LLM-Based Agent For An E-commerce Shopping Task

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Electronics and Comminication Engineering

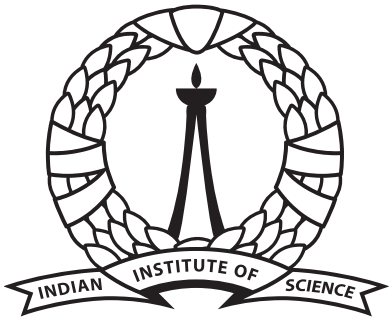
BY

Lokesh Kashyap

SR. No.: 04-02-06-10-51-23-1-22924

Under the guidance of

Prof. Aditya Gopalan



Department of Electrical Communication Engineerin

Indian Institute of Science

Bangalore – 560 012 (INDIA)

June, 2025

Declaration of Originality

I, **Lokesh Kashyap**, with SR No. **04-02-06-10-51-23-1-22924** hereby declare that the material presented in the thesis titled

LLM-Based Agent For An E-commerce Shopping Task

represents original work carried out by me in the **Department of Electrical Communication Engineerin** at **Indian Institute of Science** during the years **2023-2025**.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

Acknowledgements

I would like to express my heartfelt gratitude to Prof. Aditya Gopalan for his exceptional guidance and unwavering support throughout my M.Tech project. His profound expertise, insightful feedback, and constant encouragement were instrumental in shaping the direction of this work. I am especially thankful for his patient mentorship and the valuable one-on-one interactions, which significantly enhanced my learning experience and helped me navigate complex challenges with clarity and confidence

Abstract

The rapid expansion of e-commerce platforms has created a growing demand for intelligent systems that can assist users in navigating vast product catalogs through natural language interactions. In this work, we develop a Large Language Model (LLM)-based agent tailored for the WebShop environment—a simulated online shopping platform designed for training and evaluating conversational agents. Our agent leverages instruction-tuned language models to generate contextually relevant queries and refine them over multiple interactions to retrieve suitable products aligned with user intent.

To enable effective learning, we first bootstrap the agent's behavior through supervised imitation learning, training it on human demonstration data for both query generation and product selection. We then improve the agent using a reward-guided fine-tuning approach with Group Relative Policy Optimization (GRPO), which enhances query generation and product selection based on reward feedback. Experimental results demonstrate that our LLM-based agent significantly improves retrieval effectiveness compared to baseline models, producing more precise and human-aligned queries.

This project highlights the potential of LLMs as intelligent assistants in real-world e-commerce applications, bridging the gap between user intent and product discovery through natural language understanding.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Over the past decade, the rapid proliferation of *e-commerce platforms* has transformed consumer behavior, enabling users to access vast and diverse product catalogs with just a few clicks. As online marketplaces scale to millions of products, effective product discovery becomes increasingly dependent on the sophistication of their underlying *search and recommendation systems.* However, traditional *keyword-based search engines*—though fast and scalable—struggle to capture the complexity and nuance of *natural language queries* that users often employ to express their needs.

Users frequently issue instructions like *"I am looking for a women's oversized black hoodie under 40 dollars"*—requests that are inherently *multi-attribute, open-ended,* and sometimes ambiguous. Such queries require more than simple keyword matching; they demand *contextual understanding, semantic reasoning,* and *flexible query formulation.* This gap highlights the need for intelligent agents capable of interacting with users in *natural language* while translating their intent into effective product searches.

Recent advances in *Natural Language Processing (NLP),* particularly the emergence of *Large Language Models (LLMs),* have significantly pushed the boundaries of machine understanding

and generation. Models such as GPT, LLaMA, PaLM, and Qwen demonstrate remarkable *few-shot and zero-shot* capabilities across a wide range of tasks, from question answering to dialogue systems. Their capacity to understand and generate contextually rich language makes them strong candidates for building intelligent conversational agents in the e-commerce domain.

Within this context, our work explores the integration of LLMs into the WebShop environment—a simulated, instruction-following product search benchmark. The goal is to develop a *language model-based agent* that can interpret user instructions, generate high-quality queries, and iteratively refine its search behavior through feedback. Unlike static rule-based systems, the agent learns dynamically, using *reinforcement learning and multiple query refinements* to align more closely with human-like search behavior.

This project is motivated by the broader vision of making online shopping more intuitive and personalized by bridging the gap between *human intent* and *product* retrieval *using language-first agents.*

1.2 WebShop Environment

WebShop is a large-scale, simulated e-commerce environment developed to benchmark grounded language agents. [11] It consists of over 1.18 million real-world products scraped from Amazon and more than 12,000 crowd-sourced natural language instructions. The core task in *WebShop* involves an agent receiving a *natural language instruction* describing a desired product and then navigating a *multi-page e-commerce* interface to identify and purchase an appropriate item.

The environment is designed to mirror *realistic online shopping behavior.* The agent can issue search *queries, navigate through search results, examine product detail pages,* and finally select the *"Buy"* button once a satisfactory product is found. These interactions are modeled as *high-level, structured actions,* rather than low-level clicks, making WebShop more semantically rich and scalable than prior environments such as MiniWoB.

WebShop presents several key challenges for intelligent agents:

- **Understanding compositional instructions:** These often involve multiple attributes such as *color*, *size*, *brand*, and *price*, requiring the agent to comprehend complex, structured user goals.
- **Query generation and reformulation:** The agent must translate vague or underspecified instructions into effective search queries, and refine them based on intermediate results.
- **Strategic exploration:** To identify the best-matching product, the agent may need to explore multiple options, navigate back and forth between pages, and evaluate trade-offs.
- **Handling noisy web text:** Product descriptions are often inconsistent or informal, demanding robust *natural language understanding* and *error tolerance* from the agent.

To facilitate learning, WebShop provides an *automated reward function* based on how well the selected product matches the instruction in terms of *attributes, options, type, and price.* It is implemented as an OpenAI Gym environment and supports two interaction modes: *HTML mode* for human participants, and *simple mode optimized* for model training.

Overall, *WebShop* serves as a SCALABLE, REALISTIC TESTBED for training and evaluating language agents in GROUNDED, SEQUENTIAL DECISION-MAKING tasks that are closely aligned with real-world E-COMMERCE applications.

1.3 Problem statement

As e-commerce platforms scale to millions of products, enabling users to efficiently find relevant items based on *natural language instructions* becomes increasingly important. Traditional keyword-based search systems are often insufficient, especially when user goals are expressed through multi-attribute, vague, or compositional instructions—such as *"Find a women's black hoodie under Rs. 1000 with an oversized fit and zip".* These instructions require contextual understanding, flexible query generation, and strategic product selection, which conventional systems are not designed to handle.

Large Language Models (LLMs) offer powerful language understanding and generation capabilities, but applying them effectively in dynamic, interactive environments like WebShop—a benchmark simulating real-world e-commerce search—remains a challenging problem. Prior work typically separates tasks like *query generation and product selection* into distinct modules or policies, limiting scalability and introducing hand-engineered structure.

This thesis addresses the problem of building a UNIFIED, PROMPT-CONDITIONED LLM AGENT capable of performing both *query generation and product selection* using a SINGLE POLICY MODEL. The agent is guided by TWO DISTINCT PROMPTS—one for generating search queries and the other for selecting the final product—yet shares the same underlying model weights. Unlike modular architectures, this approach demands that the model learn to adapt its behavior solely based on prompt context.

