classical models like Support Vector Machines[3] and Conditional Random Fields[4] have been widely used for text classification and sequence labeling tasks such as Named Entity Recognition (Lafferty et al. [12]). More recently, deep learning-based models

[1] https://en.wikipedia.org/wiki/Deep_learning

[2] https://en.wikipedia.org/wiki/Vanishing_gradient_problem

[3] https://en.wikipedia.org/wiki/Support_vector_machine

[4] https://en.wikipedia.org/wiki/Conditional_random_field

like BERT (Devlin et al. [4]) have been invented, that leverage contextualized word representations to improve performance on tasks like sentiment analysis, intent detection, and slot filling.

Generative models learn the underlying data distribution to create new samples that resemble the original data. This category includes several architectures that have pushed the boundaries of AI-generated content. Variational Autoencoders (Kingma and Welling [10]) introduced a probabilistic approach to generating structured data, while Generative Adversarial Networks (Goodfellow et al. [5]) refined the concept by using two competing neural networks, a generator and a discriminator, to iteratively improve synthetic data generation. More recently, diffusion models (Ho et al. [6]) have surpassed GANs in generating high-quality images by modeling data transformations through iterative denoising processes. In the domain of text generation, autoregressive models like GPT (Radford et al. [15]) demonstrated the power of large-scale, unsupervised pretraining. These Large Language Models predict the next token (that can be seen as a building block of a word, approximately a syllable) in a sequence based on vast amounts of textual data, learning contextual nuances and producing human-like responses.

2.2 Large Language Models - LLMs

Large Language Models (LLMs) are a class of deep learning models that leverage transformer architectures to generate coherent and contextually relevant text. These models have revolutionized natural language processing by achieving state-of-the-art performance on a wide range of tasks, including language modeling, translation, summarization, and question-answering.

The transformer architecture, introduced by Vaswani et al. in the paper "Attention Is All You Need" [20], is the foundation of LLMs. It consists of an encoder-decoder structure, where the encoder processes the input sequence and generates a sequence of hidden states, while the decoder generates the output sequence based on the encoder's hidden states. The key innovation in transformers is the self-attention mechanism, which allows the model to weigh the importance of different input tokens when generating the output. This mechanism enables transformers to capture long-range dependencies and contextual information more effectively than RNNs.

2.2.1 Attention Mechanism

The attention mechanism is a fundamental component of the transformer architecture (Figure 2.1), enabling the model to focus on specific parts of the input sequence when generating the output. The attention mechanism computes a weighted sum of the input tokens, where the weights are learned during training based on the relevance of each token to the current context.

The self-attention mechanism works in this way:

1. create 3 vectors from embeddings (query, key, value) multiplying by 3 matrices learned during the training process;

2. calculate a score that determines how much focus goes to different parts of the input sentence as it encodes a word;

3. divide the score for more stable gradients and apply softmax;

4. multiply each value vector by the softmax score to keep the value of the word it focuses on, and sink other irrelevant words;

5. sum the weighted value vectors: this produces the output of the self-attention layer at this position.

The self-attention operation computes the relevance of each token in the input with respect to the query token using the scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

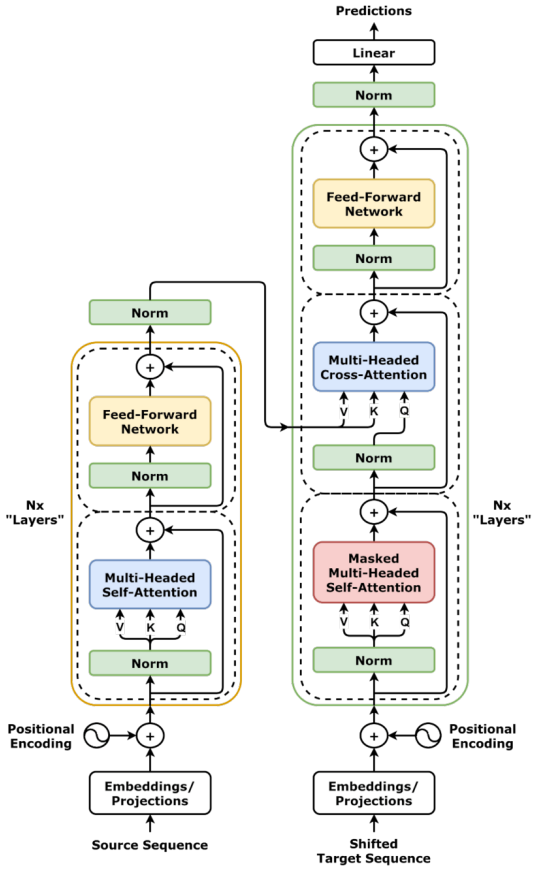


Figure 2.1: Transformer Architecture
Source: Vaswani et al., Attention Is All You Need [20]

where d_k is the dimensionality of the key vectors, ensuring that the dot products do not grow too large as input size increases. The softmax function normalizes the scores into attention weights, which determine how much influence each token should have on the final representation.

