NEMO - A Neural, Emotional Architecture for Human-AI Teaming

Stefania Costantini[1,2,*,†], Pierangelo Dell'Acqua[3,†], Giovanni De Gasperis[1,†], Francesco Gullo[1,†] and Andrea Rafanelli[4,1,†]

[1]*Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy*
[2]*Gruppo Nazionale per il Calcolo Scientifico - INdAM, Roma, Italy*
[3]*Department of Science and Technology, Linköping University, Linköping, Sweden*
[4]*Department of Computer Science, University of Pisa, Italy*

Abstract

In this work, we propose a novel architecture for agents to be employed in Human-AI Teaming in various, even critical, domains based upon affective computing, empathy, and Theory of Mind, and a description of the user profile and the operational, professional, and ethical requirements of the domain in which the agent operates. In this paper, we outline the architecture's building blocks and their interconnections. The architectural design agent encompasses a Knowledge Graph to enclose the above-mentioned kinds of knowledge and a Behavior Tree enhanced via a Neural component, where the latter elaborates sensor input from devices that monitor the user and input from the knowledge graph. The enhanced behavior tree actually interacts with the user, making actions or providing suggestions, and returns feedback to feed to the knowledge graph as a novelty in the literature. We briefly present a case study, on which to experiment once the implementation, which is presently at an initial stage, will be completed. We discuss in some detail the Prolog program implementing the behavior tree, and discuss why we chose Prolog.

Keywords

Human-AI Interaction, Human-AI Teaming, Trustworthy AI, Responsible AI

1. Introduction

One recent focus in Artificial Intelligence (AI) is building intelligent systems where humans and AI systems form teams. This to exploit the potentially synergistic relationships between human and automation, thus devising "hybrid" systems where the partners should cooperate to perform complex tasks, possibly involving a high degree of risk. As a simple example, in an AI-supported self-driving or assisted-driving vehicle, the AI component can be expected to evaluate and co-manage situations and risks, where the driver can provide the AI component with useful information on practical driving in all conditions and can self-manage the risks in the case this should be required by the circumstances. Human-automation interaction is, in fact, one of the main themes of Human-centered AI. This issue also falls in the realm of Trustworthy AI, whose requirements include respect for human autonomy, prevention of harm, fairness, and explainability, and of Responsible AI, whose goal is to employ AI in a safe, trustworthy and ethical fashion.

AI and humans, if working together in Human-AI Teaming (HAIT), can produce results exceeding what either can achieve alone, whereas they can control and improve each other. For instance, a human driver might train to cope with previously unseen situations through co-driving automation via a cooperative task shared between the human driver and the AI-based system installed on the vehicle. At

CILC 2024 – 39th Italian Conference on Computational Logic, June 26–28, 2024, Rome, Italy.
*Corresponding author.
†These authors contributed equally.
stefania.costantini@univaq.it (S. Costantini); pierangelo.dellacqua@liu.se (P. Dell'Acqua); giovanni.degasperis@univaq.it (G. De Gasperis); francesco.gullo@univaq.it (F. Gullo); andrea.rafanelli@phd.unipi.it (A. Rafanelli)
https://www.disim.univaq.it/StefaniaCostantini (S. Costantini); https://dellacqua.se/ (P. Dell'Acqua); https://www.disim.univaq.it/GiovanniDeGasperis (G. De Gasperis); https://fgullo.github.io/ (F. Gullo)
0000-0002-5686-6124 (S. Costantini); 0000-0003-3780-0389 (P. Dell'Acqua); 0000-0001-9521-4711 (G. De Gasperis); 0000-0002-7052-1114 (F. Gullo); 0000-0001-8626-2121 (A. Rafanelli)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3733/paper11.pdf

the same time, AI helps drivers in case of difficulties and immediate risks. In this synergistic relationship, humans may improve automation efficacy and capabilities. At the same time, automation may enhance human performance in a task and compensate for human inadequacies, catching and correcting possible misbehaviors, possibly also due to physically or emotionally impaired states, and providing valuable suggestions.

For the tasks of adopting AI agents in crucial tasks such as, e.g., improving care-giving in medicine and teaching and constructing effective human-AI teams, agents should be endowed with an emotion recognition and management module, capable of empathy and modelling aspects of the Theory of Mind (ToM), in the sense of being able to reconstruct what someone is thinking or feeling. Modelling a Theory of Mind is often based on forms of "Affective Computing", which is a set of techniques aimed at eliciting a human's emotional condition from physical signs, to enable the system to respond intelligently to human emotional feedback.

In this work, we devise the architectural design of an agent to be employed in HAIT, based upon affective computing, empathy, and Theory of Mind, and a description of the user profile and of the operational, professional and ethical requirements of the domain in which the agent operates. Our main contribution is the design of the architecture, which includes a Knowledge Graph (KG), a Neural component, and a Behavior Tree (BT) as its main components. As such, we term the proposed framework NEMO ("Neural EMOtional").