Training is performed end-to-end within the WebShop ENVIRONMENT, which provides a PROGRAMMATIC REWARD FUNCTION based on how well the final selected product satisfies the user instruction (in terms of attributes, options, and price). The reward is SPARSE, NON-DIFFERENTIABLE, and only available at the final step—making optimization significantly more difficult than standard supervised setups.

A further challenge arises in computing losses and gradients across multiple generations and multiple optimization iterations per instruction. Efficient training in this setting requires:

- Handling variable-length token sequences across generations and action types
- Correctly associating logits and losses with both the query and product prompts
- Applying marginalization across samples for stability and variance reduction
- Managing batch-level efficiency during forward and backward passes

In summary, this thesis investigates the problem of training a single LLM-based policy that:

- Learns both query generation and product selection using prompt variation
- Trains solely on WebShop's sparse environment reward

- Operates under a multi-sample, multi-step optimization loop
- Generalizes across diverse, open-ended product search instructions

1.4 Objectives

The primary objective of this thesis is to design, train, and evaluate a unified language-based agent capable of goal-driven product search in a simulated e-commerce setting using natural language instructions. Specifically, the step-by-step goals are:

1. Develop a single-policy LLM-based agent that can perform two distinct tasks—*query generation* and *product selection*—within the WebShop environment, using prompt conditioning to differentiate between behaviors.
2. Design two distinct prompting strategies to guide the model:
   - A *query-generation prompt* for converting instructions into search queries.
   - A *product-selection prompt* for choosing the most relevant product.
3. Train the agent with imitation learning (IL) on human demonstrations and reinforcement learning, specifically Group Relative Policy Optimization (GRPO), relying solely on WebShop's environment reward.
4. Implement a training loop that supports multiple completions and optimization steps by:
   - Sampling multiple completions per prompt (`num_generations`),
   - Performing multiple updates per instruction (`num_iterations`),
   - Marginalizing token-level losses efficiently across generations and actions.
5. Enable the model to learn prompt-sensitive behavior using shared model parameters.
6. Evaluate the agent using WebShop's benchmark metrics for accuracy and efficiency.

7. Analyze the generalization ability of the model across diverse instruction types and product domains.

By achieving these objectives, this work aims to contribute toward building general-purpose, language-first agents for realistic web-based environments, demonstrating how a single LLM can be trained to handle diverse decision-making behaviors with minimal supervision.

1.5 Contributions

This thesis presents several key contributions toward building a unified language-based agent for instruction-following in e-commerce environments:

1. **Unified Multi-Action LLM Agent:** We develop a single-policy LLM-based agent capable of handling two distinct tasks—*query generation* and *product selection*—within the WebShop environment. Unlike modular or multi-agent setups, our agent uses a *shared model architecture* conditioned solely via prompt structure.

2. **Prompt-Conditioned Task Switching:** We design two custom prompt templates that guide the same model to perform either query generation or product selection. This prompt-conditioning enables task switching without modifying model parameters.

3. **Imitation Learning (IL):**To initialize the agent's behavior, we fine-tune the policy on human demonstrations for both query generation and product selection.

4. **Reinforcement Learning with WebShop Reward:** The agent is trained using *Gradient Regularized Policy Optimization (GRPO)*, relying entirely on *WebShop's sparse environment reward* [11] without any auxiliary verifier. This setting encourages grounded, goal-directed behavior through delayed feedback.

5. **Multi-Sample, Multi-Step Optimization Framework:** We extend the training loop to support `num_generations` (multiple completions per prompt) and `num_iterations`

(multiple optimization steps per instruction), and implement proper marginalization of loss across prompt-action samples.

6. **Efficient Loss Computation Across Prompts:** We implement a unified loss pipeline that handles *variable-length sequences*, *shared tokenization*, and loss aggregation for both query and product prompts in a memory-efficient manner.

7. **Empirical Validation in WebShop:** We evaluate the agent on the WebShop benchmark and demonstrate its ability to generalize across diverse instructions and complete end-to-end product search using only language and environment reward.

Together, these contributions advance the development of *language-first agents* that can reason, explore, and act in real-world-like environments through prompt-guided learning.

Chapter 2

Related Work

A brief description of your chapter.

2.1 Large Language Models for Instruction Following

Large Language Models (LLMs) such as GPT [2], PaLM [4], LLaMA [10], and Qwen [5] have demonstrated strong instruction-following abilities across diverse tasks. Trained on large-scale text corpora and fine-tuned with *supervised instructions*, these models excel in *zero-shot* and *few-shot* scenarios by adapting to tasks through carefully designed prompts.

Recently, Qwen2.5-1.5B-Instruct, an open-source, instruction-optimized model from Alibaba, has proven effective for *multi-turn dialogue* and downstream applications due to its compact size and alignment with *chat-format prompting*.

In this work, we leverage Qwen2.5-1.5B-Instruct not just as a text generator but as a **decision-making agent** trained via **reinforcement learning** in the WebShop environment. Its *prompt-conditioned reasoning* ability and instruction alignment make it suitable for *multi-step product search and selection*.

2.2 Prompt Engineering and Multi-Task Behavior in LLMs

Prompt engineering has become a central mechanism for controlling the behavior of **Large Language Models (LLMs)** without modifying their underlying weights. By carefully designing the format and structure of textual input, a single model can be conditioned to perform a wide variety of tasks—a paradigm referred to as *prompt-conditioned control* [6]. This approach allows instruction-tuned models such as Qwen2.5-1.5B-Instruct [5] to demonstrate strong performance across tasks in both zero-shot and few-shot settings, guided entirely by prompt structure.

In particular, *instruction-style prompting* [6] has proven highly effective in guiding LLMs. These prompts typically contain explicit instructions (e.g., "Translate this sentence..." or "Find the most relevant product...") that define the task to be performed. Variants of this approach have enabled models to perform summarization, question answering, and code generation without task-specific fine-tuning.

Additionally, many instruction-tuned LLMs—such as Qwen2.5-1.5B-Instruct used in this work—are designed to operate in a *chat-style prompting* [5] format, where the input is structured as a multi-turn dialogue between a user and an assistant using roles like `"system"` and `"user"`.

In this thesis, we leverage **prompt-conditioned control** to train a single LLM agent that can perform two distinct actions: *query generation* and *product selection*. Rather than designing separate modules or training distinct models, we construct two prompt templates—one for generating e-commerce search queries and one for selecting the best-matching product from a ranked list. The model is conditioned entirely by the prompt structure, allowing for multi-task behavior using a unified policy.

2.3 Language Agents in Interactive Environments

Recent developments in large language models (LLMs) have sparked interest in building **language agents**—systems that use natural language to reason, explore, and act in interactive environments. These agents go beyond static tasks, aiming to perform long-horizon, decision-making behaviors grounded in real-world-like settings.

A notable example is the *DeepSeek Agent* [1], which introduces a general-purpose agent framework built on top of pretrained language models. It combines language modeling with modular skill composition and external memory, enabling the agent to tackle multi-step tasks such as web navigation, tool use, and long-context decision making. DeepSeek emphasizes *open-ended interaction*, *reasoning*, and *long-horizon planning*.