Multi-head attention extends this mechanism by computing multiple sets of Q, K, V matrices in parallel, allowing the model to capture different aspects of contextual relationships:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each attention head independently applies scaled dot-product attention, and the outputs are concatenated and linearly projected using W^O . This improves the model's ability to encode complex dependencies and contextual meaning.

The attention mechanism allows the model to focus on different parts of the input sequence based on the current context, enabling it to capture long-range dependencies and improve performance on tasks like text generation.

2.2.2 LLMs' Uncertainty

Despite their impressive capabilities, LLMs are inherently probabilistic and can generate responses that are syntactically correct yet factually inaccurate. Understanding and quantifying this uncertainty is crucial for evaluating the reliability of generated text, especially in high-stakes applications such as medical diagnosis, legal advice, or automated decision-making.

For example, if an LLM generates an answer to a yes/no question with probabilities:

$$P(\text{Yes}) = 0.51, P(\text{No}) = 0.49$$

then the model is nearly uncertain, and this information should be communicated rather than presenting "Yes" as a definitive response.

A key consequence of uncertainty is the phenomenon of *hallucination*, where the model generates confident but factually incorrect or fabricated information [9]. Hallucinations arise when:

- the model lacks knowledge about a specific query but still generates an answer;
- the training data contains conflicting or misleading patterns;
- the model overgeneralizes from limited training examples.

Mitigating hallucinations involves uncertainty-aware generation techniques, the most common one is *Retrieval-Augmented Generation* (RAG) [13], which enhance the prompt with additional context from a knowledge base to improve the model's factual accuracy.

The literature studied different approaches to quantify uncertainty in LLMs, and this thesis will use one of the most common methods to quantify the probability of correctness in the generated choice by the agent.

2.2.2.1 Expressing Uncertainty

A study titled "Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs" [23] investigates methods for eliciting confidence from LLMs without accessing their internal parameters or fine-tuning. The researchers propose a framework comprising three components:

- Prompting Strategies: Techniques to elicit verbalized confidence from the model;
- Sampling Methods: Generating multiple responses to assess variability;
- Aggregation Techniques: Computing consistency across responses to determine confidence levels.

The study evaluates these methods on tasks such as confidence calibration and failure prediction across various datasets and LLMs, including GPT-4 and LLaMA 2 Chat.

Key findings indicate that LLMs often exhibit overconfidence when verbalizing their certainty, possibly mirroring human confidence expression patterns. Additionally, as model capabilities increase, both calibration and failure prediction performance improve, though they remain suboptimal. They shown that implementing strategies like human-inspired prompts and assessing consistency among multiple responses can mitigate overconfidence. Notably, while methods requiring internal model access perform better, the performance gap is narrow.

2.2.2.2 Stable Explanations as Confidence Measures

In the pursuit of reliable uncertainty quantification in Large Language Models, the paper "Cycles of Thought: Measuring LLM Confidence through Stable Explanations " [1] introduced a novel framework that assesses model confidence through the stability of generated explanations.

Their approach posits that the consistency of explanations accompanying an answer can serve as a proxy for the model's certainty. Instead of assigning a single probability to an answer, the method generates multiple explanations for the same question and treats each explanation-answer pair as a distinct classifier. A posterior distribution is then computed over these classifiers, allowing for a principled estimation of confidence based on explanation stability. If the model's explanations remain stable across different reasoning paths, it suggests high confidence in the answer. Conversely, significant variation in explanations signals uncertainty. Empirical evaluations across multiple datasets demonstrated that this framework enhances confidence calibration and failure prediction, outperforming traditional baselines.

However, there are some potential drawbacks. The method requires generating multiple explanations, which increases computational cost and latency. Additionally, it can be sensitive to prompt variations, and may misinterpret repetitive patterns as high confidence.

2.2.2.3 Tokens' log-probability

The paper "Robots That Ask For Help: Uncertainty Alignment for Large Language Model Planners" [17] introduces the KnowNo framework, which is the one we took inspiration from to quantify the uncertainty of the agent in this thesis.

The KnowNo framework leverages Conformal Prediction (CP)[5], a statistical method that provides formal guarantees on the reliability of predictions, to assess uncertainty.