The KG will encompass definitions concerning all the required kinds of knowledge, plus a history of past interactions of the agent with the user. The BT enhanced via a Neural component, where the latter elaborates sensor input from devices that monitor the user and input from the KG, will actually interact with the user, making actions or providing suggestions, and return feedback to feed to the KG. This two-way relationship between KG and BT constitutes a novelty in the literature.

We provide a Prolog implementation of the BT and its relationships with the other components of the proposed NEMO framework. We have chosen Prolog for our implementation for reasons of flexibility and readability apt for the task.

The paper is organized as follows. Section 2 provides some background on behavior trees and knowledge graphs. Section 3 provides the overview of the envisaged framework. The Prolog implementation of the BT component is illustrated in Section 4, and a relevant case study for the future implemented system is outlined in Section 4.1. After discussing related work and some open issues in Section 5, we conclude the paper in Section 6.

2. Background

2.1. Behavior Trees

Behavior trees were introduced as a tool to enable modular AI in computer games. A BT is essentially a mathematical model of plan execution, where each element (task and action) of a plan is associated with a node in the tree. Their strength comes from their ability to create complex tasks composed of simple tasks without worrying about how the simple functions are implemented. For a comprehensive survey of BTs in Artificial Intelligence and Robotic applications, see [1, 2]. A BT is a directed acyclic graph consisting of different types of nodes, each associated with executable code (where such code enacts an element composing a plan). In most cases, a BT is tree-shaped, hence the name. However, unlike a traditional tree, a node in a BT can have multiple parents, allowing the reuse of that part of the tree. The traversal of a behavior tree starts at the top node. When a node is traversed, the associated code is executed, returning one of the three states: *success*, *failure*, or *running*.

The critical nodes in a BT include leaf nodes and inner nodes. An *action* is a leaf node representing a behavior that the character can perform. The action returns success or failure when it completes its execution, depending on the outcome. An action is depicted as a white circle. A *condition* is a leaf node that checks an internal or external state. It returns either success or failure. A condition is represented as a grey rounded rectangle. A *sequence selector* is an inner node with several child nodes executed sequentially. Once a child node completes its execution successfully, the sequence selector continues

executing the next child node. If every child node returns success, then the sequence selector returns success. If one of the child nodes returns failure, the sequence selector immediately returns failure. A sequence selector is depicted as a grey square with an arrow across the links to its child nodes. A *priority selector* is an inner node. It has a list of child nodes that it tries to execute one at a time until one of the child nodes returns success. If none of the child nodes executes successfully, the priority selector returns failure. A priority selector is represented with a grey circle with a question mark.

2.2. Neural Empathy-Aware Behavior Trees

To consider empathy and mimic human decision-making, in [3] we introduced *neural empathy-aware behavior trees* (NEABTs) by introducing a selector node called *emotional selector*, an *empathy node*, and a *neural node*.

The emotional selector is a node that orders its child nodes based on the agent's current affective state. The agent elaborates on the affective state during repeated interactions with the user and then tunes its reaction accordingly. Once the ordering has been established, the emotional selector behaves as a priority selector. A white circle with the character E represents an emotional selector. In contrast, an empathy node provides an emotional evaluation of its single child node. An empathy node can only be a child of an emotional selector. Its child can be a leaf or an inner node. A dashed circle line with the name of the empathy emotion represents an empathy node.

To enable the integration of deep learning models for emotion recognition and symbolic models for planning and decision-making within emotional behavior trees, we introduced *neural nodes*. A neural node takes the current state of the environment and agent as input and, using a deep learning model, makes inferences about the emotional state. It contains a model, such as an emotion recognition system, that estimates the emotional state. These estimates are then mapped into the agent's affective state variables that parameterize the emotional selector. The neural node continually updates the agent's internal emotional state, allowing the dynamic adaptation of behavior trees to the emotional context.

At present, machine learning algorithms can help classify individuals' emotions depending on the input data. Emotions can be recognized from a wide spectrum of input data, encompassing physiological data, speech [4], facial expression [5, 6], and behavioral change [7]. Physiological input data encompasses factors like heart rate, frequency of respiratory movements, sweating and skin-galvanic reaction, and EEG (electroencephalogram) signals [8, 9, 10, 11, 12].

2.3. Knowledge Graphs

Knowledge Graphs are a particular type of knowledge base [13] consisting of sets of *facts* (i.e., *triples* such as "Dante," "wrote," "Divine Comedy") that interconnect *entities* ("Dante," "Divine Comedy") via *relationships* ("wrote") [14, 15]. Entities and relationships correspond to nodes and (labelled) edges of the KG, respectively. KGs have been extensively used in a plethora of application scenarios, including knowledge completion [16], head/tail prediction [17], rule mining [18], query answering [19], and entity alignment [20, 21, 22]. KGs are also known as information graphs [23], or heterogeneous information networks [24].