In contrast to approaches that rely on external memory or tool APIs, our work focuses on a simpler but more unified architecture. We train a **single-policy LLM agent** using **Qwen** [5], an instruction-tuned transformer model. The agent is prompted in two distinct ways—one for generating search queries and another for selecting products—yet the underlying model parameters are entirely shared.

This setup enables prompt-conditioned behavior without relying on modular heads or task-specific architectures. The agent is trained directly within the WebShop environment using its native reward function, allowing it to learn grounded product search behaviors using only sparse feedback. Unlike agents that depend on expert demonstrations or external verifiers, our agent learns from scratch through interaction, reinforcing the viability of fully end-to-end, language-first decision-making systems.

2.4 WebShop and Goal-Conditioned Search

The *WebShop* environment, already introduced in Chapter 1, offers a realistic and scalable testbed for training grounded language agents in **goal-conditioned product search** [11]. Each episode begins with a natural language instruction, and the agent must generate search

queries, navigate product listings, and ultimately purchase the most suitable item using high-level structured actions such as `search[query]`, `click[index]`, and `buy`.

Type	**Argument**	**State → Next State**
search	`[Query]`	Search → Results
choose	Back to search	$*$ → Search
choose	Prev/Next page	Results → Results
choose	`[Product title]`	Results → Item
choose	`[Option]`	Item → Item
choose	Desc/Overview	Item → Item-Detail
choose	Previous	Item-Detail → Item
choose	Buy	Item → Episode End

Table 2.1: Actions in WebShop.

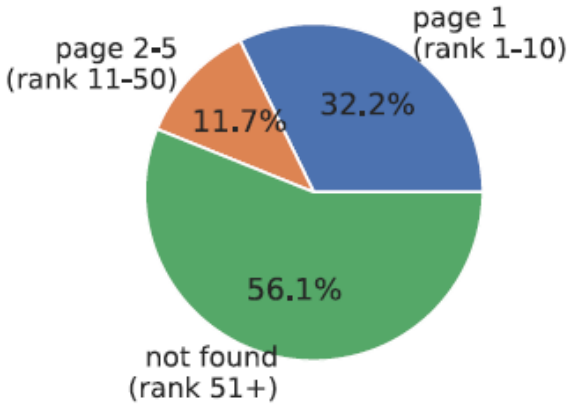


Figure 2.1: Item rank in search results when the instruction is directly used as search query

To facilitate learning, WebShop provides an **automated reward function** that evaluates the selected product based on how well it satisfies the instruction's constraints—such as attributes, options, price, and product category. This makes WebShop a valuable benchmark for evaluating *instruction-following agents* in complex, sequential, and language-driven tasks.

Instruction Format and Reward. Each natural language instruction $u \in \mathcal{U}$ contains a non-empty set of desired attributes U_{att}, a set of options U_{opt}, and a maximum price u_{price}. Instructions are generated by human annotators based on a target product y^*, ensuring that

they reflect realistic user goals. For example, the instruction:

> "Can you find me a pair of black-and-blue sneakers that is good in rain weather? I want it to have puffy soles and price less than 90 dollars."

implies $U_{att} = \{\texttt{waterproof}, \texttt{soft sole}\}$ and $U_{opt} = \{\texttt{color} : \texttt{black and blue}\}$.
The agent receives a reward r only upon choosing a final product. This reward evaluates how well the selected product's attributes Y_{att}, options Y_{opt}, and price y_{price} match the instruction:

$$r = r_{type} \frac{|U_{att} \cap Y_{att}| + |U_{opt} \cap Y_{opt}| + \mathbf{1}[\, y_{price} \leq u_{price}\,]}{|U_{att}| + |U_{opt}| + 1}, \tag{2.1}$$

where r_{type} is a text-matching penalty that reduces the reward if the chosen product is of a different type despite sharing some attributes and options. This sparse, structured reward enables the agent to receive partial credit and guides its learning process.

The original WebShop paper [11] explored several foundational training approaches:

- **Static Prompting:** A frozen, instruction-tuned language model is prompted using hand-crafted templates to generate queries or select products. This approach does not involve any learning or adaptation from environment feedback.
- **Imitation Learning:** Agents are trained via behavior cloning on expert demonstrations collected from human workers who solved WebShop tasks step-by-step. This helps bootstrap agent behavior in realistic and complex scenarios.
- **Reinforcement Learning (RL):** After imitation pretraining, agents are fine-tuned using reinforcement learning, where the WebShop environment reward serves as the optimization signal. The agent is positively rewarded if the selected product matches the instruction based on attributes, price, and product type.

These strategies form the foundation for instruction-following agents in WebShop.

2.5 Reinforcement Learning with Language Models

Reinforcement Learning (RL) has emerged as a powerful framework for aligning large language models (LLMs) with desired behaviors, especially in cases where supervised data is scarce or limited in expressivity. While LLMs are typically pretrained using unsupervised objectives such as next-token prediction, RL enables models to improve performance through interaction with environments, feedback from reward models, or human preferences.

Proximal Policy Optimization (PPO) [8] is one of the most widely used algorithms in LLM fine-tuning, particularly for preference alignment tasks such as *Reinforcement Learning from Human Feedback (RLHF)* [6]. In PPO, the LLM is treated as a policy that generates responses, and its behavior is updated based on a reward model trained to reflect human preferences. PPO optimizes a clipped surrogate objective that balances exploration and stability during updates.

Direct Preference Optimization (DPO) [7] is a recent alternative that eliminates the need for online sampling or PPO-style rollouts. Instead, it treats preference pairs as soft labels and directly optimizes a contrastive loss that encourages preferred outputs to be more likely than dispreferred ones. DPO is sample-efficient and stable, making it attractive for scaling alignment without complex infrastructure.

Group Relative Policy Optimization (GRPO) [9] is a scalable, value-free reinforcement learning algorithm designed to fine-tune language models using groupwise relative preferences. Instead of relying on a value function or reward model, GRPO samples multiple candidate completions for a given input and computes relative advantages by comparing their reward scores within the group. This enables gradient updates that favor higher-reward completions without estimating absolute values.

GRPO is particularly efficient in scenarios with limited or sparse reward signals, as it eliminates the need for separate critic networks. It also supports stable training across multiple generations per prompt and naturally integrates with instruction-following tasks. These char-

acteristics make it well-suited for reinforcement learning in large language models, especially under constrained compute or weak supervision settings.

Unlike prior approaches that employ reward models or **verifier-based optimization** [3] to evaluate the quality or correctness of generated outputs, our work uses only the **native WebShop reward** as the optimization signal. This reward function provides a scalar score based on how well the final selected product satisfies the instruction in terms of attributes, price, and category. By avoiding reliance on external verifiers or learned reward models, our approach remains lightweight and grounded in environment-based feedback.

In our work, we adopt a GRPO-based RL framework to fine-tune an LLM policy that performs both query generation and product selection. Unlike traditional RLHF pipelines that use human preference models, we rely solely on the native WebShop reward to supervise learning. Moreover, we marginalize over multiple generations and update the model across several optimization steps per instruction, requiring loss aggregation and stable learning dynamics. This makes our approach a practical instantiation of RL with LLMs in a real-world grounded environment.

