Problem Solving Through Human-AI Preference-Based Co-operation

Abstract

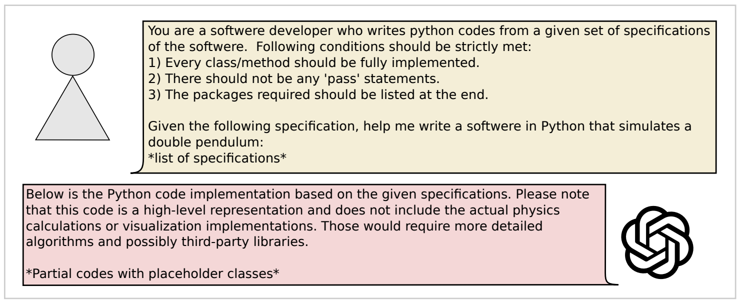
While there is a widespread belief that artificial general intelligence (AGI) – or even super-human AI – is imminent, complex problems in expert domains are far from being solved. We argue that such problems require human-AI cooperation and that the current state of the art in generative AI is unable to play the role of a reliable partner due to a multitude of shortcomings, including inability to keep track of a complex solution artifact (e.g., a software program), limited support for versatile human preference expression and lack of adapting to human preference in an interactive setting. To address these challenges, we propose `HAI-Co`^{2}, a novel human-AI co-construction framework. We formalize `HAI-Co`^{2} and discuss the difficult open research problems that it faces. Finally, we present a case study of `HAI-Co`^{2} and demonstrate its efficacy compared to monolithic generative AI models.

1 Introduction

Despite the impressive achievements of generative AI spearheaded by Large Language Models (LLMs), Vision Language Models and code models (Lozhkov et al., 2024; Wang et al., 2021), multiple recent investigations have pointed out their lack of competence in dealing with complex generation problems that require intricate planning (Kambhampati et al., 2024) or task adherence while keeping track of multiple constraints (Xie et al., 2024). A broad class of such complex problems requires active human participation. Therefore, although the recent focus in generative AI has mostly been on complete automation, we believe that human-AI cooperation is a more promising approach for complex problems of this kind.

To effectively support human-AI cooperation, we draw inspiration from how humans collectively address complex problems to devise solutions: Humans often solve complex problems by iteratively co-constructing a solution step by step, revising and refining draft solutions while transitioning between different levels of abstraction, and exchanging information about preferences and potential improvements in natural language (Damşa, 2013). *Our position is that this type of human-human cooperation is a promising template for the cooperation of humans and AI agents.* On this basis, we propose **H**uman-**AI Co-Co**nstruction (`HAI-Co`^{2}), a novel framework for human-AI cooperative problem-solving that builds on preference-based learning and search methodology and relies on natural language to facilitate interaction. The problem-solving process is conceived as a process of *systematic search* in a *construction space* $\mathcal{X}$ of *candidate solutions*, i.e., as a co-constructive process, in which candidate solutions are modified step by step until a sufficiently good solution has been found.

What are co-construction problems? The broad class of problems that we seek to address in this paper is primarily centered around domain-specific expert applications where the task is to construct a solution that meets specific requirements, e.g., a computer program in software engineering or a machine learning pipeline in automated machine learning (AutoML). We choose the term *co-construction* because of the necessity for the AI agent to work closely with human experts. First, current AI systems' capabilities are limited and cannot fully replace human expertise in these tasks, e.g., due to insufficient knowledge and reasoning capabilities, lack of trustworthiness and bias. Second, some expert tasks (e.g., designing websites and building data analysis and visualization software) require irreplaceable human supervision; solutions to be co-constructed in these problems are defined based on personalized human preference. While the co-constructed solution must fulfill a set of objective correctness criteria, the human expert imposes a broad set of subjective criteria as well. For example, in a software development setup, the final piece of software must



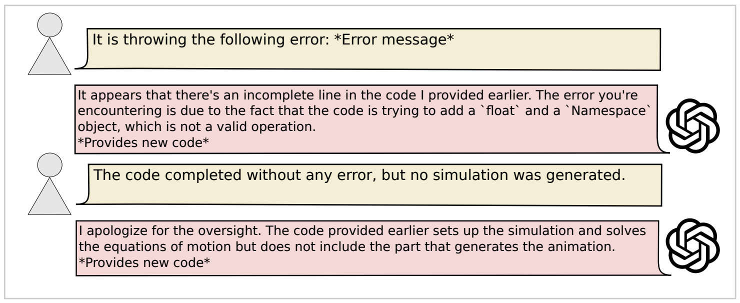
You are a softwere developer who writes python codes from a given set of specifications of the softwere. Following conditions should be strictly met:
1) Every class/method should be fully implemented.
2) There should not be any 'pass' statements.
3) The packages required should be listed at the end.

Given the following specification, help me write a softwere in Python that simulates a double pendulum:
list of specifications

Below is the Python code implementation based on the given specifications. Please note that this code is a high-level representation and does not include the actual physics calculations or visualization implementations. Those would require more detailed algorithms and possibly third-party libraries.

Partial codes with placeholder classes

(a) Fails to follow user preference



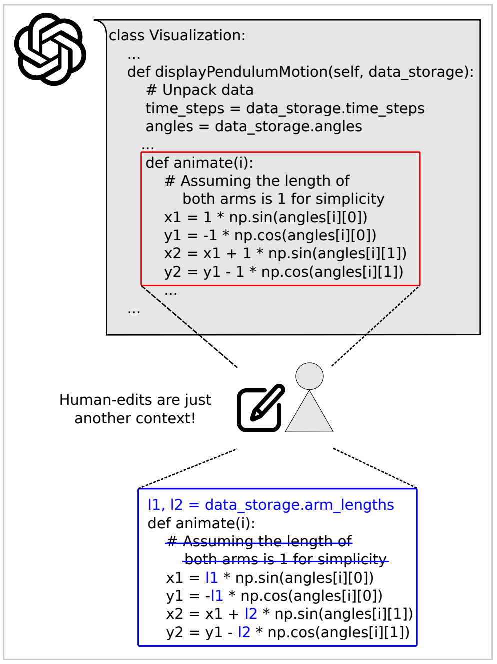
It is throwing the following error: *Error message*

It appears that there's an incomplete line in the code I provided earlier. The error you're encountering is due to the fact that the code is trying to add a `float` and a `Namespace` object, which is not a valid operation.
Provides new code

The code completed without any error, but no simulation was generated.

I apologize for the oversight. The code provided earlier sets up the simulation and solves the equations of motion but does not include the part that generates the animation.
Provides new code

(b) Unreliable refinement of modular & complex artifact



```
class Visualization:
  ...
  def displayPendulumMotion(self, data_storage):
    # Unpack data
    time_steps = data_storage.time_steps
    angles = data_storage.angles
    ...
    def animate(i):
      # Assuming the length of
        both arms is 1 for simplicity
      x1 = 1 * np.sin(angles[i][0])
      y1 = -1 * np.cos(angles[i][0])
      x2 = x1 + 1 * np.sin(angles[i][1])
      y2 = y1 - 1 * np.cos(angles[i][1])
      ...
  ...
```

```
l1, l2 = data_storage.arm_lengths
def animate(i):
  ~~# Assuming the length of~~
    ~~both arms is 1 for simplicity~~
  x1 = l1 * np.sin(angles[i][0])
  y1 = -l1 * np.cos(angles[i][0])
  x2 = x1 + l2 * np.sin(angles[i][1])
  y2 = y1 - l2 * np.cos(angles[i][1])
```

(c) Unaware of active human participation

Figure 1: Existing generative AI lacks proficiency in key aspects of co-construction of solutions to complex problems. We give a code synthesis example. (a) GPT-4 Turbo fails to follow preferences explicitly stated by the human expert. (b) Due to the lack of a persistent object representation, a modification request targeted toward one feature of the desired solution leads to the unwanted (and erroneous) modification of another feature. (c) Human expert modifies the generated code directly to remove inline assumptions and introduce general variables; such active participation is not demarcated and recorded by the AI and there is no facility to extract the implicit preference and follow it elsewhere.

be executable, secure, and bug-free; at the same time, every expert developer has their own style – of coding, commenting, modularization, etc. – that should be followed as well. This implies that the AI agent must be able to follow and adapt to the personalized preferences of the human expert (which are not necessarily stable but may evolve in the course of the problem-solving process).