A noteworthy technique that is commonly exploited for tasks on KGs is *Knowledge graph embeddings* (KGEs) [25], which aims at generating a vector representation for entities and relationships of a KG.

In this work, we use KGs to represent (various kinds of) knowledge in the proposed NEMO framework. The main reasons why we employ KGs and prefer it over other types of knowledge base are as follows.

- First, the graph structure underlying KGs allows for capturing well the interrelations between the various entities of interest, and having such interrelations always and easily available, without performing possibly expensive operations (e.g., join operations in relational databases). This is eagerly needed in our context, as the proposed NEMO framework needs to continuously select specific portions of knowledge to be passed to the NEABT (see Section 3). Thus, this operation needs to be done efficiently.

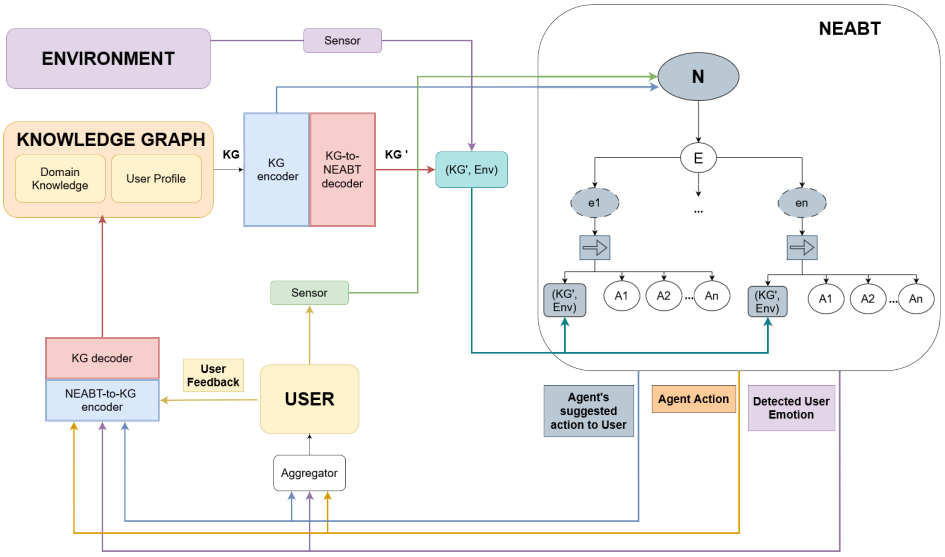

Figure 1: The proposed NEMO framework

- Second, KGs allow for the representation of heterogeneous entities and relationships among them.
- Third, KGs are highly flexible and versatile, as the knowledge therein need not to be organized according to any a-priori fixed schema. As such, KGs can easily integrate knowledge from different sources and perform updates on the acquired knowledge. Support for source heterogeneity and flexibility/versatility are particularly required for the proposed NEMO framework. In fact, as detailed in Section 3, the KG in NEMO encompasses different types of knowledge, possibly coming from different sources (i.e., domain knowledge of various forms and user profiles). Also, our NEMO framework has a mechanism for continuously updating the KG.
- Lastly, KGs can easily, yet effectively be represented as numerical vectors. This is required in our context in order to make KGs amenable to be processed by components of the proposed NEMO framework which do require numerical representations (i.e., the NEABT component, see Section 3). KGE techniques (see above) can represent KGs numerically. These are well-established techniques for which one can exploit the results of the corresponding research area, which has been very active and fruitful in the last few years.

3. Framework

The architecture of the proposed NEMO framework is illustrated in Figure 1. The main components of the architecture are the *User*, the *Environment*, a *Knowledge Graph*, and a *Behavior Tree*, specifically a NEABT. In this framework, the agent is identified by the NEABT. The overall interaction between the components of the framework is described next.

The NEABT is fed with signals from Environment, User, and KG. Such signals are exploited by the NEABT to perform its computation and to output (i) an action to be suggested to the User, (ii) an action actually performed by the agent (e.g., an empathetic action), and (iii) User's emotion detected by its neural node ('N', see below). Threefold NEABT's output passes through an "Aggregator", responsible for suitably aggregating and presenting NEABT's outputs to the User. The Aggregator may perform something either very simple (e.g., derive a textual representation of the three outputs and concatenate them) or more sophisticated (e.g., exploit a large language model (LLM)). NEABT's outputs and User's feedback are used to update back the KG. This way, we have a two-way, loop-back mechanism in which the NEABT exploits the KG for its internals, and the NEABT is exploited to update the KG properly.

In the following, we describe the User, Environment, KG, and NEABT components in more detail.