2.6 Comparison with Our Approach

In contrast to prior approaches, our work introduces the following key innovations:

- **Unified Policy:** We develop a single LLM-based policy trained to perform both *query generation* and *product selection*, whereas earlier pipelines used separate modules or decoupled components for these tasks.

- **Prompt-Conditioned Task Switching:** Our model employs prompt conditioning to distinguish between task behaviors, allowing multi-action control using shared model weights and a unified architecture.

- **Multi-Sample, Multi-Step Optimization:** We extend the training loop to support both `num_generations` (multiple completions per prompt) and `num_iterations` (multi-

ple optimization steps per instruction), which requires careful loss marginalization and efficient gradient flow across prompt types and action modes.

- **Reinforcement Learning with WebShop Reward:** The entire training process is guided by WebShop's native reward signal—no external verifier, handcrafted logic, or auxiliary module is used.

These innovations enable our agent to exhibit more flexible, scalable, and adaptive instruction-following behavior in a realistic product search setting.

2.7 Rule-Based Baseline in WebShop

The original WebShop benchmark includes a simple, non-learnable **rule-based baseline**. In this setup, the agent performs three fixed actions:

1. It submits the raw user instruction as a **search query**, without any reformulation or filtering.
2. It **selects and purchases the first product** returned on the results page.
3. For any available product options (e.g., **size**, **color**), it automatically selects the **first option** in each category.

This baseline relies entirely on the lexical retrieval capabilities of the WebShop search engine, functioning as a basic **information retrieval (IR)** pipeline. While it can achieve non-zero attribute or option rewards in certain cases — particularly when the instruction is highly specific and well-matched to the top-ranked product — it lacks the flexibility to handle **natural language variation**, **ambiguity**, or **long-horizon planning**.

Key Limitations:

- **No learning or adaptation:** The behavior is static and cannot improve with feedback.

- **No understanding of ambiguity:** It cannot disambiguate vague, compositional, or multi-attribute instructions.
- **No search strategy:** It cannot explore alternative queries or compare between candidate products.

Algorithm 1: Rule-Based Product Search and Selection

Input: Instruction u

Output: Final selected product p with options

begin

    // --- Agent ---

    $q \leftarrow u$; // Use instruction as query

    // --- Environment ---

    results $\leftarrow$ Search(q) ; // WebShop returns product list

    // --- Agent ---

    $p \leftarrow$ results[0] ; // Select first product

    foreach *option category* c *in* p **do**

        p.selected[c] $\leftarrow$ p.options[c][0] ; // Select first option

    return p

Despite its simplicity, the rule-based baseline serves as a minimal performance threshold. In contrast, our work develops a trainable LLM-based agent that performs both query generation and product selection, guided entirely by reward feedback from the WebShop environment. This enables adaptive, end-to-end behavior and significant improvements in product search success.

Chapter 3

Methodology

3.1 Prompt-Based Multi-Action Control

Large Language Models (LLMs) exhibit remarkable generalization capabilities when guided by carefully crafted prompts. In this work, we implement a prompt-based interface to control the agent's behavior across two tasks: (i) **search query generation and (ii) product selection**. This strategy allows us to train a unified policy model capable of multi-task behavior without modifying the model architecture.

3.1.1 Unified Prompt Design for Multi-Action Behavior

Rather than maintaining separate models or modules for each action, we define distinct prompt templates that condition the model on task-specific behavior. Each prompt follows a role-based structure—commonly used in instruction-tuned chat models—where the input is split into "system" and "user" roles.

3.1.2 Prompt Templates and Execution Flow

The prompts are passed to the model using the format expected by chat-style language models (e.g., Qwen2.5-1.5B-Instruct). The following two templates guide the model to perform either query generation or product selection.

(i) Query Generation Prompt This prompt is used to translate a user's instruction into a concise search query.

```
prompt_query = [
    {"role": "system", "content": "You are a helpful assistant that writes concise
    ↪  e-commerce search queries."},
    {"role": "user", "content": (
        f"Instruction: {instruction}\n\n"
        "Now generate only the search query. Do not include any extra explanation or
        ↪  formatting. "
        "Just write the query as a single line."
    )},
]
query = self.model_chat(prompt_query).strip()
```

Figure 3.1: Prompt template for search query generation

Post-processing ensures that the returned string is clean and usable as an input to the search engine.

(ii) Product Selection Prompt This prompt is used to choose the best product from a top-10 list based on the original instruction and the search query.

```
prompt_select = [
    {"role": "system", "content": "Pick the best product that matches the
    ↪  instruction."},
    {"role": "user", "content": f"""
Instruction: {instruction}
Search Query: {query}

Here are the top-10 search results:
{product_block}

Please respond with **only a single number from 1 to 10**, representing the best
↪  matching product. Do not explain or include any other text. Just write the
↪  number.
"""}
]
selection = self.model_chat(prompt_select)
```

Figure 3.2: Prompt template for product selection

A regular expression is used to extract the numeric response and convert it into an index.

If the model's output is malformed, a fallback index of 0 is used.

3.1.3 Algorithm: Prompt-Based Multi-Action Control

Algorithm 2: Prompt-Based Agent-Environment Loop

Input: Instruction u, policy model M, environment E

Output: Reward r

`// Step 1: Query Generation`

$P_{\text{query}} \leftarrow \text{ConstructQueryPrompt}(u)$;

$q \leftarrow M(P_{\text{query}})$;

`// Step 2: Execute Search in Environment`

$R \leftarrow E.\texttt{search}(q)$; `// Returns top-10 products`

`// Step 3: Product Selection`

$P_{\text{select}} \leftarrow \text{ConstructSelectionPrompt}(u, q, R)$;

$s \leftarrow M(P_{\text{select}})$;

$i \leftarrow \text{ParseNumberFrom}(s)$;

`// Step 4: Buy Product and Receive Reward`

$r \leftarrow E.\texttt{buy}(R[i])$;

return r

3.1.4 Advantages of Prompt-Based Control

- **Simplicity:** No need to change model weights or architecture—behavior is fully determined by prompt content.
- **Scalability:** New actions can be added simply by designing additional prompts.
- **Modularity:** Prompts are human-readable, debuggable, and allow transparent behavior control.

This approach makes it feasible to treat the LLM as a general-purpose decision-making agent, where action switching is entirely managed through prompt design.

3.2 Imitation Learning

Imitation learning (IL) is often used to bootstrap instruction-following agents by mimicking human behavior from expert trajectories. In the WebShop benchmark, two types of supervised IL approaches are introduced: (i) **search query generation, and (ii) click prediction for product selection.**

3.2.1 Imitating Human Search Generation

The second imitation task frames search query generation as a sequence-to-sequence language modeling problem. Given a user instruction u, the agent learns to produce a search action $a = \texttt{search[...]}$, without using any contextual information from previous interactions.

The dataset $\mathcal{D} = \{(u, a)\}_{i=1}^{M}$ consists of $M = 1,421$ instruction-query pairs extracted from 1,012 human trajectories. A BART model is fine-tuned using conditional language modeling to generate the appropriate search query:

$$\mathcal{L}_{\text{search}} = \mathbb{E}_{(u,a)\sim\mathcal{D}} \left[-\log \pi_\phi(a \mid u) \right]$$

This method provides a simple but effective way to train an initial search behavior, which can later be refined via reinforcement learning.