In the paper, they ask the LLM to generate a set of five action for a given prompt (since the `logit_bias` parametere in the OpenAI API was limited to five tokens at that time), and then they ask again for the action to select. This will not be the case of this thesis, since the actions will always be the same, but we will use the same math behind the uncertainty calculation.

KnowNo computes uncertainty evaluating the "validity" of each option. For each action, CP calculates a confidence interval based on previous data and task context, and from this, a set of valid actions is generated. This set can include one or more actions, and the size of this set is indicative of the level of uncertainty:

- Singleton: If CP narrows down the options to just one action, this indicates low uncertainty, and the robot can proceed confidently with the task. The model is highly certain that this action is the most appropriate next step.
- Multiple Options: When CP leaves multiple possible actions in the valid set, this may indicate high uncertainty. In such cases, KnowNo triggers the robot to request human assistance. This allows the robot to seek clarification when it is unsure, thereby avoiding errors that might arise from acting on uncertain predictions.

Technically speaking, the computation of the uncertainty can be summarize in 5 steps:

1. give each action a single-token label (eg. `A)`, `B)`, `C)`, `D)`, `E)`);
2. use the `logit_bias` parameter in the API to force the model to only answer using these labels;
3. get the log-probabilities of the tokens and scale them;
4. filter the resulting set of option with a threshold computed with CP;
5. either the result will be a singleton or a set of options.

They also say that the framework has the advantage of being model-agnostic, as it can be applied to LLMs out-of-the-box without requiring any fine-tuning, thanks to the "caution" that is given if the resulting filtered set of options is not a singleton.

2.3 Agents

As widely explained in the book "An Introduction to Multiagent Systems" [22], we can summarize the definition of an agent as an autonomous entity that perceives its environment through sensors and acts upon it through effectors, making decisions based on its perceptions and objectives in order to achieve specific goals.

This definition highlights several key aspects of agents:

- Autonomy: Agents operate without direct human intervention, controlling their own actions.
- Perception and Action: They interact with the environment via sensors (perception) and effectors (action execution).
- Decision-making: Agents select actions based on their internal model, goals, and the state of the environment.
- Non-determinism and Adaptability: Since environments are generally non-deterministic, agents must be prepared for uncertainty and potential failures in action execution.
- Preconditions and Constraints: Actions are subject to certain conditions that must be met for successful execution.

[5] `https://en.wikipedia.org/wiki/Conformal_prediction`

Thus, an agent's fundamental challenge is deciding which actions to perform in order to best satisfy its objectives, given the constraints and uncertainties of its environment.

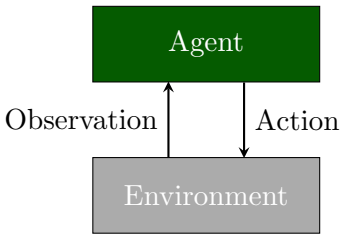


Figure 2.2: Agent Design Scheme
Source: redesign of a scheme in [22]

As shown in Figure 2.2, an agent is some entity that perceives the environment and reacts to it. The setting can be anything from a simple thermostat to a complex system like a self-driving car. The idea is that the agent is able to react to a change in the environment and take actions to achieve its goals.

We will analyze in detail the prompts and the choices in the Chapter 5 Section 5.2, but to give an some anticipation to align our agent with the definition above, we can map some of its concept to what this thesis will analyze:

- Perception and Action: what the server sends about the current state of the environment can be seen as the perception of the agent, while the action it can take will be given in the prompt in a specific way.
- Decision-making: the decision-making process will be the generation of the text by the LLM, weighted by the uncertainty.
- Non-determinism and Adaptability: to emulate the non-determinism of the environment, the state received by the server will be used "raw" in the prompt, without any hard processing or parsing.
- Preconditions and Constraints: being in a "limited" map with a fixed number of cells, is itself a constraint the agent must consider.

2.3.1 BDI Architecture

The Belief-Desire-Intention (BDI) architecture is a widely adopted framework in artificial intelligence (AI) for modeling rational agents. It was formally developed by Rao and Georgeff in 1995 [16] and has been implemented in several architectures, including PRS (1987), dMARS (1998), JAM (1999), Jack (2001), and JADEX (2005). BDI provides a structured approach to practical reasoning, allowing agents to function effectively in dynamic and unpredictable environments.

2.3.1.1 Core Components of BDI

BDI agents operate based on three key components:

- Belief: Represents the agent's knowledge about the world, including past events and observations;
- Desire (Goals): Defines the agent's objectives or preferred end states;
- Intention: Represents the commitments of an agent toward achieving specific goals through selected plans.

BDI has been extensively used in fields like robotics, automated planning, and multi-agent systems.