User. The User performs reactions and actions based on the signals provided by the NEABT. Sensory data from users flow into the NEABT through a sensor, which represents them in a proper numerical format. Also, the User's feedback – e.g., whether (or to which extent) the User has adopted the Agent's suggested action – is sent back to the KG. User's reactions/actions are assumed to be determined by all three NEABT output types. In particular, the User's emotion detected by the NEABT at the previous iteration is essential for establishing the emotional conditions that most influence the User. In fact, the User likely performs certain actions only under certain emotional circumstances. Thus, the emotional conditions that determine a certain User's behavior are an important signal to consider for both the presentation of NEABT's output to the User and the subsequent KG update step.

Environment. A sensor detects signals from the surrounding environment and represents them in some numerical format. Thus, the NEABT (along with the KG representation) is ready to process them.

KG. The Knowledge Graph (KG) contains information about domain knowledge and user profiles. KG's information is provided to the NEABT in a twofold form. It is first encoded in some proper numerical format and passed to NEABT's neural node (see below). The encoding is performed by a KG encoder component, which can be implemented, e.g., with a KGE (see Section 2.3). KG's encoded information is then decoded into a format suitable for processing by the internal nodes of the NEABT. A KG-to-BT decoder performs KG's information decoding. This can be implemented, e.g., as a neural network component whose training can be performed on a ground truth defined through manual annotation or the agent's historical data. The KG is fed with NEABT's output and user feedback. Such data in input to the KG need to be translated in a format suitable for updating the KG; for example, a set of KG triples to be added and a set of KG triples to be removed. A further encoder-decoder component performs this translation. Again, such an encoder-decoder can be implemented as a neural network and trained with a ground truth defined manually or through historical data.

Note that more sophisticated implementations and training strategies of the two encoder-decoder components may be possible. For instance, one could consider training the two encoder-decoder components simultaneously in an end-to-end fashion. This is expected to increase the effectiveness of those components. However, a downside of this solution is that it is technically challenging. A major hardness in this regard consists in making the NEABT a "differentiable" component so that it can be safely involved in the envisaged end-to-end neural training setting. For this reason, this could be an interesting direction of research that deserves dedicated effort, and we defer to future work.

Another interesting technical challenge is how to self-supervise the training of the two encoder-decoder components to overcome the burden of building a ground truth. An idea in this regard could be to mask a subset of the triples of the KG and use them in substitution for the ground truth. However, this leaves non-trivial technical questions open, such as how to select the triples of the KG to be masked and, more importantly, how to map the output of the NEABT into KG triples. For this reason, we leave this problem for future work as well.

Finally, a further intriguing idea – still deferred to future work – consists in avoiding the encoder-decoder components at all, and letting the interaction between the KG and the NEABT be carried out in an unsupervised way.

NEABT. The NEMO framework deploys a NEABT as a behavior tree. NEABT's neural node receives the KG's information and the user's sensory data and makes inferences about the user's emotional state. These estimates are mapped into the user affective state variables that parametrize the neural node child, the emotional selector. In turn, the emotional state selector passes the values of the affective state variables to its child nodes, empathy nodes. Each child empathy node provides an empathic evaluation of its subtree. In Figure 1, every subtree has a root node, a sequence selector with a condition node as a child and several action nodes. The condition child node returns success/failure by performing a test condition upon the input pair (KG', Env). The corresponding action child nodes are executed if the condition node returns success. By doing so, the NEABT can execute actions over the environment. Some of these action nodes define the NEABT threefold output.

Specifically, in the envisaged class of applications, to Human-AI Teaming, the Knowledge Graph

will include the user profile along with the history of past interactions, allowing the KG to interact with the NEABT in two ways: by passing data from the user profile to the neural node indicating the user's "classification" among several possibilities (in Section 4 some basic examples will be provided), and then passing to the leaves the type of action most suitable for that user (here too, we will provide some minimal example). Additionally, the KG should encompass an ontology on driving to assess, for instance, whether the user's action is allowed and/or ethical.

4. Prolog implementation

The implementation of the envisaged system is in an embryonic phase. However, we have first chosen to complete the implementation of our NEABT, making its connections with the other architecture components explicit. To do so, we chose to employ Prolog for the following reasons:

- The Prolog representation of a BT is easily readable and modifiable;
- Prolog rules provide the modularity and flexibility to represent the various components of the BT;
- Due to Prolog's fast prototyping capabilities, the implementation is ready to use and to be connected with any other component when ready.

In Figure 2, we show the first implementation of our NEABT. This implementation adheres to the workflow described in Figure 1: the neural network component processes knowledge graph embeddings and sensor data from the user to estimate the user's emotional state and related probability. The emotion selector component uses the predicted emotional state and context from the knowledge graph to select relevant sub-trees or nodes representing potential empathetic responses or actions. The context and action evaluation component tests the conditions. It executes the appropriate actions from the selected empathetic nodes, considering the user's emotional state and the context from the knowledge graph. If an action fails or the context changes, the system can select and execute alternative empathetic responses or actions based on the updated information.