Algorithm 3: Supervised Fine-Tuning for Search Query Generation

Input: Instruction-query dataset $\mathcal{D} = \{(u_i, q_i)\}_{i=1}^{N}$, pre-trained LLM $\mathcal{M}$, tokenizer $\mathcal{T}$

Output: Fine-tuned model $\mathcal{M}^*$ for generating e-commerce search queries

foreach (u_i, q_i) *in* $\mathcal{D}$ **do**

  Format prompt as chat-style message:;

    prompt ← `T.apply_chat_template(`
    `[system: "Generate a search query",`
    `user: "Instruction: {u}"]);`

  Append target query with end token:;

    input.text ← prompt + q_i + ⟨`endofquery`⟩;

  Tokenize full input:;

    `input.ids, attention.mask` ← `T(input.text)`;

  Mask loss on prompt tokens:;

    labels ← input.ids; labels[0:`len(prompt)`] ← −100;

Use language modeling loss:;

  $\mathcal{L}$ = `CrossEntropyLoss`($\mathcal{M}$(`input.ids`), `labels`);

Train model $\mathcal{M}$ using AdamW optimizer for several epochs with gradient accumulation and checkpointing;

return *Fine-tuned model* $\mathcal{M}^*$

3.2.2 Imitating Human Product Selection

In this task, the agent learns to select the most appropriate product from a list of retrieved results based on a given instruction. The agent is trained on a dataset $\mathcal{D}_{\text{select}} = \{(u_i, \mathcal{P}_i, y_i)\}_{i=1}^{N}$, where each sample contains:

- u_i: a natural language instruction describing the user's shopping intent,
- $\mathcal{P}_i = \{p_1, p_2, \ldots, p_K\}$: a list of K candidate products represented as textual summaries (including title, price, and attributes),

- $y_i \in \{1, \ldots, K\}$: the index of the product selected by the human demonstrator.

Model Architecture To model the selection process, we use a transformer-based architecture that encodes both the user instruction and each product candidate. Our architecture consists of:

- A shared transformer encoder (e.g., Qwen or BERT) to obtain contextual embeddings.
- A multi-head cross-attention layer that allows each product candidate to attend over the instruction.
- A mean-pooling and linear projection layer to produce scalar scores for each candidate.

Given the encoded instruction $h_u = \text{LLM}(u)$ and candidate encodings $h_{p_j} = \text{LLM}(p_j)$, we compute the cross-attended representations:

$$\tilde{h}_{p_j} = \text{CrossAttn}(h_{p_j}, h_u, h_u)$$

We then compute scores for each candidate as:

$$S(u, p_j) = \mathbf{w}^\top \cdot \text{mean}(\tilde{h}_{p_j})$$

These scores are normalized via softmax to produce a probability distribution over all candidates:

$$\pi_\theta(p_j \mid u, \mathcal{P}) = \frac{\exp(S(u, p_j))}{\sum_{k=1}^{K} \exp(S(u, p_k))}$$

Training Objective The model is trained using a cross-entropy loss over the selected product index:

$$\mathcal{L}_{\text{select}} = \mathbb{E}_{(u,\mathcal{P},y)\sim\mathcal{D}_{\text{select}}} \left[-\log \pi_\theta(p_y \mid u, \mathcal{P}) \right]$$

Algorithm 4: Training Imitation Model for Product Selection

Input: Dataset $\mathcal{D} = \{(u_i, P_i, y_i)\}_{i=1}^{N}$ where u_i is an instruction, P_i is a list of products, and y_i is the human-selected index

Output: Trained model parameters

Initialize model encoder, cross-attention, scorer;
Initialize optimizer;
for *epoch* = 1 **to** *num_epochs* **do**
    for *(batch_instructions, batch_products, batch_labels) in DataLoader($\mathcal{D}$)* **do**
        instruction_tokens ← tokenize(batch_instructions, repeat per candidate);
        product_tokens ← tokenize(all candidates);
        instruction_embs ← encoder(instruction_tokens);
        product_embs ← encoder(product_tokens);
        attended_prod_embs ← cross_attention(query=product_embs, key=instruction_embs, value=instruction_embs);
        pooled_embs ← mean(attended_prod_embs **along** sequence_length);
        scores ← scorer(pooled_embs);
        scores_grouped ← split(scores, by number of candidates per instruction);
        loss ← 0;
        for *each* $(score\_vec, y)$ *in* $(scores\_grouped, batch\_labels)$ **do**
            log_probs ← log_softmax(score_vec);
            loss ← loss − log_probs[y];
        loss ← loss / batch_size;
        Backpropagate loss;
        optimizer.step();
        optimizer.zero_grad();
return trained model;

Implementation Details We use the Qwen2.5-1.5B-Instruct model for both query and product representations. Product texts are formatted as concatenated strings (title, price, attributes), and the model is trained using AdamW with a learning rate of 2×10^{-5}. The training loop supports variable-length candidate sets and masks out padding via score clipping.

3.3 Reinforcement Learning with GRPO

3.3.1 Why Group Relative Policy Optimization?

Training language models for decision-making in complex environments such as WebShop [11] poses several challenges:

- **Sparse rewards:** The agent receives a meaningful reward only after selecting a product, making credit assignment difficult.
- **Delayed feedback:** Many actions (e.g. query generation) happen long before the final reward is observed.
- **No stable value baseline:** Traditional policy-gradient methods like PPO require training a separate value function or reward model [8], which can be unreliable for language models and text-based policies.

Motivation. To address these challenges, we adopt **Group Relative Policy Optimization (GRPO)** [1], a reinforcement learning algorithm that updates the policy by comparing multiple sampled completions relative to one another. Unlike PPO:

- Requires **no critic network or reward model** — the WebShop environment's native reward is used directly.
- Operates on **groups of completions**, computing relative advantages by comparing each completion's reward to the mean reward of its group.
- Provides **low-variance gradients**, since the baseline is computed per group, making training stable even with sparse rewards.

Suitability for our setup. In our work, the policy must generate both queries and product selections using a single language model. GRPO seamlessly supports:

- Multiple sampled generations per instruction (`num_generations`), allowing better exploration of diverse actions.
- Marginalization across multiple seeds (`num_marginals`), reducing gradient noise by averaging per logical completion.
- Multiple optimization iterations per instruction (`num_iterations`), improving sample efficiency by updating the model multiple times on the same sampled completions.
- Efficient training without auxiliary models, making it practical for large-scale fine-tuning on complex, interactive environments like WebShop [11].

3.3.2 Formal Derivation of the GRPO Objective

For each instruction u, we draw G logical generations (number of generations), and for each logical generation we draw M marginal completions (number of marginals), for a total of $N = G \times M$ completions per instruction:

$$\mathcal{C}_u = \{c_{g,m}\}_{g=1,\ldots,G;\ m=1,\ldots,M}.$$

Here, each completion $c_{g,m}$ denotes one full interaction trajectory under instruction u, consisting of:

- the search query generated by the model,
- the top-10 product list retrieved by the environment,
- the product index selected by the model as the final action,
- the resulting reward received from the environment.