2.4 State of the Art

A logistic problem is a fundamental challenge in the field of Artificial Intelligence (AI), since depending on the complexity of the specific problem, it can contain tasks such as route optimization, supply chain management, and delivery scheduling. These problems arise in various domains, including transportation, e-commerce, and manufacturing, where efficient resource allocation and decision-making are critical. Given the complexity of modern logistics, AI has emerged as a powerful tool for finding optimal or near-optimal solutions.

Traditional research techniques, such as linear programming and heuristics, have been widely employed. However, with the increasing availability of data and computational power, machine learning (ML) and deep learning methods have become more prevalent. These methods can predict demand, optimize routes dynamically, and enhance decision-making under uncertainty based on the data. Additionally, reinforcement learning (RL) has gained attention for its ability to learn optimal strategies through trial and error, particularly in dynamic and unpredictable environments.

In the recent years with the explosion of Large Language Models (LLMs), many researchers started to apply them to different fields, including planning and logistics.

2.4.1 PDDL Based Solutions

Planning Domain Definition Language (PDDL) is a human-readable format for problems in automated planning that gives a description of the possible states of the world, a description of the set of possible actions, a specific initial state of the world, and a specific set of desired goals.
Source: Wikipedia[6]

The fundamental distinction between a PDDL-based solution and any Machine Learning/Deep Learning approach lies in the very nature of how problems are defined and solved.

In a PDDL-based system, the problem must be explicitly encoded using a formal, structured language that describes the initial state, goal state, and available actions. The planner then takes on the computationally intensive task of exploring a vast search space, systematically generating and evaluating possible action sequences to determine an optimal path from the initial state to the goal state.

While effective in structured environments, this method is inherently time-consuming and computationally demanding. Since the planner must traverse a potentially enormous state space—guided by heuristics but still constrained by the rigid formalism of PDDL, it may struggle with real-time decision-making, making it unsuitable for dynamic, fast-paced applications.

Listing 2.1: Domain file example for a bit toggle problem

```
(define (domain bit-toggle)
  (:requirements :strips :negative-preconditions)
  (:predicates
    (bit ?b)                          ; predicate meaning
                                      ; bit ?b is set (true)
  )

  (:action setbit
    :parameters (?b)
    :precondition (not (bit ?b))      ; can only set a bit if
                                      ; it is not already set
    :effect (bit ?b)                  ; setting the bit to true
  )

  (:action unsetbit
    :parameters (?b)
    :precondition (bit ?b)            ; can only unset a bit if
```

[6] https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language

```
18                                      ; it is currently set
19      :effect (not (bit ?b))          ; setting the bit to false
20    )
21  )
```

Listing 2.2: Problem file example for a bit toggle problem

```
1  (define (problem bit-toggle-full-problem)
2    (:domain bit-toggle-full)
3    (:objects
4       b1 b2 b3
5    )
6    (:init)                            ; Initially all bits are unset (false)
7
8    (:goal                             ; It can be any combination of T/F
9       (and (bit b1) (bit b2) (not(bit b3)))
10   )
11 )
```

With the increasing number of variables (actions or predicates), the number of arcs and nodes grows exponentially. A little example that makes this problem easy to visualize is the Domain where we can have N possible bits, that can be turned to `true` or `false` (Domain file in Listing 2.1) and the Problem where everything start at `false` and we want a specific final combination (Problem file in Listing 2.2).