Since we see expert domains as the primary setting in which co-construction excels at solving problems, we use "expert" and "human" interchangeably in this paper.

Is the current state of generative AI enough? Multiple prior investigations have laid out the inherent shortcomings of present day generative AI that limit its applicability to the type of problems we are addressing in this paper. In an example case study tailored towards code generation presented in Figure 1, we demonstrate some of the major bottlenecks of GPT-4 Turbo, a recent update[1] over the original GPT-4 model (OpenAI, 2024). In Figure 1 (a), GPT-4 Turbo ignores the human expert's explicit instructions to generate a complete Python code with the required module specifications, echoing Xie et al. (2024)'s observation that current language-based AI agents lack task adherence. After repeated prompting with partial code snippets, the process produces a complete – albeit faulty – code. This limitation becomes even more irreconcilable as more varied and realistic expressions of human preference are taken into account – for the human expert to contribute productively, one must allow preferences expressed via explicit instructions, binary choice, ranking, etc. Current generative AI solutions do not facilitate such multi-modal preference incorporation. Figure 1 (b) shows unreliable debugging attempts. Specifically, the LLM performs unrelated (and faulty) edits to address a bug and even introduces new errors. This demonstrates that existing LLMs fail to handle complex, modular software code (Jiang et al., 2024). The common practice is that the human (as

[1]We use the `gpt-4-1106-preview` instance: `https://openai.com/index/new-models-and-developer-products-announced-at-devday/`

a knowledgable expert who keeps track of overall context) identifies faulty output and repeatedly prompts the model to guide it to the correct generation – this is implicitly adopting a co-construction paradigm. However, Figure 1 (c) shows that current modes of human-AI interaction cannot unleash the full potential of co-construction – direct modification of the co-constructed candidate solution by the human expert does not bear any special significance to the LLM and it treats it as just another context. There is no explicit mechanism for the AI to learn implicit preferences expressed by the human through active participation.

While these examples are focused on code synthesis, there is evidence of similar shortcomings in other domains (Kambhampati et al., 2024; Lecler et al., 2023). Code synthesis in particular – and the experience from day-to-day use of generative AI for solving complex problems in general – point toward co-construction as a naturally evolved problem-solving paradigm where the human expert tries to search for the optimal solution by interacting with the AI. However, the current state of generative AI hinders its role as a reliable partner in successful co-construction. This is because the "one-dimensional" interaction between human and AI typical of how AI agents are used today often fails to steer the co-construction towards a solution that satisfies the user's constraints.

Our contribution. In this paper, we formalize co-constructice problem solving and thereby aim to address important limitations of current generative AI models. We present `HAI-Co`^{2}, a framework with three fundamental properties that facilitate human-AI co-construction. First, it introduces multiple levels of abstractions to the candidate solution, providing a seamless interface for the human expert and the AI agent to modify and keep track of the complex, modular co-constructed candidate solution. Second, `HAI-Co`^{2} allows multi-modal preference input from the human expert, with natural language as the central mediator to capture information-rich guidance signals, along with other forms of active expert participation such as categorical choice-based preference. Finally, `HAI-Co`^{2} introduces a search-based methodology of co-construction where the candidate solution (represented on multiple levels of abstraction) is iteratively revised to maximize the perceived utility modeled from the preference input.

The rest of the paper is organized as follows. Following a discussion of relevant literature in Section 2, we introduce the framework of `HAI-Co`^{2} in Section 3 and outline open research challenges implied by this framework in Section 4. Section 5 illustrates `HAI-Co`^{2} in a case study, prior to concluding the pape in Section 6.

2 Related work

We group relevant prior work into five major strands: reinforcement learning from human feedback, RLHF search- and evolution-driven construction, assistance games, LLM agents for complex problem solving and persistent solution spaces for iterative construction. We now discuss how they attempt to address expert-AI co-construction. However, as we will see in Section 4, they fail to comprehensively tackle its challenges.

Reinforcement Learning from Human Feedback. RLHF focuses on learning a policy preferred by humans, most commonly relying on comparisons between candidate solutions (Kaufmann et al., 2024). The goal is to learn a policy that maximizes a reward or utility function that is consistent with the human feedback. Originating in classical reinforcement learning domains such as games and continuous control (Christiano et al., 2017), RLHF has been extended to a variety of domains, most notably fine-tuning generative models such as LLMs (Stiennon et al., 2020; Ouyang et al., 2022), eventually leading to the development of AI models that can generate human-preferred responses in natural language such as ChatGPT.

RLHF for generative AI is typically employed in a single-turn setting, where the agent generates an immediate response to a query, evaluated by a human expert. This contrasts with expert-AI co-construction, which involves multi-turn interactions where agent and expert collaboratively construct a solution. Multi-turn interactions introduce challenges such as extended time horizons and large action spaces. Extensions to RLHF have been proposed that address these issues (Zhou et al., 2024).

Even multi-turn RLHF, however, is not well suited to expert-AI co-construction without further extension: It does not maintain an explicit representation of the solution space, which is crucial for systematic solution construction. In principle, RLHF could be used to learn the AI agent's policy in `HAI-Co`^{2}, but it is challenging to do so interactively as required in `HAI-Co`^{2}.

Search- and Evolution-Driven Construction. Our framework emphasizes iterative search within the construction space, a process akin to evolutionary optimization, which iteratively generates and evaluates candidate solutions (Bäck, 1996). This evolution can be viewed as a form of search-based construction. Interactive evolutionary computation, a preference-based extension, is particularly relevant to our work as it involves human evaluation of candidate solutions (Takagi, 2001; Wang & Pei, 2024). For example, these methods have been applied to search-based procedural content generation in video games (Togelius et al., 2011). Our approach differs in the core approach to the search process: Traditional evolutionary methods maintain a population of candidate solutions and generate new ones through mutation and recombination. In contrast, in our framework, each iteration ends with a single candidate solution that is then the basis for the next iteration. In addition, we leverage the extensive prior knowledge of pretrained language models to guide the search and use natural language communication to facilitate cooperation between the AI agent and the human expert.