Per-marginal advantage calculation. For each marginal m ($m \in \{1, \ldots, M\}$), we generate K completions ($k \in \{1, \ldots, K\}$) and obtain rewards:

$$\{\, r_{m,1},\, r_{m,2},\, \ldots,\, r_{m,K} \,\}.$$

We first compute the mean reward within that marginal:

$$\bar{r}_m = \frac{1}{K} \sum_{k=1}^{K} r_{m,k}.$$

Then, we compute the per-generation advantage for each completion:

$$A_{m,k} = r_{m,k} - \bar{r}_m, \qquad k \in \{1, \ldots, K\},$$

which yields a vector of length K.

Log-Probabilities and Policy Decomposition. Let $\pi_\theta(c_{g,m} \mid u)$ denote the policy under parameters θ that generates completion $c_{g,m}$ given instruction u, and let π_{old} and π_{ref} be frozen and/or reference policies as needed. Each completion can be factorized into a query $y_{g,m}$ and a product selection $a_{g,m}$:

$$\pi_\theta(a_{g,m} \mid s) = \sum_{y_{g,m}} \pi_\theta(y_{g,m} \mid s)\, \pi_\theta(a_{g,m} \mid s, y_{g,m}).$$

Gradient derivation. Differentiating with respect to θ yields:

$$\nabla_\theta \pi_\theta(a \mid s) = \sum_y [\pi_\theta(a \mid s, y)\, \nabla_\theta \pi_\theta(y \mid s) \;+\; \pi_\theta(y \mid s)\, \nabla_\theta \pi_\theta(a \mid s, y)]$$

which we approximate with Monte Carlo samples.

$$\nabla_\theta \pi_\theta(y_j) \approx \frac{1}{n} \sum_{i=1}^{n} \left[\nabla_\theta \pi_\theta(y_i \mid s) + \pi_\theta(y_i \mid s)\, \nabla_\theta \log \pi_\theta(y_i \mid s) \right].$$

Training objective. Given sampled completions $c_{g,m} = (y_{g,m}, a_{g,m})$, the advantage-weighted objective is:

$$\hat{\mathcal{L}}(\theta) = -\frac{1}{GM} \sum_{g=1}^{G} \sum_{m=1}^{M} \hat{A}_{g,m} \log \pi_\theta(c_{g,m} \mid u)\,.$$

$$L(\theta) = -\frac{1}{GM} \sum_{g,m} \hat{A}_{g,m}[(\log \pi_\theta(y_{g,m} \mid s) + \log \pi_\theta(a_{g,m} \mid s, y_{g,m}))],$$

which matches the implementation where the query and selection losses are computed separately and summed. However, in practice we decode the query and product selection separately for efficiency. This requires scaling the query part of the gradient by $\pi_\theta(a \mid s, y)$ during backpropagation — as implemented in code — so that the gradient matches $\nabla_\theta \log \pi_\theta(a \mid s)$ under the hierarchical policy.

KL-regularization. To prevent the policy from deviating excessively from a fixed reference policy, we add a KL-penalty term to the objective. Specifically, we compute the per-token KL divergence between the current policy π_θ and the reference policy π_{ref} for both the query and selection token distributions. Given log-probabilities $\log \pi_\theta$ and $\log \pi_{\text{ref}}$, we approximate the per-token KL as:

$$\text{KL}_t \approx \exp(\log \pi_{\text{ref}} - \log \pi_\theta) - (\log \pi_{\text{ref}} - \log \pi_\theta) - 1,$$

which is a numerically stable expression for $\text{KL}(\pi_{\text{ref}} \,\|\, \pi_\theta)$. To control its effect on the gradient, we clip each per-token KL value to a maximum of `MAX_KL`:

$$\text{KL}_t \leftarrow \text{clip}(\text{KL}_t,\, 0,\, \texttt{MAX\_KL}),$$

and sum across all tokens in the sequence. Finally, this regularization is scaled by a small coefficient β before being added to the overall loss:

$$L(\theta) \leftarrow L(\theta) + \beta\, \text{KL}_t.$$

In practice, we apply this procedure separately to the query and selection token distributions.

Training loop. Finally, we repeat this process for `num_iterations` optimization steps per instruction to improve data efficiency. The overall objective is averaged across all instructions in the batch.

Algorithm 5: Training with GRPO

Input: num_generations G, num_marginals M, num_iterations I, policy π_θ

Output: Updated policy parameters θ

foreach *training step* **do**
  $u \leftarrow \mathcal{E}$.`start_session()`;
  `marginal_data` $\leftarrow [\,]$; `// Store all marginals`
  for $m \leftarrow 1$ **to** M **do**
    `env_results` $\leftarrow [\,]$; `// Store generations per marginal`
    for $g \leftarrow 1$ **to** G **do**
      $(q_{g,m}, a_{g,m}, r_{g,m}) \leftarrow$ `generate_and_select`(u);
      Append $(q_{g,m}, a_{g,m}, r_{g,m})$ to `env_results`;
    `inputs`$_m \leftarrow$ `tokenize_and_prepare(env_results)`;
    Append `inputs`$_m$ to `marginal_data`;
  for *iter* $\leftarrow 1$ **to** I **do**
    ; `// Multiple optimization steps per instruction`
    `all_losses` $\leftarrow [\,]$;
    foreach *inputs*$_m \in$ *marginal_data* **do**
      $L_m \leftarrow$ `compute_GRPO_loss`$(\pi_\theta,$ `inputs`$_m)$;
      Append L_m to `all_losses`;
    $L \leftarrow$ `mean(all_losses)`;
    $\theta \leftarrow$ `optimize`(θ, L);

Chapter 4

Experiments and Results

4.1 Experimental Setup

This section describes the datasets, evaluation metrics, baseline, and training setup for our experiments. The *WebShop environment* and its components have been introduced in Chapter 1.

4.1.1 Dataset and Splits

We use the official ***WebShop*** splits [11]: 10,000 instructions for training, 1,000 for validation, and 500 for testing. The underlying product catalog contains approximately 1.18 million items, as described earlier.

4.1.2 Evaluation Metrics

Each policy is evaluated using the standard **WebShop** metrics:

- ***Success Rate (%)***: The percentage of episodes in which the agent successfully selects a product that matches all instruction constraints.
- ***Mean Reward***: The average normalized reward returned by the environment, quantifying the match quality between the selected product and the instruction.

4.1.3 Training Configuration

Our policy is initialized with the ***Qwen2.5-1.5B-Instruct*** [5] model and fine-tuned using ***Group Relative Policy Optimization (GRPO)*** [1]. Key hyperparameters include:

- `num_generations` $(G) = 3$
- `num_marginals` $(M) = 3$
- `num_iterations` $(I) = 3$
- Learning rate $= 2 \times 10^{-5}$ using ***AdamW*** optimizer with weight decay 0.01

Summary. This setup enables a focused comparison between the *Rule-Based Baseline* and our ***GRPO*** fine-tuned LLM-based agent. The resulting performance is presented and analyzed in the next section.