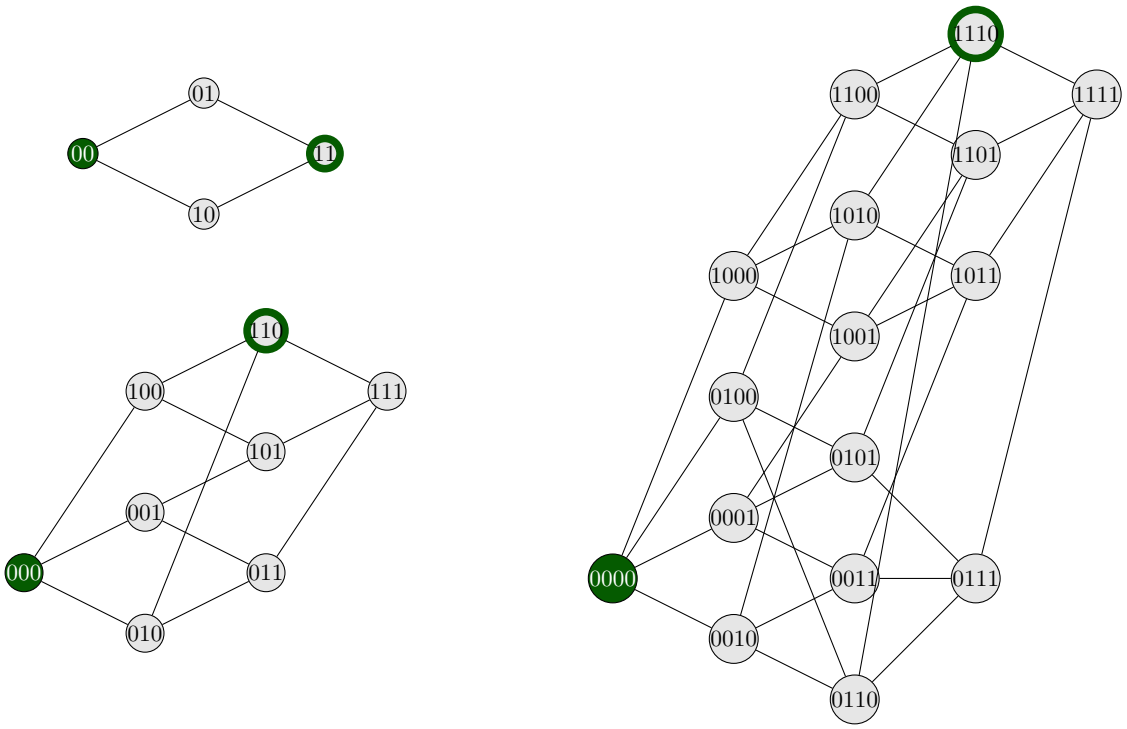
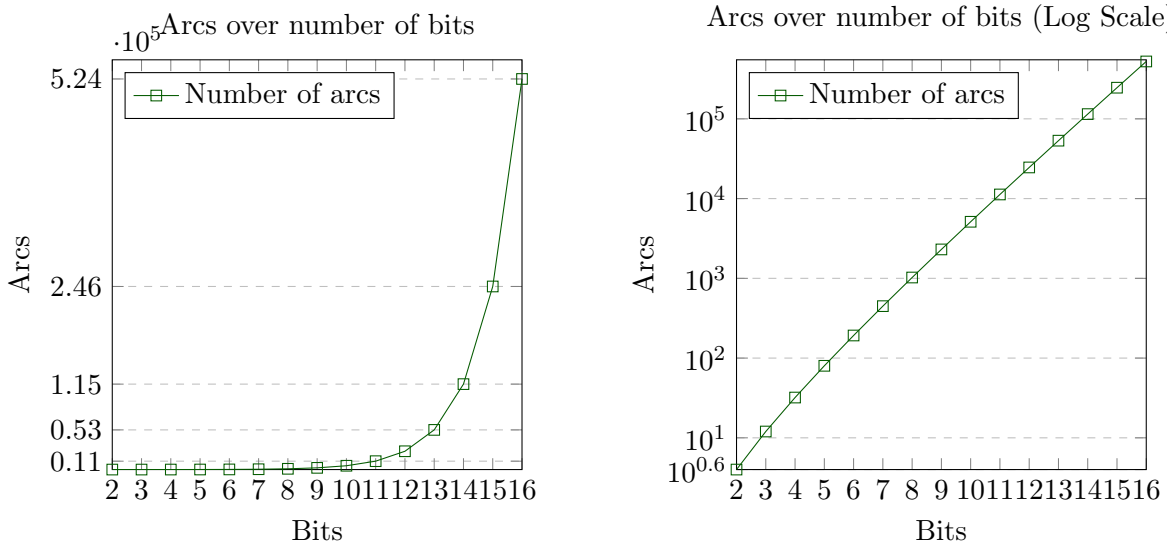


Figure 2.3: Graphs for bit-toggle problem with 2, 3, and 4 bits

As we can see in the plot Figure 2.4, we can visualize the number of arcs (example of graphs for 2, 3 and 4 bits in Figure 2.3) that grows exponentially with the number of bits, as well as the number of states obviously. This shows how even a simple problem with a simple solution can become time-intensive and not suitable for real-time applications.

Listing 2.3: Plan for the bit toggle problem, solved by LAMA-first planner

```
1          ; Found Plan (output)
2  (setbit b3)
```



(a) label 1

(b) label 1

Figure 2.4: Arcs per Bit

```
3 (setbit b2)
4 (setbit b1)
```

However, a PDDL approach is more explainable, since all the information is provided by the user and the output result is a sequence of actions (example at Listing 2.3). This makes it easier to understand and debug the solution, as each step is explicitly defined. Of course, there might be different paths to reach the goal, and the planner might choose one based on heuristics or optimization criteria. This transparency in the decision-making process is one of the key advantages of using PDDL for planning problems.