Assistance Games. Hadfield-Menell et al. (2016) introduce the framework of cooperative inverse reinforcement learning, later called assistance games (Shah et al., 2021; Laidlaw et al., 2024), for cooperative decision-making between an AI agent and a human expert. They model human-AI interaction as a two player game betwen human expert and AI agent. The AI agent aims to maximize the expert's utility function without explicit knowledge of it. The AI agent learns this function by observing the expert's actions and receiving feedback on its own actions. Our setting is similar in that AI and human collaborate to construct a solution that only the human expert can evaluate. It differs in that we focus on search-based co-construction of solutions rather than decision-making in a cooperative setting. When viewing solution construction as a sequential decision-making problem, our framework can be seen as a search-based approach to assistance games where the AI agent's actions are the steps taken to construct the solution. Despite the nice theoretical properties of the assistance games framework (Shah et al., 2021), it has not yet been widely applied to real-world problems, likely due to the complexity of the human-AI interaction and the difficulty of interactively modeling the human expert's utility function from various sources of feedback. We hope that our framework can provide a more practical approach to human-AI co-construction by focusing on the specific challenges of solution construction and leveraging the extensive prior knowledge of pretrained language models to guide the search process.

LLM agents for complex problem solving. The rapid increase in the capabilities of LLMs has triggered multiple recent efforts to integrate them at the core of autonomous agents that interact with the environment, plan, and act to solve complex problems (Wang et al., 2024). Typical approaches adopt integrating different tool-usage capabilities into LLMs via efficient prompting, often with multimodal capabilities (Chen et al., 2023). A single agent is often insufficient to solve complex problems; thus, multiple agents with different capabilities have to be integrated. Recent efforts in LLM-based multi-agent systems seek to mimic such cooperative problem-solving by role-playing LLMs via in-context examples (Li et al., 2023) or fine-tuning (Juneja et al., 2024). Despite some promising achievements, such agentic ecosystems are inherently limited by the constituent LLMs' inability to plan and execute tasks (Xie et al., 2024; Kambhampati et al., 2024). These frameworks of LLM-based autonomous agents are largely designed for autonomous problem solving, not for human-AI co-construction as we envision. Recently, arguments in favor of strategically allocating tasks between humans and LLM-based agents to exploit their distinct strengths have been put forward (He et al., 2024), which aligns with our approach of leveraging the strengths of both humans and AI agents in co-construction. An alternative view of our framework is hence an extension of LLM-based multi-agent systems with a human agent as a key component, focusing on the co-construction of solutions.

Persistent Solution Space for Iterative Construction. A fundamental component of our proposed framework is the explicit representation of the construction space for systematic solution search. Such a persistent memory can be useful for LLM agents. Sumers et al. (2024) propose a cognitive architecture for language agents that connects LLMs to internal memory and external environments, grounding them in existing knowledge or external observations. Similarly, Modarressi et al. (2024) introduce a structured memory component that LLM agents can use for storage and retrieval. Although these approaches do not directly address expert-AI co-construction challenges, they relate to our approach by providing agents with persistent memory to store intermediate solutions and relevant information for problem solving.

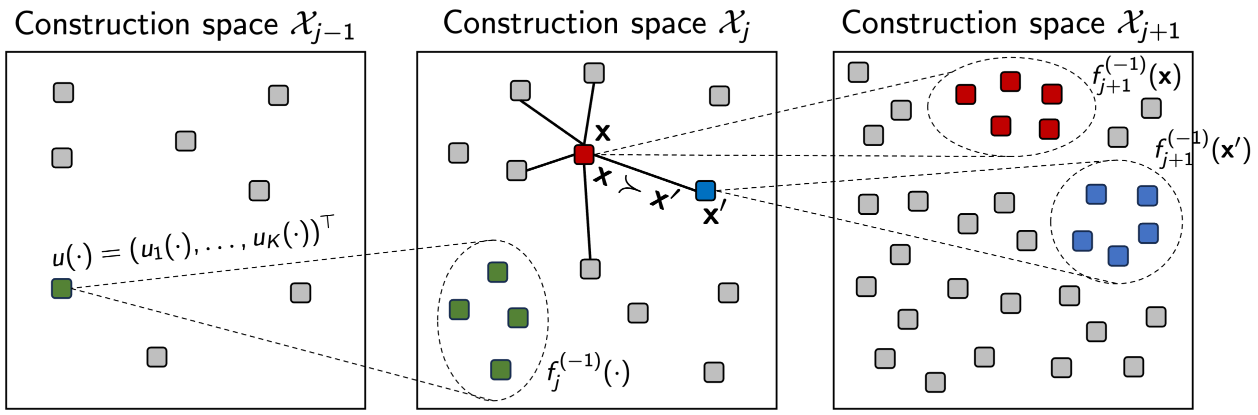

Construction space $\mathcal{X}_{j-1}$
Construction space $\mathcal{X}_j$
Construction space $\mathcal{X}_{j+1}$
$u(\cdot) = (u_1(\cdot), \ldots, u_K(\cdot))^\top$
$\mathbf{x}$
$\boldsymbol{x} \succ \boldsymbol{x}'$
$\mathbf{x}'$
$f_j^{(-1)}(\cdot)$
$f_{j+1}^{(-1)}(\mathbf{x})$
$f_{j+1}^{(-1)}(\mathbf{x}')$

Figure 2: Illustration of the hierarchy of construction spaces in HAI-Co^2. Each point $\boldsymbol{x}$ symbolizes a candidate solution (on a certain level of abstraction), e.g., a software program. The topology of the space is specified by a suitable neighborhood structure (as illustrated for point $\boldsymbol{x}$). Each point is associated with a latent utility u, possibly multi-dimensional and comprised of local utilities $u_1, \ldots, u_K$, and preferential information (e.g., $\boldsymbol{x} \succ \boldsymbol{x}'$: solution $\boldsymbol{x}$ is better than $\boldsymbol{x}'$) that provides information about promising regions in the space. The relationship between the different abstraction levels is specified by the abstraction mappings f_j resp. the (inverse) refinement mappings $f_j^{(-1)}$.

3 HAI-Co^2: Human-AI co-construction through preference-based search

In this section, we propose HAI-Co^2, a framework for cooperative problem solving. Broadly speaking, HAI-Co^2 is meant to formalize an interactive problem-solving scenario, in which a human expert seeks to (co-)construct a *candidate solution* – such as a computer program – with the help of an AI agent. The problem-solving process is conceived as a process of *systematic search* in a space $\mathcal{X}$ of *candidate solutions*, i.e., as a (co-)constructive process, in which candidate solutions are modified or extended step by step until a (sufficiently) good solution has been found. Therefore, we also refer to the search space $\mathcal{X}$ as the *construction space*. The construction space, its hierarchical organization and its topology (or neighborhood structure) are depicted in Figure 2.

Actions taken by the AI agent during the search (e.g., adapting a candidate solution or asking the expert a question regarding where to move next) depend on its *informational state* $\mathcal{I}$, which comprises its experience so far, e.g., about the expert's preferences, any relevant information about the current context, the solutions considered so far and the best solution constructed so far. Formally, the behavior of the AI agent can be determined by a *policy* π that maps informational states to actions.

3.1 Construction space and abstraction hierarchy

The construction space will typically be large, most often even (countably) infinite. For example, the construction space may consist of all computer programs in a specific programming language. Spaces of this kind cannot be specified in an explicit way. Instead, they will be defined implicitly and may even be adapted or designed on-the-fly in the course of the problem-solving process. In this regard, the *formal representation* of candidate solutions is of major importance and will strongly influence the efficacy and efficiency of the problem-solving process. Moreover, it is also clear that the representation of solutions will not be universal but rather specific to the expert domain. For example, a computer program will not be represented in the same way as a machine learning pipeline or data science workflow. It should be noted that we do not make any assumption of *completeness* for candidate solutions: at any stage of the search, a candidate solution $\boldsymbol{x} \in \mathcal{X}$ can be partial or incomplete (i.e., an incomplete codebase, an incomplete ML pipeline, etc.).