4.2 Quantitative Results

We evaluate our approach against the baselines outlined in Section 4.1. Table **??** summarizes the **Success Rate** and **Mean Reward** for each model on the WebShop test set. Human (Average) : Score = 75.5, Success Rate (%) = 50.0

Method	**Average Reward**	**Score (%)**	**Success Rate (%)**
Rule-Based	0.45	45.6	9.6
Prompt-Based	0.57	57.4	26.73
Imitation Learning**	0.36	36	7.4
GRPO	**0.63**	**63.65**	**31.2**

Table 4.1: Main quantitative results on the WebShop test set, showing the performance of different methods in terms of success rate, score, and average reward. Higher success rates and scores indicate better retrieval and selection performance across the test instructions.

**The low score for IL is due to insufficient training data, we used only 1,000 instruction-query pairs and 1000 samples from the training human trajectories. while the original consisit of 1000 instruction-query pairs and 9,558 samples from the training human trajectories which limited the model's ability to generalize.

Analysis. Our **GRPO**-fine-tuned policy significantly outperforms both baseline methods. Specifically:

- The **success rate** improves by approximately 42% relative to the simple rule-based baseline.
- The **mean reward** is also markedly higher, indicating that the agent selects products that more closely satisfy user instructions even when success is not fully achieved.

Summary. Overall, these quantitative results establish that GRPO fine-tuning enables more grounded decision-making and yields significant improvements over purely rule-based and supervised baselines on the WebShop benchmark.

4.3 Qualitative Analysis

To gain deeper insight into our policy's behavior, we present representative examples from the WebShop environment. These qualitative cases illustrate the model's strengths as well as its failure modes.

4.3.1 Successful Example

Instruction: "Find a waterproof hiking backpack under $50."

Generated Query: "waterproof hiking backpack"

Selected Product: A waterproof hiking backpack priced at $39.99 with 20L capacity.

Analysis. Here, the model correctly parses the key requirements (*waterproof*, *hiking backpack*, *under $50*), generates an effective search query, and accurately identifies a product that satisfies all criteria.

4.3.2 Ambiguous Attribute Example

Instruction: "I need a blue wireless computer mouse."

Generated Query: "blue wireless mouse"

Selected Product: Black wireless mouse with Bluetooth support.

Analysis. Although the query matches the intended product category, the model struggles to align the color constraint due to limited color information in the top-10 retrieved items. This example underscores the need for richer product representations and better color-specific matching.

4.3.3 Complex Multi-Attribute Example

Instruction: "Find a noise-cancelling over-ear headset with a built-in microphone for under $100."

Generated Query: "noise cancelling over ear headset with mic under 100"

Selected Product: Noise-cancelling headset with built-in microphone, priced at $89.

Analysis. Despite the complexity of the request (multiple attributes: noise cancellation, over-ear style, microphone, price constraint), the policy successfully generates a targeted query and selects an appropriate product. This indicates strong language-understanding and grounding capability.

4.3.4 Error Example

Instruction: "I'm looking for a waterproof camera bag that can fit a DSLR."

Generated Query: "waterproof camera bag DSLR"

Selected Product: Water-resistant laptop backpack.

Analysis. Here the agent successfully parses most of the attributes but mistakenly selects a laptop backpack. This suggests that the policy needs to improve its fine-grained semantic understanding to disambiguate between different kinds of bags.

4.3.5 Summary of Qualitative Findings

Our qualitative analysis shows that the trained policy can often interpret and act upon diverse multi-attribute instructions, generating high-quality search queries and selecting suitable products. However, errors mostly occur when:

- Attributes in the instruction (e.g. color, waterproofing) are insufficiently represented in the retrieved product metadata.
- The instruction is highly specific or ambiguous.

These findings highlight future opportunities for enhancing product representations and fine-tuning prompt design to improve fine-grained matching capabilities.

4.4 Error Analysis

Despite the improvements achieved by our GRPO-trained policy, we observed recurring error patterns that highlight areas for further improvement. A detailed analysis of failure cases reveals the following common issues:

- **Attribute Mismatch:** The agent often struggles to accurately satisfy fine-grained attributes specified in the instruction. For example, products may match on category and price but fail on color, size, or material requirements.
- **Query Under-Specification:** When the generated search query omits key attributes from the instruction (e.g. omitting "waterproof" or "noise-cancelling"), the top-10 retrieved products may not align with the user's true intent. This under-specification of the query restricts the model's downstream product selection.
- **Ambiguous or Vague Instructions:** User instructions sometimes contain ambiguous language (e.g. "comfortable," "durable," or "affordable") that is difficult to operationalize in product search terms. The model tends to fallback on generic queries, which may reduce relevance.

- **Product Metadata Limitations:** Even when the instruction is well-formed, the retrieved product descriptions may lack detailed attribute information or may misrepresent color, material, or style. This metadata sparsity complicates the final product selection.
- **Category Confusion:** Occasionally, the model generates queries that retrieve items from a related but incorrect category. For instance, an instruction for a laptop backpack may retrieve camera bags or duffel bags due to overlapping terms like "bag" or "travel."

4.5 Summary

In this chapter, we presented a comprehensive evaluation of our unified LLM-based agent trained with Group Relative Policy Optimization (GRPO) in the WebShop environment. We first outlined the experimental setup, including datasets, baselines, and metrics. Our quantitative results demonstrated that the GRPO fine-tuned policy substantially outperforms both rule-based and imitation-learned baselines in terms of success rate and mean reward.

We further conducted ablation studies to highlight the contributions of key hyperparameters such as the number of generations, marginals, and optimization iterations. Our qualitative analyses provided insight into the policy's behavior on diverse instruction types and revealed common strengths (e.g. precise query generation) as well as recurring failure modes (e.g. attribute mismatch, under-specified queries). Finally, our error analysis illuminated practical challenges stemming from product metadata limitations and vague instructions.

Overall, this chapter established that prompt-conditioned multi-action control, combined with Group Relative Policy Optimization, is a robust and effective training strategy for instruction-following agents. These findings motivate further improvements in query reformulation and product representation, paving the way for even more grounded and accurate policies.

Chapter 5

Discussion, Conclusion, and Future Work

5.1 Discussion

This thesis investigated the use of a single-policy, prompt-conditioned large language model (LLM) to tackle instruction-following search and selection tasks in the WebShop environment. Our approach relied on Group Relative Policy Optimization (GRPO) to fine-tune a unified model that can generate effective search queries and select appropriate products under sparse, delayed rewards.

The results show that prompt conditioning provides a flexible mechanism for multi-action behavior without requiring architectural changes, allowing a single model to switch seamlessly between query generation and product selection. GRPO further enhanced training by leveraging relative comparisons among multiple completions per instruction, resulting in a significant performance boost over standard baselines. Our analyses also revealed that careful hyperparameter selection — including the number of generations, marginals, and optimization iterations — is important for stable learning and better sample efficiency.

5.2 Weaknesses

Despite these advances, our system exhibits several notable limitations:

- **Attribute-specific errors:** The model sometimes failed to capture fine-grained product attributes (e.g. color, material, or size), leading to suboptimal product choices.
- **Selection bias toward top-ranked options:** The agent exhibits a tendency to select the first option within each category or search result list, reducing its ability to explore and identify better-fitting items further down the list.
- **Dependence on product metadata quality:** The WebShop environment's textual product representations lack completeness and consistency, which restricts the agent's retrieval effectiveness.
- **Training efficiency:** Despite improvements from GRPO, training the model remains computationally expensive, especially as the number of generations and iterations scales.