Literature: an example of a problem related to the one presented in this thesis, solved using PDDL, can be found in the paper "An AI Planning Approach to Emergency Material Scheduling Using Numerical PDDL" by Yang et al. [24].

In their work, they utilize PDDL 2.1 that allows to model the scheduling problem, incorporating factors such energy consumption constraints. Their approach employs the Metric-FF planner to generate optimized scheduling plans that minimize total scheduling time and transportation energy usage. However, while this demonstrates the applicability of AI planning to emergency logistics, their model simplifies the real-world scenario by assuming predefined transport routes, limited vehicle types, and abstract representations of congestion effects. This highlights a broader limitation of PDDL in capturing the full complexity of dynamic and uncertain environments often encountered in emergency response situations.

2.4.2 Reinforcement Learning Solutions

Reinforcement Learning (RL) is a branch of machine learning focused on making decisions to maximize cumulative rewards in a given situation. Unlike supervised learning, which relies on a training dataset with predefined answers, RL involves learning through experience. In RL, an agent learns to achieve a goal in an uncertain, potentially complex environment by performing actions and receiving feedback through rewards or penalties.

Source: GeegksforGeeks [7]

Reinforcement Learning is a learning setting, where the learner is an Agent that can perform a set of actions depending on its state in a set of states and the environment.

It works by defining:

[7] `https://www.geeksforgeeks.org/what-is-reinforcement-learning/`

- **Environment**: the world in which the agent operates
- **Agent**: the decision-maker that interacts with the environment
- **Actions**: the possible moves the agent can make
- **Rewards**: the feedback the agent receives for its actions
- **Policy**: the strategy the agent uses to select Actions

In performing action `a` in state `s`, the learner receive an immediate reward `r(s,a)`. In some states, some actions could be not possible or valid.

The task is to learn a policy (a full specification of what action to take at each state) allowing the agent to choose for each state the action maximizing the overall reward, including future moves.

To deal with this delayed reward problem, the agent has to trade-off exploitation and exploration:

- **Exploitation**: the agent chooses the action that it knows will give some reward
- **Exploration**: the agent tries alternative actions that could end in bigger rewards

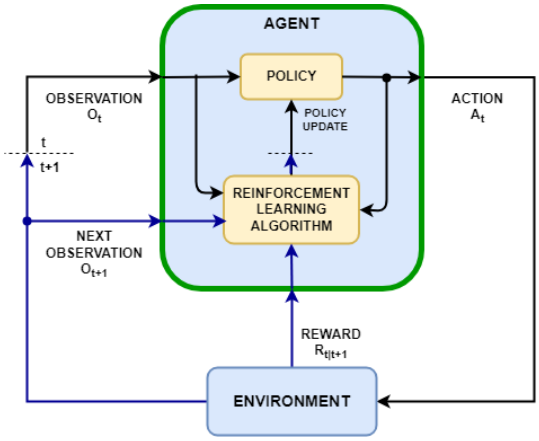


Figure 2.5: RL Agent Scheme
Source: Mathworks[8]

When considering a logistics problem, reinforcement learning naturally comes to mind. This is because defining a reward function is relatively straightforward: it could be measured in terms of packages delivered per minute, per step, or a similar metric. Additionally, the entire process can be simulated in a virtual environment, allowing multiple parallel simulations to accelerate the agent's learning process. As illustrated in Figure 2.5, the structure of the Reinforcement Learning framework closely resembles the agent-based model depicted in Figure 2.2. In both cases, the agent interacts with its environment, receives feedback in the form of rewards, and continuously refines its policy to optimize future performance.

However, RL has its own set of challenges. The most common one is the convergence to a local minimum in the reward function. This means that the agent might learn a suboptimal strategy that is not the best one. Moreover, RL is not explainable, meaning that we can't understand why the agent took a specific action in a specific situation.

Another issue with RL is the cost of training. Since the agent learns through trial and error, it needs to perform a large number of actions to explore the environment and learn the best strategy. This can be computationally expensive and time-consuming, especially for complex problems with many variables and states. Moreover, once the agent is trained, its adaptability to new environments or situations is limited, as it is optimized for a specific reward function and environment configuration.

[8] `https://it.mathworks.com/help/reinforcement-learning/ug/create-agents-for-reinforcement-learning.html`

Literature: an example of a problem similar to the one presented in this thesis, solved using Reinforcement Learning, can be found in the paper "DeliverAI: a distributed path-sharing network based solution for the last mile food delivery problem" by Ashman et al. [14].