Accordingly, the main predicates of our program are:

- $neabt\_structure/2$, it extracts all the structural knowledge about the behavior trees described utilizing the $child/3$ facts.
- $neural\_node/4$, (node **N** in Figure 1) this predicate encodes a neural network (indicated as nn, and accessed via a dedicated plugin). It takes as input the user sensor data, the KG embeddings, and the associated probability values from which it estimates the recognized emotional state.
- $emo\_selector/6$, (node **E** in Figure 1) this predicate is responsible for selecting relevant sub-trees or nodes based on the user's emotional state, the associated probability values, the context provided by the knowledge graph decoder and the environment state. It uses knowledge about the NEABT structure to discover nodes that succeed given the current situation using the $empathy\_node\_success/2$ predicate.
- $empathy\_node\_success/2$ (nodes **[e1, .., en]** in Figure 1), this predicate attempts to execute the sequence of actions within a sub-tree while testing the context provided by the knowledge graph decoder (KGd) and environment context (Env), recursively descending the tree; if a condition is met, it then launches the respective sequence of actions within the sub-tree or descends to sub-nodes. If an action fails, either because the user does not "comply" or impeding obstacles show up, it returns the failure to the upper selector node $emo\_selector/6$, so that the next best empathy node can be selected according to the probability ranking.
- $execute\_action/1$, it generates commands to the agent to execute the action as an external plugin. Each atomic action can fail independently, even if the context is appropriate. If a failure occurs, it returns the failure signal to the main $emo\_selector/6$ predicate, which then selects the next best emotional sub-tree.

```
neabt_structure(E, Children) :-
    % output: Children
    append(Action_nodes, Emphatic_nodes, All_nodes),
    findall(Child, child(E, All_Nodes, Child), Children),
    member(C, Children), C \= E.

neural_node(UserSensor, KGEmb, Prob, EmoState) :-
    % output: EmoState
    nn(UserSensor, KGEmb, Prob, EmoState).

emo_selector(E, EmoState, Prob, KGd, Env, SuccessNodes) :-
    % output: SelectedNodes
    neabt_structure(E, Children),
    select_relevant_nodes(EmoState, Prob, Children, SelectedNodes),
    % select_relevant_nodes is an external algorithm
    findall(Node, (member(Node, SelectedNodes),
                   empathy_node_success(Node, kg_condition(KGd, Env))),
            SuccessNodes).
    % kg_condition(KBd, Env) are facts from the KG

empathy_node_success(N, Condition) :-
    % BT implementation, extract the first that succeed
    neabt_structure(N, Children),
    (
        call(Condition) ->
        foreach(A, Children, action(A),
                execute_actions(A)), % external plugin
        foreach(SubN, Children, node(SubN),
                empathy_node_success(SubN, Condition)),
    ;
    true
    ).

neabt(UserSensor, KG, Env, E, SuccessNodes) :-
    neural_codec(KG, KGd, KGEmb),
    neural_node(UserSensor, KGEmb, Prob, EmoState),
    emo_selector(E, EmoState, Prob, KGd, Env, SuccessNodes).
```

Figure 2: Prolog prototype implementation

- $neabt/4$ is the main entry point, combining the neural network node, emotion selector, KG, environment and execution of appropriate empathy nodes. The predicate $neural\_codec$ encodes the KG into a suitable format for the neural network ($KGEmb$), and generates a decode representation (KGd) that provide the context.

4.1. Motivating example: Driver Co-Pilot

Here, we envision a case study that involves developing an intelligent agent that actively functions as a "companion" (co-driver) and support system for drivers. The agent will assist drivers by providing interventions in risky situations that may arise due to external circumstances and/or the driver's

health condition and emotional state, taking into account emotional aspects that could impact driving performance.

The intelligent agent will also be trained through interaction with the human user following the recent "Human-AI teaming" paradigm. A human driver could cooperatively train the agent by collaboratively performing various driving-related tasks, even under challenging scenarios. In this synergistic relationship during the training phase, humans enhance the effectiveness of automation (capabilities and performance). At the same time, the agent installed in each vehicle improves human efficiency and compensates for human inadequacies by intercepting and correcting potential erroneous behaviors, possibly resulting from compromised physical or emotional states.

Potential intervention modes for the agent to assist a struggling driver could include automatically activating (semi-)autonomous driving mode (if available) so the driver can momentarily divert their attention. Alternatively, the agent could more actively engage with the driver to regain attentiveness, for instance, by recommending stimulating music on a dedicated radio station. In case of health issues, the agent could recommend pulling over to rest or take medication (e.g., for hypertension) or, in critical cases, seek emergency assistance by contacting emergency services.