During problem solving, it is often useful to look at (candidate) solutions on multiple levels of abstraction. In many cases, for example, a rough draft of the solution is found in a first phase of the process, and this draft is then worked out in more detail in a second phase. More generally, one can imagine a search process that switches back and forth between different levels of abstraction whenever appropriate. Therefore, we assume the construction space $\mathcal{X}$ is equipped with a hierarchy of abstraction levels. Formally, this can be modeled by a sequence $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_J$ of spaces, where $\mathcal{X}_j$ is a refinement of $\mathcal{X}_{j-1}$ – or, vice versa, $\mathcal{X}_{j-1}$ an abstraction of $\mathcal{X}_j$.

We describe the abstraction process from $\mathcal{X}_j$ to $\mathcal{X}_{j-1}$ as a surjection $f_j : \mathcal{X}_j \to \mathcal{X}_{j-1}$ such that $\boldsymbol{x}' = f_j(\boldsymbol{x}) \in \mathcal{X}_{j-1}$; that is, $\boldsymbol{x}'$ is the abstraction of $\boldsymbol{x}$ on the abstraction level modeled by $\mathcal{X}_{j-1}$. We denote by $f_j^{(-1)}(\boldsymbol{x}') = \{\boldsymbol{x} \in \mathcal{X}_j \,|\, f_j(\boldsymbol{x}) = \boldsymbol{x}'\}$ the set of all refinements of $\boldsymbol{x}' \in \mathcal{X}_{j-1}$ on abstraction level $\mathcal{X}_j$. Note that refinements are not unique, which is why a transition from $\mathcal{X}_{j-1}$ to $\mathcal{X}_j$ may come with a certain arbitrariness. In our case study, we work with three levels of abstraction, considering a Python program as a refinement of a UML diagram, which in turn is a refinement of a natural language description.

3.2 Latent utility

We assume that the construction space is equipped with a latent *utility function* reflecting the preferences of the expert, i.e., the quality of solutions as perceived by the expert. In general, "quality" may refer to various dimensions or criteria, and different objectives might be pursued at the same time; we formalize this with a *multi-dimensional* utility function $u(\boldsymbol{x}) = (u_1(\boldsymbol{x}), \ldots, u_K(\boldsymbol{x}))^\top$ comprised of local utility functions u_i. For example, a computer program could be rated by average runtime or memory consumption. The local utility functions can be combined into a scalar utility function $U : \mathcal{X} \to \mathbb{R}$ via a suitable aggregation operator.

Various factors influencing the quality of candidate solutions can be distinguished, notably hard and soft constraints. *Hard constraints* refer to (functional) properties that qualify a candidate as a valid solution. For example, a computer program should properly compile and not contain any syntax errors. Even if invalid solutions should normally be considered useless, the abstract notion of utility is flexible enough to distinguish different levels of invalidity. For example, a non-executable computer program may still have a non-zero utility if the error can easily be fixed by the expert. In any case, hard constraints will normally not identify a solution in a unique way. For example, there are many computer programs that are functionally equivalent in the sense of having the same input-output behavior. *Soft constraints* refer to criteria that make a solution more or less desirable such as the length of a computer program and its time and memory consumption.

In general, the utility (be it in the form of the multi-dimensional utility function u or the scalar utility function U) is not known to the AI, nor is the expert explicitly aware of it. Rather, this utility is latent and underlies the expert's preference feedback. From this preference feedback, the AI can learn an approximation $\hat{U}$. The AI's goal is then to construct a solution $\boldsymbol{x}^*$ that maximizes $\hat{U}$, or which is at least close to the maximizer, while simultaneously improving the approximation quality of $\hat{U}$. The utility U also induces utilities on higher levels of abstraction. For example, one way to "lift" a utility function from level $\mathcal{X}_j$ to the more abstract level $\mathcal{X}_{j-1}$ is via aggregation: $U(\boldsymbol{x}') = \alpha(\{U(\boldsymbol{x}) \,|\, \boldsymbol{x} \in \mathcal{X}_j, f_j(\boldsymbol{x}) = \boldsymbol{x}'\})$, where α is an appropriate aggregation function (Grabisch et al., 2009).

3.3 Interaction and preference-based search

Search through the construction space is guided by an underlying *search strategy* – in principle, any heuristic search method (properly balancing exploration of the construction space and exploitation of acquired knowledge) may serve as a point of departure. However, in `HAI-Co`^{2}, the search is also interactive and largely controlled by the human-AI cooperation.

To guide the search, human and AI can communicate via natural language; e.g., the AI agent may ask the expert for feedback or explicit advice. Alternatively, the expert may actively intervene, for example by critiquing or modifying a candidate solution. A third type of interaction, particularly important in the context of `HAI-Co`^{2}, is driven through *preferential feedback*: By informing the AI agent about the quality of candidate solutions, the expert provides hints at presumably more promising (and, likewise, less promising) regions of the construction space, and hence suggests promising "search directions" to the AI agent. To give an example from our case study (Section 5), the expert compares two competing candidate solutions with each other (e.g., whether a modification has improved a solution or made it worse) and provides this feedback (in natural language) to the AI agent for the next iteration. The AI agent utilizes the feedback to improve its approximation $\hat{U}$ of the latent utility function, which is an important element of its informational state.

The way in which the AI agent and the human expert cooperate with each other is defined in the form of a *protocol.* Among other things, the protocol clarifies the type of queries and responses on the two sides (AI agent and human expert) and the (preference) feedback that can be given by the expert.

In summary, the specification of a concrete *instantiation of* `HAI-Co`^{2} includes the following elements:

- (Hierarchical) representation of candidate solutions (domain-specific)
- Structure of construction space $\mathcal{X}$, refinement/abstraction mappings, neighborhood structure
- Search operators (for modification of candidate solutions, refinement, abstraction, etc.) and strategy
- Natural language methods and protocol for cooperation
- Representation of informational states, the AI agent's action space and policy
- Utility: soft/hard constraints, preference relations/predicates (i.e., what type of preferences can in general be expressed, and in which form)

While some of these components can be specified by hand, others could be subject to (machine) learning and data-driven adaptation.

`HAI-Co`^{2} comprises a broad variety of human-AI interaction in natural language as well as categorical choices and active modification of the candidate solution by the human expert. The search policy π is designed to generate a (locally) optimal candidate solution based on the immediate as well as historical feedback, thereby adapting to the preference signals from the human expert user. The hierarchical abstraction of the search space facilitates a modular modification of the complex candidate solution. As we will see in the case study (Section 5), `HAI-Co`^{2} also allows for incorporating creative components into the generation of candidate solutions, for example through the injection of randomness in the heuristic search process or the refinement of abstract into more concrete solutions.

4 Challenges and practical issues

Our characterization of `HAI-Co`^{2} implies multiple challenges that need to be addressed to realize co-construction effectively. In the following, we briefly describe these challenges with reference to the current state of research.