They aimed at solving the last-mile delivery problem by developing a distributed path-sharing network based on Reinforcement Learning. Their approach uses a multi-agent system to optimize delivery routes and schedules, considering factors such as traffic congestion, delivery time windows, and vehicle capacity.

However, their model simplifies the real-world scenario by assuming fixed delivery locations and known traffic patterns, which may not accurately reflect the dynamic and uncertain nature of real-world logistics environments. Moreover, their approach requires extensive training and tuning to achieve optimal performance.

2.4.3 Planning in LLM

LLMs are trained on vast amounts of textual data and have demonstrated remarkable performance across a wide range of language tasks, from translation and summarization to reasoning and problem-solving. This success has naturally led researchers to explore whether these models can be repurposed for more complex, multi-step decision-making problems that require planning.

The key idea is that the same abilities that allow LLMs to understand and generate language can be harnessed to decompose a planning task into intermediate steps, reason about the consequences of actions, and even generate entire action sequences with minimal or no task-specific training.

2.4.3.1 Chain-of-Thought Reasoning

One of the most influential ideas for using LLMs in planning is chain-of-thought (CoT) prompting. Instead of asking the model to jump directly from a problem statement to a final answer, CoT prompting encourages the model to "think aloud" by generating intermediate reasoning steps. This decomposition can help in planning problems where the solution involves multiple, logically connected steps.

This was first discovered by Wei et al. [21], who demonstrated that prompting the LLM to 'answer step by step' led to improved performance on mathematical problems compared to requesting only the final answer. They also showed that this step-by-step approach could be applied to other fields, ultimately giving rise to Chain-of-Thought (CoT) reasoning.

"Reasoning" Models Reasoning-focused LLMs are trained to generate multiple Chain-of-Thought (CoT) steps, exploring different solution paths before selecting the most optimal one, often using Reinforcement Learning (RL) [3] techniques such as RLHF (Reinforcement Learning from Human Feedback) or self-consistency methods during the training.

This approach enhances both accuracy and explainability, as the model articulates its reasoning process while still operating as a generative AI system. Expanding on this concept, reasoning models can integrate external tools, memory, and API calls, forming what is commonly referred to as an LLM Agent, capable of autonomous decision-making and real-world interaction.

Last and more famous reasoning models released to the public are by:

- **OpenAI**: o1[9], o1-mini[10] and o3-mini[11] are reasoning models designed to enhance logical problem-solving capabilities. The o1 model is tailored to tackle complex problems across various domains, offering robust reasoning skills. Building upon this foundation, o3-mini provides a more cost-effective and faster alternative, optimized for tasks in science, technology, engineering, mathematics (STEM), and coding;

- **DeepSeek**: DeepSeek-R1[12], is a notable AI development from a startup[13]. Released in early 2025, DeepSeek-R1 is recognized for its powerful reasoning and coding skills, achieved at a

[9] https://openai.com/o1/
[10] https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/
[11] https://openai.com/index/openai-o3-mini/
[12] https://github.com/deepseek-ai/DeepSeek-R1
[13] https://www.deepseek.com/

fraction of the development cost compared to other leading models. Its open-source nature and efficiency have made it a significant player in the AI landscape.

2.4.3.2 Zero-Shot and Few-Shot Planning

In zero-shot planning, LLMs generate action sequences by utilizing their extensive pretraining on text and code, effectively inferring plausible step-by-step solutions to given tasks. Few-shot planning further enhances this by providing LLMs with a small set of demonstrations, enabling them to generalize patterns and improve their action sequencing capabilities.

However, while LLMs can produce reasonable plans, their direct applicability to embodied environments remains challenging. Huang et al. [8] highlight the limitations of naive LLM planning, noting that LLMs struggle with real-world constraints, action feasibility, and long-horizon dependencies. Their work demonstrates that these shortcomings can be mitigated by leveraging the world knowledge embedded within LLMs and applying structured guidance, such as constraints on action generation and feedback-based refinements.

Similarly, Silver et al. [19] extend this inquiry to classical AI planning domains by evaluating few-shot prompting of LLMs on problems expressed in the Planning Domain Definition Language (PDDL). Their findings reveal mixed results: while LLMs can generate syntactically correct PDDL plans in certain domains, they often fail due to a lack of explicit access to transition models and logical constraints inherent to planning problems. Nonetheless, their study also introduces a hybrid approach where LLMs are used to initialize heuristic-based search planners, demonstrating that even imperfect LLM-generated plans can improve the efficiency of traditional AI planning methods.

These findings collectively suggest that while LLMs alone are not yet fully capable of robust autonomous planning, their ability to extract and apply commonsense knowledge makes them valuable tools for augmenting structured planning frameworks. By integrating LLM-generated outputs with classical search-based methods, researchers have shown improvements in planning efficiency and problem-solving robustness, highlighting a promising direction for future research at the intersection of language models and automated planning.