Consider, for instance, the case of truck drivers. Trucks come equipped with tachographs, devices that capture various data, including the operational hours of the vehicle and the driver's behavior concerning, e.g., speed, stops, etc. Regulations govern drivers' activities for safety and compliance purposes. The data necessary for checking compliance are stored within the tachograph to identify any breaches and generate a report detailing any violations that can lead to fines (also from the truck company) or even constitute a criminal offence. The KG can be interfaced with such devices to gather the data to feed the BT so as to issue suitable actions to help the driver avoid or alleviate violations, possibly explaining their reasons (e.g., too much stop because of some driver's discomfort). Importantly, our system might detect severe physical or psychological pain and alert human operators and emergency services.

Accordingly, in our use case, the system considers the following values for its decision-making process:

- The neural network can detect and categorize the driver's emotional state into one of the following values: [*sad, angry, happy, neutral, tired*]
- Based on the assessed emotional state and environmental factors, the system can select from a set of predefined actions to assist the driver:
    1. *advisory*, provide advisory messages or recommendations to the driver.
    2. *explicative*, offer explanatory information or guidance to the driver.
    3. *stop*, recommend or initiate the process of stopping the vehicle in critical situations.
    4. *reassurance*, deliver reassuring messages to alleviate the driver's stress or frustration.
- The system continuously monitors the driving environment through various sensors, collecting data such as:
    1. $road\_status$, information about the current road conditions, traffic, and potential hazards.
    2. $driving\_hours$, the duration of the current driving session.
    3. $speed\_data$, the vehicle's speed data, including also current speed and historical speed patterns.
    4. $stop\_duration$, the frequency and duration of stops taken during the driving session.

To provide an example, consider the following scenario: a user is driving, and the system receives environment sensor data [road_status(busy), driving_hours(5), speed_data(120, 150), stop_duration(1, 30)], indicating prolonged driving at high speeds with few stops. The emotional state detected is also identified as *tired*. Based on this input, our framework reasoning flow is as follows:

1. The neural network node processes the user sensor data and detects the emotional state *tired*.
2. The emotion selector identifies relevant nodes based on the *tired* emotional state.
3. The empathy node execution evaluates the user's state and selects an appropriate action, such as an *advisory*:
   `AgentAction = advisory_message("Slow down and take a break soon.").`

5. Related Work and Discussion

Neural architecture and the role of emotions. According to Damasio [26], emotions are unconscious reactions to internal or external stimuli that activate neural patterns in the brain. Emotions are innate reactions of the brain that are expressed through facial expressions, body language, and attitudes [27]. They affect the way people feel (consciously or unconsciously) since feelings are mental experiences of body states, which arise as the brain interprets emotions [28]. That, in turn, triggers changes in behavior and well-being.

The NEMO architecture follows Damasio's definition of emotions. Emotions are elicited in the neural nodes of NEABTs from both the environment and the agent's input. The recognized emotions form the affective state of the agent. The agent is conscious of the emotions in its affective state. The emotional values are then passed down in the behavior tree to the emotional and empathy nodes. Doing so triggers changes in the agent's behavior.

Knowledge graphs. Knowledge Graphs [14, 15] is a particular type of knowledge base [13] where knowledge is organized in a graph-like structure, i.e., with *triples* that define relationships (edges) among entities (nodes) of interest. KGs are also known as information graphs [23], or heterogeneous information networks [24].

KGs have been extensively used in a plethora of application scenarios, including knowledge completion [16], head/tail prediction [17], rule mining [18], query answering [19], and entity alignment [20, 21, 22]. KGs have also recently recently emerged as supporting tools for Retrieval-Augmented Generation (RAG) for Large Language Models (LLMs) [29, 30, 31, 32].

A well-established technique that is commonly exploited for tasks on KGs is Knowledge graph embeddings (KGEs) [25, 17]. KGEs generate numerical vector representations for entities and relationships of a KG, thus making them amenable to be processed in downstream tasks where a numerical representation is required (e.g., neural network-based machine-learning tasks). Although KGEs can differ (significantly) from one another in their definition, a shared key aspect of all KGEs is that they are typically defined based on a so-called *embedding scoring function* or simply *embedding score*. This function quantifies how likely a triple exists in the KG based on the embeddings of the entities and the relationship of that triple. Several KGEs have appeared in the last few years. The distinctive features among embeddings are the score function and the optimization loss. *Translational embeddings* in the TransE [33] family and the recent PairRE [34] assumes that the relationship of a triple performs a translation between the entities of that triple. *Semantic embeddings*, such as DistMult [18] or HolE [35], interpret the relationship as a multiplicative operator. *Complex embeddings*, such as RotatE [36] and ComplEx [37], use complex-valued vectors and operations in the complex plane. *Neural-network embeddings*, such as ConvE [38], perform sequences of nonlinear operations.

In this work, we use KGs as a building block of the proposed NEMO framework and KGEs as a tool to derive numerical representations of the KG. The latter is needed to make the knowledge in the KG processable by other components of our NEMO framework (which require knowledge represented in a numerical form).