Specification of abstraction hierarchy. Wwo core components of `HAI-Co`^{2} are (i) the abstraction hierarchy of the construction space and (ii) the neighborhood structure that facilitates preference-based search. A synergistic implementation of (i) and (ii) poses a non-trivial research challenge. In the domain of code generation, Le et al. (2024) propose a modular code generation approach to circumvent this challenge: they generate a chain-of-thought style intermediate description of the subtasks followed by modular codes implementing each of them. Such a hierarchical generation approach can be extended to generalized solution co-construction. However, relying on purely natural language-based intermediate representations limits the utility of the hierarchy. The choice of abstraction representation is domain-specific: UML descriptions are a suitable abstraction for code generation, but not for an AI scientific assistant. The abstraction specification must adhere to the neighborhood structure on all levels of abstraction; e.g., if two points are neighbors, their abstractions must also be neighbors. This poses additional constraints on the choice of abstraction, the choice of neighborhood structure, and the choice of refinements between two given levels of abstraction. Ideally, all levels of abstraction must be suitable for human preference expression: an "unnatural" choice of abstraction can render the co-construction tiresome for the human expert, negatively affecting productivity.

Specification of informational state. `HAI-Co`^{2} utilizes an informational state to keep track of relevant information in the interaction history. Given that the search policy is conditioned on it, an efficient representation of the informational state is a core challenge of `HAI-Co`^{2}. Typically such interactive co-constructions are expected to span a long sequence context. While we have observed a significant surge in the context-size of present-day generative AI (e.g., GPT-4 Turbo can handle up to 128K tokens in the input prompt), recent research has questioned the effective usability of such very long context information (Liu et al., 2024). The representation of the informational state needs to be compatible with the abstraction specification of the construction space as well as well as the choice of how preference signals from the human expert are encoded. This is particularly important as the reflection of any preference signal upon the candidate solution is manifested via the informational state – an unreliable update of the informational state subsequently worsens the solution quality and may result in a divergent search.

Communication in natural language. When humans co-construct a solution, communication in natural language plays an important role. Natural language is a powerful and at the same time succinct medium for conveying information. Given the expressivity of natural language, human and AI agent can easily communicate different options of how to improve the current solution, both at a detailed level and in more abstract terms. Similarly, preference learning is facilitated by natural language since many preferences are easily specified in natural language. The challenge here is that the language capabilities of LLMs have advanced to an impressive level for the general domain, but this does not apply to complex expert domains (Magesh et al., 2024; Hager et al., 2024; Anand et al., 2024).

Multimodal human-AI interaction. Natural language-based interaction is not the ideal channel for all types of preference. Categorical preference can be communicated more simply by pointing towards the better solution. Thus, we would like to incorporate multiple types of preference into `HAI-Co`^{2}. This poses the challenge of aligning these multiple modes of feedback with each other. For example, the human expert may express the need for a security feature in a software engineering problem explicitly, or they can express it implicitly by choosing a candidate solution that includes the feature over one that does not. The AI needs to extract equivalent preference information in these two scenarios. Contemporary research in recommender systems that deal with modeling user preferences on multiple item modalities (Guo et al., 2018; Xu et al., 2021) can serve as a starting point. However, the relative complexity and nuances of preference in the case of `HAI-Co`^{2} hinder a trivial extension of recommendation-oriented solutions.

Dynamic user preference. Current techniques of aligning neural AI systems to human preferences, broadly referred to as RLHF (Reinforcement Learning from Human Feedback), typically involve a two-stage process: learning a reward model on preference data followed by fine-tuning a foundation model (often an LLM or a diffusion model) upon reward supervision from the reward model (Kaufmann et al., 2024). This setup is fundamentally limited to static adaptation in the regime of expert-domain co-construction; a single model of human preference is imitated by the agent that cannot adapt to the personalized preference of the human expert. In practice, user preferences are dynamic and evolve over time and are, as discussed by Lichtenstein & Slovic (2006), commonly constructed during elicitation and as such highly influenced by the elicitation process. This is a fundamental challenge in co-construction problems, where the AI agent must adapt to the evolving preference of the human expert. Multi-turn RLHF (Zhou et al., 2024), though it extends the context of preference-adherence to an iterative, conversational regime, cannot solve the challenge of dynamically evolving user preference. The PAL framework, proposed by Chen et al. (2024), provides a partial solution to our problem via personalized modeling of static human preferences. Unlike traditional policy learning, `HAI-Co`^{2} motivates a reward-free exploration of the solution space (Jin et al., 2020). In-context reinforcement learning can pave the way towards handling dynamic preference signals (Yang et al., 2024; Lee et al., 2024). However, the action space in the scope of `HAI-Co`^{2} overlaps with the generation of multiple hierarchical views of the candidate solution, rendering the problem much harder than existing work on in-context RL. Prior work with LLMs showcases the possibilities of using them as in-context agents, though exploration abilities will need fine-tuning-based interventions (Krishnamurthy et al., 2024).

Creativity-correctness dilemma. The specific class of co-construction problems that we seek to address requires creative generation. At the same time, in most expert-domain applications, the solution needs to fulfill objective correctness criteria. With generative models, the two requirements of creativity and correctness become counteractive. Creative generation typically emerges in highly stochastic regimes (e.g., high temperature decoding) (Wang et al., 2023a). However, increased stochasticity carries the risk of hallucination (Aithal et al., 2024). For problems with definite answers, it has already been shown that more robust reasoning can be achieved by stochastic exploration of the generation space and identifying the subset of solutions that are most consistent (Wang et al., 2023b). However, such self-consistency methods are limited to problem classes with definitive answers and cannot be readily applied to the co-construction problems that we characterize in this paper. In `HAI-Co`^{2}, this can be generalized into a broader learning problem of exploration-exploitation trade-off. In the early iterations of co-construction, when the preference input from the human expert is likely to be vague, the AI may bias towards exploration of the construction space in search for a creative solution backbone. As the co-construction proceeds, the human expert fixates on the feature requirements and the AI must refrain from abrupt modifications and build on the preference model developed from the early exploration.

Evaluation of co-construction techniques. Due to the personalized and dynamic preferences of the expert and the complexity and modularity of the solution, the evaluation of co-construction is a difficult challenge. We identify multiple dimensions of evaluation that need to be addressed:

- *Quality of the solution* should be evaluated using domain-specific measures; irrespective of the process of co-construction or human preference, the solution must fulfill some objective criteria of correctness.
- *Preference-adherence* is an essential criterion of the co-construction problem; across multi-iteration co-construction, the generation should closely follow what the human expert asks of it.
- *Self-consistency* is another key aspect of `HAI-Co`^{2}, as it allows multiple levels of abstraction along with multiple modes of human preference input; it is essential to quantize how consistently the hierarchical abstraction is represented and different modes of preference input are aligned.
- *Complexity* of co-construction includes the computational complexity of generation and preference-based search – resulting in potentially high computational cost – and the cognitive complexity of the framework – resulting in cognitive load for the expert user. The latter demands significant research efforts from a multi-disciplinary approach to ensure that automated assistants truly bring value to the expert.

Given that human experts are costly and have limited time, LLM-based simulation of human-AI interaction may facilitate large-scale evaluation (Tamoyan et al., 2024). Even though many of our four evaluation criteria seem to demand human evaluation, we conjecture that the development of artificial critic models (McAleese et al., 2024), with human value alignment, will be an important research direction in the future.