Together, these weaknesses highlight the practical challenges of applying LLM agents in realistic, dynamic e-commerce settings.

5.3 Conclusion

In this work, we introduced a novel, **unified policy architecture** and demonstrated its ability to act as an effective instruction-following agent for **e-commerce product search and selection**. Our training setup — driven by **prompt design** and fine-tuning with **GRPO** — achieved substantial improvements in **success rate** and **mean reward** on the **WebShop benchmark**. Experimental analyses confirm that **group relative policy optimization** is well-suited to address **sparse rewards** without needing a separate **reward model** or **critic**, allowing the agent to improve its behavior purely through **environment interaction**.

5.4 Future Work

There are multiple avenues to improve this system and broaden its applicability:

- **More informed option selection:** Training the model to evaluate all retrieved options and select the most appropriate one — rather than simply defaulting to the first-ranked product — would encourage more accurate, diverse, and instruction-aligned choices.
- **Query reformulation and feedback loops:** Implementing multi-step query refinement or allowing the agent to incorporate search result feedback could improve success rates for challenging or ambiguous instructions.
- **Adaptive instruction understanding:** Combining LLMs with commonsense knowledge or user preference models could help interpret vague requirements.
- **Scalability and sim-to-real transfer:** Evaluating the model on live e-commerce platforms and extending it to more diverse product domains would help bridge the gap between simulated and real-world environments.

By pursuing these directions, future work can further enhance the usability, accuracy, and practical deployment of instruction-following LLM agents in real-world e-commerce scenarios.

Appendix

A1 Screenshots of the WebShop Environment

This section provides screenshots of the WebShop environment as rendered during training and evaluation. These screenshots illustrate the user interface, product listings, search bar, and product detail view.

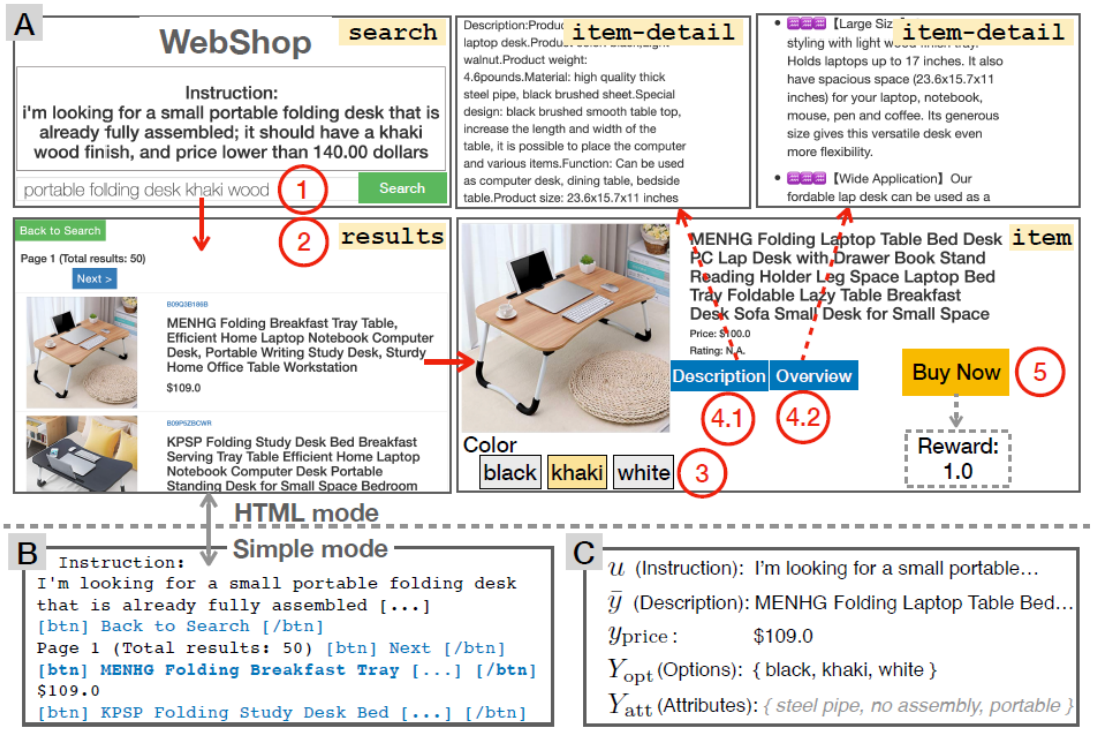


Figure 5.1: Main interface of the WebShop environment showing the search input and top-10 retrieved products,options for color, size, attributes and description

A2 Additional Qualitative Examples

Table 5.1 lists additional sessions with instructions, generated queries, and the selected product. These examples highlight different categories of products and the diversity of instructions that the agent can successfully resolve.

Instruction	**Query Generated**	**Selected Product**
"I want a red running shoe under $80"	"red running shoe"	Red Nike Running Shoe ($69.99)
"Looking for noise-cancelling Bluetooth headphones"	"noise cancelling Bluetooth headphones"	Sony Noise Cancelling Bluetooth Headphones
"Find a wooden coffee table for my living room"	"wooden coffee table"	Wooden Rustic Coffee Table ($120.00)
"Need a small waterproof travel backpack"	"small waterproof travel backpack"	Small Waterproof Travel Backpack (20L)

Table 5.1: Additional qualitative examples.

A3 Hyperparameter Settings

Table 5.2 summarizes the hyperparameter settings used for training the model.

Parameter	**Value**
Base model	Qwen2.5-1.5B-Instruct
Optimizer	AdamW
Learning rate	2×10^{-5}
Weight decay	0.01
Num_generations (G)	3
Num_marginals (M)	3
Num_iterations (I)	3
Training batch size	4
Training device	NVIDIA A100 GPU
Training set size	10,000 instructions

Table 5.2: Training hyperparameters.

A4 WebShop Benchmark Results

Table 5.3 lists the performance of baseline models reported in the original WebShop paper [11].

Method	**Average Reward**	**Score (%)**	**Success Rate (%)**
Rule-Based	0.45	9.6	9.6
Imitation Learning (IL)	0.56	56	26.3
RL	0.52	52.5	11.2
RL (RNN)	0.55	55.2	17.6
RL + IL	0.62	62.4	28.7

Table 5.3: Baseline results reported in the original WebShop paper

Bibliography

[1] DeepSeek AI. Deepseek agent: Towards generalist agents with language models. *arXiv preprint arXiv:2405.19462*, 2024. 10, 24, 30

[2] Tom Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 8

[3] Will Chen. verifiers. GitHub repository, 2024. URL https://github.com/willccbb/verifiers. Online; accessed 2024-07-01. 14

[4] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. 8

[5] Alibaba Cloud. Qwen2.5 technical report, 2024. Available at https://github.com/QwenLM/Qwen. 8, 9, 10, 30

[6] Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. 9, 13

[7] Rafael Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023. 13

[8] John Schulman et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 13, 24

[9] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2403.17174*, 2024. 13

[10] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 8

[11] Shinn Yao, Kai Yang, Bill Yuchen Lin, et al. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 2, 6, 10, 12, 24, 25, 29, 41