Knowledge graphs and behavior trees. A significant use of the KG component in our NEMO framework is to have it interact with the NEABT component so that (i) the KG can adequately guide the actions to be performed by the agent and to be suggested to the user and (ii) the output of NEABT can be employed to update the knowledge in the KG suitably. This gives a "two-way" interaction between KGs and BTs, where the KG influences the processing logic of the BT, and, the other way around, the BT contributes to the knowledge stored in the KG. To the best of our knowledge, *this is the first work where a two-way interaction of this kind between KGs and BTs has been employed.* A few works exist in the literature about the simultaneous use of KGs and BTs. For instance, Axelsson and Skantze [39] devise a BT-based model that exploits a KG to generate the presentation of information by an agent to an audience. However, Axelsson and Skantze's model utilizes the KG *solely for presenting information to the user*, and not for *guiding the choices made by the BT*, like in our NEMO framework. Zhou *et al.* [40] propose a methodology that exploits a KG to generate a BT for robot task planning. However, in

Zhou *et al.*'s methodology, the KG is used *to generate a BT* (to be in turn used to help a robot perform its task planning), and not for *interacting with a pre-existing BT, and contributing together with it to perform agent's actions and suggest actions to the user*, like in our NEMO framework. Venkata *et al.* [41] devise a framework for knowledge transfer through BTs in a multi-agent system. Specifically, Venkata *et al.*'s framework encompasses a mechanism where the knowledge acquired by single agents is shared, through BTs, with the other agents of the multi-agent system. Thus, the use of BTs and knowledge representation Venkata *et al.*'s work is conceptually far from the use we do in our NEMO framework. Moreover, Venkata *et al.*'s framework does not use KGs to represent agents' knowledge.

From the above discussion, it is apparent that the literature about the simultaneous use of KGs and BTs is only marginally related to what we propose in this work. This makes our proposal of a two-way interaction between KGs and BTs entirely novel.

6. Conclusions and Future Directions

We have proposed an architecture for a system to be adopted in human-computer interaction and human-AI teaming. The architecture features relevant elements of novelty, encompassing a Knowledge Graph, a Neural Network, and a Behavior Tree, that moreover, as never done before, interact in two ways. We have presented a partial implementation of a part of the architecture, particularly the behavior tree and the related neural nodes, developed in Prolog, and we have discussed the advantages of the implementation. We have outlined a significant case study for testing our future system. However, once implemented, we intend to apply our system to other applications we are developing, firstly, assistive robotics. Future work includes further development of the implementation – including investigation of alternative paradigms, e.g., answer set programming (ASP) – and a suitable experimentation phase.

References

[1] M. Colledanchise, P. Ögren, Behavior trees in robotics and AI: An introduction, CRC Press, 2018.

[2] M. Iovino, E. Scukins, J. Styrud, P. Ögren, C. Smith, A survey of behavior trees in robotics and ai, Robotics and Autonomous Systems 154 (2022) 104096.

[3] S. Costantini, P. Dell'Acqua, G. De Gasperis, A. Rafanelli, Empowering emotional behavior trees with neural computation for digital forensic, 15th European Symposium on Computational Intelligence and Mathematics (ESCIM 2024) (in press).

[4] X. Lu, Deep learning based emotion recognition and visualization of figural representation, Frontiers in psychology 12 (2022) 818833.

[5] W. Wei, Q. Jia, F. Yongli, G. Chen, M. Chu, Multi-modal facial expression feature based on deep-neural networks, Journal on Multimodal User Interfaces 14 (2019).

[6] S. Hossain, G. Muhammad, An audio-visual emotion recognition system using deep learning fusion for a cognitive wireless framework, IEEE Wireless Communications 26 (2019) 62–68.

[7] M. N. Shiota, C. Vornlocher, L. Jia, Emotional mechanisms of behavior change: Existing techniques, best practices, and a new approach, Policy Insights from the Behavioral and Brain Sciences 10 (2023) 201–211.

[8] S. Alsubai, Emotion detection using deep normalized attention-based neural network and modified-random forest, Sensors 23 (2022) 225.

[9] S. An, Z. Yu, Mental and emotional recognition of college students based on brain signal features and data mining., Security & Communication Networks (2022).

[10] A. Subasi, T. Tuncer, S. Dogan, D. Tanko, U. Sakoglu, Eeg-based emotion recognition using tunable q wavelet transform and rotation forest ensemble classifier, Biomedical Signal Processing and Control 68 (2021) 102648.

[11] I. S. Ahmad, S. Zhang, S. Saminu, L. Wang, A. E. K. Isselmou, Z. Cai, I. Javaid, S. Kamhi, U. Kulsum, Deep learning based on cnn for emotion recognition using eeg signal, WSEAS (2021).