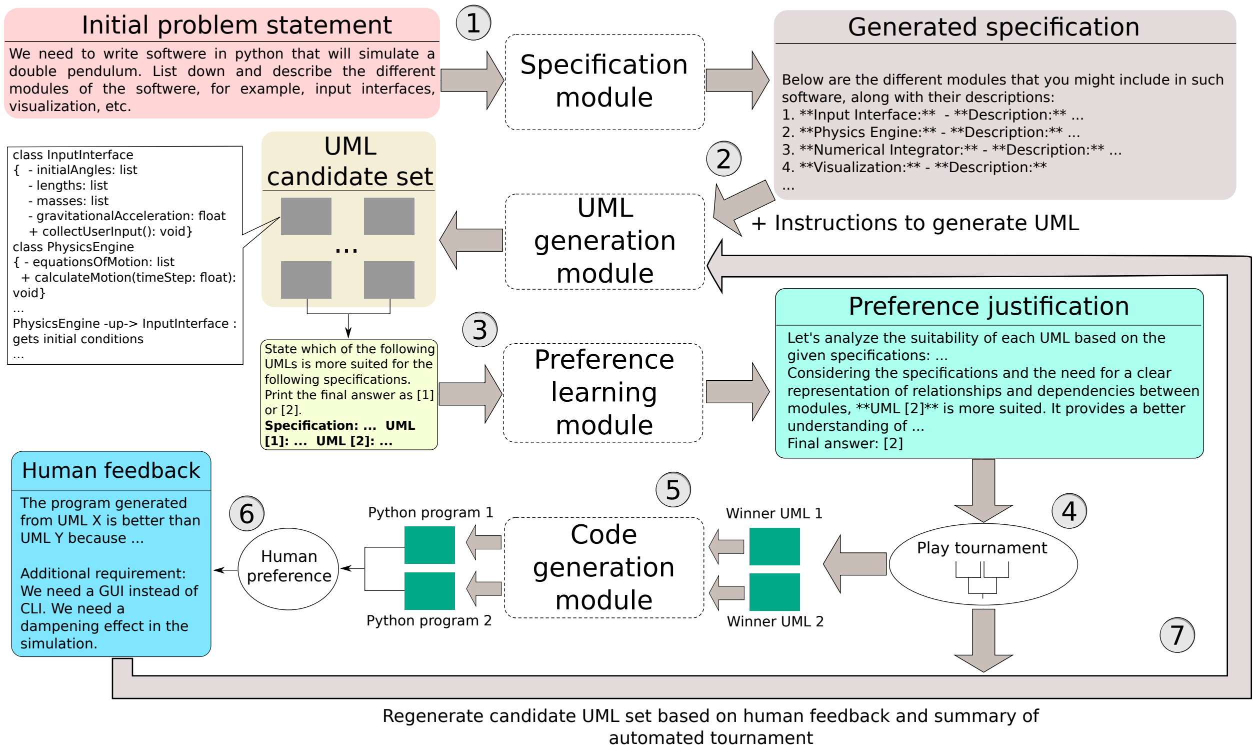
5 An exemplary implementation of `HAI-Co`^{2}

In this section, we provide a case study of `HAI-Co`^{2}, tailored to code generation as a co-construction problem. This case study does not claim scientific rigor on its own; instead, we use it to demonstrate what prior findings (see Section 1 and Section 2) already establish.

The initial problem description is underspecified. During the cooperation, the user can introduce new requirements, ask for modifications to the already generated code, and so on. We approximate different aspects of `HAI-Co`^{2} (the surjective mappings between different abstraction hierarchies of the construction space, policy and heuristic search strategy and the multi-dimensional utility function) using baseline implementation strategies for ease of demonstration. Future research endeavors should be directed to more in-depth implementation of these features.

Problem. The user wants to develop a modular Python codebase for simulating a double pendulum. Modules should include components such as I/O interface, visualization and physics engine.

In this example, the construction space $\mathcal{X}$ consists of the set of all Python programs. Three distinct levels of abstraction are implemented: a specification of the simulation software in natural language ($\mathcal{X}_0$), a UML description of the software ($\mathcal{X}_1$), and finally, the Python program ($\mathcal{X}_2$) itself. The abstraction refinements $f_1^{(-1)}$ and $f_2^{(-1)}$ (as introduced in Section 3) are implemented using suitably prompted instances of GPT-4 Turbo that we denote as *UML generation module* and *Code generation module*, respectively; while the former produces a (stochastic) set of refinements in UML given a natural language specification, the latter



We need to write softwere in python that will simulate a double pendulum. List down and describe the different modules of the softwere, for example, input interfaces, visualization, etc.

Below are the different modules that you might include in such software, along with their descriptions:
1. **Input Interface:** - **Description:** ...
2. **Physics Engine:** - **Description:** ...
3. **Numerical Integrator:** - **Description:** ...
4. **Visualization:** - **Description:**
...

```
class InputInterface
{  - initialAngles: list
   - lengths: list
   - masses: list
   - gravitationalAcceleration: float
   + collectUserInput(): void}
class PhysicsEngine
{ - equationsOfMotion: list
  + calculateMotion(timeStep: float):
void}
...
PhysicsEngine -up-> InputInterface :
gets initial conditions
...
```

State which of the following UMLs is more suited for the following specifications. Print the final answer as [1] or [2].
Specification: ... UML [1]: ... UML [2]: ...

Let's analyze the suitability of each UML based on the given specifications: ...
Considering the specifications and the need for a clear representation of relationships and dependencies between modules, **UML [2]** is more suited. It provides a better understanding of ...
Final answer: [2]

The program generated from UML X is better than UML Y because ...

Additional requirement: We need a GUI instead of CLI. We need a dampening effect in the simulation.

Figure 3: Instantiation of HAI-Co^2 for the problem of building a double pendulum simulation. A solution is co-constructed through human-AI cooperation as follows. ① The specification module takes the initial problem statement as input and generates a natural language description of the software. ② The UML generation module generates multiple candidate UML descriptions from the generated specification using stochastic decoding. ③ Pairs of candidate UMLs are provided to the Preference learning module, which chooses a winner for each pair and generates a justification of its choice. We simulate a tournament ④ using the preference learning module in this pairwise manner and end up with a pair of "best" UML candidates. ⑤ The code generation module independently generates two Python programs, one for each UML candidate. Upon execution of these two candidate programs, the expert user provides their feedback ⑥ by choosing one of them as a "better" candidate, along with (optional) justification behind their choice and (optional) additional requirements. The human feedback, along with a summary of the tournament choices made by the preference learning module, are then used by the UML generation module to generate a new set of candidate UMLs.

generates Python implementations of a given UML description. The informational state $\mathcal{I}$ is realized at different levels of abstraction within the contexts of these LLMs. We do not implement a concrete realization of the policy π; instead, we rely on the limited abilities of LLM instances to explore and implement policy iterations (Krishnamurthy et al., 2024; Brooks et al., 2023).

The implementation, as depicted in Figure 3, instantiates HAI-Co^2 as follows. The co-construction starts with the user providing an underspecified description of the task (in this example, building a simulation software in Python). The specification module (① in Figure 3) generates the natural language abstraction of the candidate solution as a list of possible components of the software along with their functionalities [2]. This serves as a transparent interface in natural language that provides a layout of the construction. The user can directly edit the specification if they have specific requirements in mind (*preemptive reviewer*), or choose to continue with the workflow and decide on the specifics upon observing the final program (*lazy reviewer*).