Literature: in the paper "Exploring and Benchmarking Planning Capabilities of Large Language Models" by Bohnet et al. [2], the authors systematically analyze the planning capabilities of LLMs through a novel benchmarking suite that includes both classical planning tasks (expressed in PDDL) and natural language-based planning problems. Their work highlights the limitations of LLMs in planning, particularly their tendency to generate suboptimal or incorrect plans despite their strong language understanding capabilities. To address these shortcomings, they explore various methods to improve LLM-based planning (including many-shot in-context learning, fine-tuning with optimal plans, and the use of chain-of-thought reasoning techniques such as Monte Carlo Tree Search (MCTS) and Tree-of-Thought (ToT)). The results indicate that, while LLMs struggle with planning in zero-shot and few-shot settings, their performance significantly improves when provided with structured demonstrations and reasoning strategies. Moreover, fine-tuning on high-quality plan data leads to near-perfect accuracy in some cases, even with relatively small models. However, challenges remain in out-of-distribution generalization, where models fail to generalize effectively to novel scenarios without additional training. Their analysis also identifies key failure modes in LLM planning, such as constraint violations, failure to reach goal states, and incorrect action sequences, emphasizing the need for better training data curation and reasoning frameworks.

Literature: in "Generalized Planning in PDDL Domains with Pretrained Large Language Models" by Silver et al. [18], the authors investigate whether LLMs, specifically GPT-4, can serve as generalized planners, not just solving a single planning task, but synthesizing programs that generate plans for an entire domain. They introduce a pipeline where GPT-4 is prompted to summarize the domain, propose a general strategy, and then implement it in Python. Additionally, they incorporate automated debugging, where GPT-4 iteratively refines its generated programs based on validation feedback. Their evaluation on seven PDDL domains demonstrates that GPT-4 can often generate efficient, domain-specific planning programs that generalize well from only a few training examples. The study also finds that automated debugging significantly improves performance, while the effectiveness

of Chain-of-Thought (CoT) summarization is domain-dependent. Notably, GPT-4 outperforms previous generalized planning approaches in some cases, particularly when leveraging semantic cues from domain descriptions. However, limitations remain, especially in handling domains requiring deeper structural reasoning or non-trivial search processes. Their results suggest that LLMs can be powerful tools for generalized planning when properly guided, but refinements in prompting strategies and failure correction mechanisms are needed.

3 Experiment Setting

3.1 Problem Definition

3.2 Environment - Deliveroo.js

3.3 GPT Models

4 Agent Development

4.1 First Approach

4.2 Second Approach

4.3 Final Agent

4.4 Closest Cell to the Goal

5 Data Collection

5.1 Visualize the Attention

5.2 Prompts

5.3 Prompt Creation Choices

5.4 Heatmap Generation

6 Results Discussion

6.1 Stateless

6.2 Stateful

6.3 Stateless and Stateful Combined results

6.4 Closest Cell to the Goal Problems

6.5 Models Comparison

7 Future Works

8 Conclusions

Bibliography

[1] Evan Becker and Stefano Soatto. Cycles of thought: Measuring llm confidence through stable explanations, 2024.

[2] Bernd Bohnet, Azade Nova, Aaron T Parisi, Kevin Swersky, Katayoon Goshvadi, Hanjun Dai, Dale Schuurmans, Noah Fiedel, and Hanie Sedghi. Exploring and benchmarking the planning capabilities of large language models, 2024.

[3] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[8] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022.

[9] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, March 2023.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[12] John Lafferty, Andrew Mccallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289, 01 2001.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[14] Ashman Mehra, Snehanshu Saha, Vaskar Raychoudhury, and Archana Mathur. Deliverai: Reinforcement learning based distributed path-sharing network for food deliveries, 2024.

[15] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[16] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor R. Lesser and Les Gasser, editors, *1st International Conference on Multi Agent Systems (ICMAS 1995)*, pages 312–319, San Francisco, CA, USA, 12-14 June 1995. The MIT Press.

[17] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners, 2023.

[18] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models, 2023.

[19] Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[21] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[22] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 1 edition, 2002. Chapter 2.

[23] Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms, 2024.

[24] Liping Yang and Ruishi Liang. An ai planning approach to emergency material scheduling using numerical pddl. In *Proceedings of the 2022 International Conference on Artificial Intelligence, Internet and Digital Economy (ICAID 2022)*, pages 47–54. Atlantis Press, 2022.

Appendix A Attachment

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.