[12] R. Sánchez-Reolid, A. S. García, M. A. Vicente-Querol, L. Fernández-Aguilar, M. T. López, A. Fernández-Caballero, P. González, Artificial neural networks to assess emotional states from brain-computer interface, Electronics 7 (2018) 384.
[13] O. Deshpande, D. S. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, A. Doan, Building, maintaining, and using knowledge bases: a report from the trenches, in: SIGMOD, 2013, pp. 1209–1220.
[14] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM CSUR 54 (2022) 71:1–71:37.
[15] G. Weikum, Knowledge graphs 2021: A data odyssey, PVLDB 14 (2021) 3233–3238.
[16] X. Wang, L. Chen, T. Ban, M. Usman, Y. Guan, S. Liu, T. Wu, H. Chen, Knowledge graph quality control: A survey, Fundamental Research 1 (2021) 607–626.
[17] S. Ji, S. Pan, E. Cambria, P. Marttinen, S. Y. Philip, A survey on knowledge graphs: Representation, acquisition, and applications, Trans. Neural Netw. Learn. Syst. 33 (2021) 494–514.
[18] B. Yang, S. W.-t. Yih, X. He, J. Gao, L. Deng, Embedding entities and relations for learning and inference in knowledge bases, in: ICLR, 2015.
[19] Y. Wu, Y. Xu, X. Lin, W. Zhang, A holistic approach for answering logical queries on knowledge graphs, in: ICDE, 2023, pp. 2345–2357.
[20] S. S. Bhowmick, E. C. Dragut, W. Meng, Globally aware contextual embeddings for named entity recognition in social media streams, in: ICDE, 2023, pp. 1544–1557.
[21] J. Huang, Z. Sun, Q. Chen, X. Xu, W. Ren, W. Hu, Deep active alignment of knowledge graph entities and schemata, PACMMOD 1 (2023) 159:1–159:26.
[22] A. Zeakis, G. Papadakis, D. Skoutas, M. Koubarakis, Pre-trained embeddings for entity resolution: An experimental analysis, PVLDB 16 (2023) 2225–2238.
[23] M. Lissandrini, D. Mottin, T. Palpanas, D. Papadimitriou, Y. Velegrakis, Unleashing the power of information graphs, ACM SIGMOD Record 43 (2015) 21–26.
[24] C. Shi, Y. Li, J. Zhang, Y. Sun, S. Y. Philip, A survey of heterogeneous information network analysis, TKDE 29 (2016) 17–37.
[25] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, TKDE 29 (2017) 2724–2743.
[26] A. Damasio, Descartes' Error: Emotion, Reason and the Human Brain, Science and psychology, Papermac, 1996.
[27] J. P. Eberhard, Brain Landscape: The Coexistence of Neuroscience and Architecture, Oxford University Press, 2009.
[28] A. Damasio, Looking for Spinoza, A Harvest book, Harcourt, 2003.
[29] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, H. Wang, Retrieval-augmented generation for large language models: A survey, CoRR abs/2312.10997 (2023).
[30] S. Hao, T. Liu, Z. Wang, Z. Hu, ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings, in: NeurIPS, 2023.
[31] X. Wang, Q. Yang, Y. Qiu, J. Liang, Q. He, Z. Gu, Y. Xiao, W. Wang, KnowledGPT: Enhancing large language models with retrieval and storage access on knowledge bases, CoRR abs/2308.11761 (2023).
[32] J. Zhang, Graph-toolformer: To empower LLMs with graph reasoning ability via prompt augmented by ChatGPT, CoRR abs/2304.11116 (2023).
[33] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, NeurIPS 26 (2013).
[34] L. Chao, J. He, T. Wang, W. Chu, PairRE: Knowledge graph embeddings via paired relation vectors, in: ACL, 2021, pp. 4360–4369.
[35] M. Nickel, V. Tresp, H.-P. Kriegel, et al., A three-way model for collective learning on multi-relational data, in: ICML, 2011, pp. 3104482–3104584.
[36] Z. Sun, Z. Deng, J. Nie, J. Tang, RotatE: Knowledge graph embedding by relational rotation in complex space, in: ICLR, 2019.

[37] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, G. Bouchard, Complex embeddings for simple link prediction, in: ICML, 2016, pp. 2071–2080.
[38] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2d knowledge graph embeddings, in: AAAI, 2018.
[39] N. Axelsson, G. Skantze, Using knowledge graphs and behaviour trees for feedback-aware presentation agents, in: Proc. of Int. Conf. on Intelligent Virtual Agents (IVA), 2020, pp. 4:1–4:8.
[40] Y. Zhou, S. Zhu, W. Song, J. Gu, J. Ren, X. Xi, T. Jin, Z. Mu, Robot planning based on behavior tree and knowledge graph, in: Proc. of Int. Conf. on Robotics and Biomimetics ROBIO, 2022, pp. 827–832.
[41] S. S. O. Venkata, R. Parasuraman, R. M. Pidaparti, KT-BT: A framework for knowledge transfer through behavior trees in multirobot systems, IEEE Trans. Robotics 39 (2023) 4114–4130.