[2]See `https://anonymous.4open.science/w/ExAIC-Interactions-10A2/SpecificationLayer.html` for the details

Next, the UML generation module generates a set of stochastic refinements of the natural language specification into UML descriptions (② in Figure 3). The UML description of the software forces the subsequent code generation module to generate a final program that consists of multiple, independent components (in this case, Python classes) and well-defined dependencies among such components. Micro-level changes to the code (e.g., changing the design of the GUI, choice of numerical algorithms in the simulator, etc.) can be facilitated now without changing the complete codebase – a desirable property of our implementation that is closer to real-life software engineering. This addresses the challenge monolithic code LLMs face in scenarios in which persistent editing is required.

To facilitate exploration of the candidate solution space, we generate multiple UML descriptions from a given specification by setting a high decoding temperature in the UML generation module and sampling multiple responses [3]. Intuitively, we seek to exploit earlier findings that a highly stochastic generation regime facilitates better novelty (Wang et al., 2023a). However, generating the Python programs from all such candidate UMLs and verifying them one by one is both computationally expensive and infeasible for the human user.

To facilitate partial automation of searching through the candidate UMLs, we simulate human preference using GPT-4 Turbo in the preference learning module (③ in Figure 3). Specifically, the task of the preference learning module is to provide judgment on the relative quality of a given pair of UML candidates along with a justification of the preference. We simulate a tournament (④ in Figure 3) among the candidate UMLs by iteratively declaring one among a pair as the winner of a round. After a logarithmic order of such rounds ($\log_2 n$ being the depth of the tournament tree for n candidate UMLs), the preference learning layer comes up with a final pair and a summary of preference justifications [4]. Note that although we seek to minimize the cost of human intervention in this step by automating preference-based ranking, one can envision the human expert providing their judgement on these UMLs. In such a setup, the preference-learning module needs to explicitly adapt to such gold preference examples.

Next, we utilize the code generation module to translate each of the two selected UML candidates into a candidate Python program (⑤ in Figure 3) that will be used for human feedback post-execution [5]. Aligned to the goal of co-construction, in this last stage, the user provides their binary judgment on the relative quality of the two generated Python programs along with (optionally) natural language feedback. Such feedback can incorporate the errors found in the program execution (if any), additional requirements, etc. This feedback, along with the summary of the tournament generated by the preference learning module, are together used as a context for the UML generation module to generate a new set of candidate UMLs. This iterative process continues until the user is satisfied with the solution.

Comparison with monolithic LLMs. We compare among GPT-4 Turbo [6], Meta Llama-3 80B Instruct [7], and `HAI-Co`^{2} towards this case study of developing a double pendulum simulation software in Python. We observe multiple immediate improvements with `HAI-Co`^{2} over monolithic LLMs that we summarize as follows:

- **Modularity.** The incorporation of UML descriptions as interim representation naturally pushes the code generation component in `HAI-Co`^{2} towards modular code generation. Single LLMs like GPT-4 Turbo or Llama-3 70B, although able to *sometimes* follow the instruction of modular code generation, do not respect any defined structure across multiple samples of generation.

- **Ease of iterative modification.** The hierarchical abstraction further facilitates seamless iterative modification of the candidate solution across multiple passes of human feedback. Monolithic LLMs tend to introduce unrelated modifications or simply refuse to modify the code despite explicit instructions.

[3]See `https://anonymous.4open.science/w/ExAIC-Interactions-10A2/UMLLayer.html` for the 16 samples we generate in this case study

[4]See `https://anonymous.4open.science/w/ExAIC-Interactions-10A2/PreferenceLayer.html` for the tournament on the 16 UMLs generated in the first phase

[5]See `https://anonymous.4open.science/w/ExAIC-Interactions-10A2/CodeGenLayer.html` for a complete round of code generation

[6]See `https://anonymous.4open.science/w/ExAIC-Interactions-10A2/GPT-4-turbo.html`

[7]See `https://anonymous.4open.science/w/ExAIC-Interactions-10A2/Meta-Llama-3-80B-Instruct.html`

- **Preference adherence.** Although our implementation of in-context preference-based search via GPT-4 Turbo is off-the-shelf and quite simple, `HAI-Co`^{2} tends to exhibit more transparent and adaptive preference-adherence behavior whereas monolithic LLMs diverge from their initial preference-adherence as the co-construction progresses.

Limitations and further improvements. The immediate improvements we observe are prevalent without any dedicated implementation – neither of the two refinement maps (from natural language to UML and from UML to Python) nor of the preference-based search policy. In this implementation, we do not accommodate direct modification of candidate solutions by the expert as a mode of human feedback. We posit that development along these directions will further improve the quality of co-construction and confirm `HAI-Co`^{2}'s potential as an effective framework for the class of co-construction problems we aim to address. Modern software development relies on software engineering tools such as type systems, test drivers, static program analysis tools, monitoring and debugging tools and security vulnerability detectors. Realistic software artifacts are complex and their full evaluation by humans without these tools is infeasible. Our implementation of `HAI-Co`^{2}– not intended as a systematic evaluation of `HAI-Co`^{2}'s effectiveness in the software domain – will need to be extended with many of the elements that are standard in the DevOps pipeline (see Le et al. (2022); Maninger et al. (2024) for examples of how to integrate such standard tools). We use the LLM's generative capacity as is, e.g., when it explains why one generated solution is better than another. An interesting research direction would be *explanatory interactive learning* (Ross et al., 2017; Teso & Kersting, 2019; Friedrich et al., 2023) where more faithful explanations are produced through interactively constraining model explanations.

6 Conclusion

This paper presents a novel research perspective towards complex problem solving via human-AI co-construction. Our position is that existing generative AI agents require active human participation to successfully construct solutions to complex problems, but cannot effectively serve as reliable partners in human-AI cooperation due to their current limitations. We find evidence for this position in multiple prior research areas across a broad set of domains. Our case study focuses on software generation using GPT-4 Turbo, a strong proprietary LLM, and exemplifies the major drawbacks of current LLMs such as inability to follow human preference, unreliable refinement of complex solution artifacts and limitations to facilitate active human participation. We observed that although day-to-day usage of generative AI tends to adopt a type of human-AI co-construction paradigm in an uninformed manner, the challenges that LLMs face confine such interactions to a much weaker form. As a remedy, we introduce `HAI-Co`^{2}, a framework that is motivated by the effectiveness of collective human problem solving. `HAI-Co`^{2} facilitates a solution construction space with multiple levels of abstractions in which human and AI iteratively refine the candidate solution through search guided by human preference. `HAI-Co`^{2} allows active human participation along with versatility in preference expression. After presenting a formalization of `HAI-Co`^{2}, we discussed the research challenges for this new approach as well as possible future directions for addressing them. Finally, we presented a case study for the application of code generation and noted clear improvements compared to monolithic LLMs.

References

Sumukh K Aithal, Pratyush Maini, Zachary C. Lipton, and J. Zico Kolter. Understanding Hallucinations in Diffusion Models through Mode Interpolation, 2024. URL `https://arxiv.org/abs/2406.09358`.

Abhinav Anand, Shweta Verma, Krishna Narasimhan, and Mira Mezini. A Critical Study of What Code-LLMs (Do Not) Learn. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996. ISBN 978-0-19-535670-0.

Ethan Brooks, Logan A Walls, Richard Lewis, and Satinder Singh. Large Language Models can Implement Policy Iteration. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=LWxjWoBTsr`.

Daiwei Chen, Yi Chen, Aniket Rege, and Ramya Korlakai Vinayak. Pal: Pluralistic alignment framework for learning from heterogeneous preferences, 2024. URL https://arxiv.org/abs/2406.08469.

Liang Chen, Yichi Zhang, Shuhuai Ren, Haozhe Zhao, Zefan Cai, Yuchi Wang, Peiyi Wang, Tianyu Liu, and Baobao Chang. Towards End-to-End Embodied Decision Making via Multi-modal Large Language Model: Explorations with GPT4-Vision and Beyond, 2023. URL https://arxiv.org/abs/2310.02071.

Paul Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30. Curran Associates, Inc., 2017.

Crina Damşa. Knowledge Co-construction and Object-oriented Collaboration. A Study of Learning through Collaborative Construction of Knowledge Objects in Higher Education, 2013.

Felix Friedrich, Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. A typology for exploring the mitigation of shortcut behaviour. *Nature Machine Intelligence*, 2:1–12, 2023.

M. Grabisch, J.L. Marichal, R. Mesiar, and E. Pap. *Aggregation Functions*. Cambridge University Press, 2009.

Yangyang Guo, Zhiyong Cheng, Liqiang Nie, Xin-Shun Xu, and Mohan Kankanhalli. Multi-modal preference modeling for product search. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pp. 1865–1873, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356657. doi: 10.1145/3240508.3240541. URL https://doi.org/10.1145/3240508.3240541.

Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative Inverse Reinforcement Learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29. Curran Associates, Inc., 2016.

Paul Hager, Friederike Jungmann, Robbie Holland, Kunal Bhagat, Inga Hubrecht, Manuel Knauer, Jakob Vielhauer, Marcus Makowski, Rickmer Braren, Georgios Kaissis, and Daniel Rueckert. Evaluation and mitigation of the limitations of large language models in clinical decision-making. *Nature Medicine*, 2024.

Junda He, Christoph Treude, and David Lo. LLM-Based Multi-Agent Systems for Software Engineering: Vision and the Road Ahead, 2024. URL https://arxiv.org/abs/2404.04834.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A Survey on Large Language Models for Code Generation, 2024. URL https://arxiv.org/abs/2406.00515.

Chi Jin, Akshay Krishnamurthy, Max Simchowitz, and Tiancheng Yu. Reward-free exploration for reinforcement learning. In *International Conference on Machine Learning*, pp. 4870–4879. PMLR, 2020.

Gurusha Juneja, Subhabrata Dutta, and Tanmoy Chakraborty. $\mathtt{LM}^2$: A Simple Society of Language Models Solves Complex Reasoning, 2024. URL https://arxiv.org/abs/2404.02255.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Kaya Stechly, Mudit Verma, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A Survey of Reinforcement Learning from Human Feedback, 2024. URL https://arxiv.org/abs/2312.14925.

Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context?, 2024. URL https://arxiv.org/abs/2403.15371.

Cassidy Laidlaw, Eli Bronstein, Timothy Guo, Dylan Feng, Lukas Berglund, Justin Svegliato, Stuart Russell, and Anca Dragan. Scalably Solving Assistance Games. In *ICML 2024 Workshop on Models of Human Feedback for AI Alignment*, 2024.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning, 2022. URL `https://arxiv.org/abs/2207.01780`.

Hung Le, Hailin Chen, Amrita Saha, Akash Gokul, Doyen Sahoo, and Shafiq Joty. CodeChain: Towards Modular Code Generation Through Chain of Self-revisions with Representative Sub-modules. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=vYhglxSj8j`.

Augustin Lecler, Loïc Duron, and Philippe Soyer. Revolutionizing radiology with gpt-based models: Current applications, future possibilities and limitations of chatgpt. *Diagnostic and Interventional Imaging*, 104 (6):269–274, 2023. ISSN 2211-5684. doi: https://doi.org/10.1016/j.diii.2023.02.003. URL `https://www.sciencedirect.com/science/article/pii/S221156842300027X`.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=3IyL2XWDkG`.

Sarah Lichtenstein and Paul Slovic (eds.). *The Construction of Preference*. Cambridge University Press, 2006. doi: 10.1017/CBO9780511618031.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, and et al. Starcoder 2 and the stack v2: The next generation. *CoRR*, abs/2402.19173, 2024. doi: 10.48550/ARXIV.2402.19173. URL `https://doi.org/10.48550/arXiv.2402.19173`.

Varun Magesh, Faiz Surani, Matthew Dahl, Mirac Suzgun, Christopher D. Manning, and Daniel E. Ho. Hallucination-free? assessing the reliability of leading ai legal research tools, 2024. URL `https://arxiv.org/abs/2405.20362`.

Daniel Maninger, Krishna Narasimhan, and Mira Mezini. Towards Trustworthy AI Software Development Assistance. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER'24, pp. 112–116, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705007. doi: 10.1145/3639476.3639770. URL `https://doi.org/10.1145/3639476.3639770`.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. LLM Critics Help Catch LLM Bugs, 2024. URL `https://arxiv.org/abs/2407.00215`.

Ali Modarressi, Abdullatif Köksal, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Memllm: Finetuning llms to use an explicit read-write memory, 2024. URL `https://arxiv.org/abs/2404.11672`.

OpenAI. GPT-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 2662–2670, 2017. doi: 10.24963/ijcai.2017/371. URL https://doi.org/10.24963/ijcai.2017/371.

Rohin Shah, Pedro Freire, Neel Alex, Rachel Freedman, Dmitrii Krasheninnikov, Lawrence Chan, Michael D. Dennis, Pieter Abbeel, Anca Dragan, and Stuart Russell. Benefits of Assistance over Reward Learning, 2021. preprint.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33. Curran Associates, Inc., 2020.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive Architectures for Language Agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001. doi: 10.1109/5.949485.

Hovhannes Tamoyan, Hendrik Schuff, and Iryna Gurevych. LLM Roleplay: Simulating Human-Chatbot Interaction, 2024. URL https://arxiv.org/abs/2407.03974.

Stefano Teso and Kristian Kersting. Explanatory interactive machine learning. In *AAAI/ACM Conference on AI, Ethics, and Society (AIES 2019)*, pp. 239–245, 2019.

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011. doi: 10.1109/TCIAIG.2011.2148116.

Chi Wang, Susan Xueqing Liu, and Ahmed H. Awadallah. Cost-effective hyperparameter optimization for large language model generation inference, 2023a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL http://dx.doi.org/10.1007/s11704-024-40231-1.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yanan Wang and Yan Pei. A comprehensive survey on interactive evolutionary computation in the first two decades of the 21st century. *Applied Soft Computing*, pp. 111950, 2024. doi: 10.1016/j.asoc.2024.111950.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *CoRR*, abs/2109.00859, 2021. URL https://arxiv.org/abs/2109.00859.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A Benchmark for Real-World Planning with Language Agents. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

Cai Xu, Ziyu Guan, Wei Zhao, Quanzhou Wu, Meng Yan, Long Chen, and Qiguang Miao. Recommendation by users' multimodal preferences for smart city applications. *IEEE Transactions on Industrial Informatics*, 17(6):4197–4205, 2021. doi: 10.1109/TII.2020.3008923.

Rui Yang, Xiaoman Pan, Feng Luo, Shuang Qiu, Han Zhong, Dong Yu, and Jianshu Chen. Rewards-in-context: Multi-objective alignment of foundation models with dynamic preference adjustment. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 56276–56297. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/yang24q.html`.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/zhou24t.html`.