Software and System Modeling manuscript No.
(will be inserted by the editor)

Foundations for Complex Model Transformations by Reinforcement Learning with Uncertain Human Guidance

Istvan David[1,2] · **Kyanna Dagenais**[1]

Received: date / Accepted: date

Abstract Model-driven engineering problems often require complex model transformations (MTs), i.e., MTs that are chained in extensive sequences. Pertinent examples of such problems include model synchronization, automated model repair, and design space exploration. Manually developing complex MTs is an error-prone and often infeasible process. Reinforcement learning (RL) is an apt way to alleviate these issues. In RL, an autonomous agent explores the state space through trial and error to identify beneficial sequences of actions, such as MTs. However, RL methods exhibit performance issues in complex problems. In these situations, human guidance can be of high utility. In this paper, we present an approach and technical framework for developing complex MT sequences through RL, guided by potentially uncertain human advice. Our framework allows user-defined MTs to be mapped onto RL primitives, and executes them as RL programs to find optimal MT sequences. Our evaluation shows that human guidance, even if uncertain, substantially improves RL performance, and results in more efficient development of complex MTs. Through a trade-off between the certainty and timeliness of human advice, our method takes a step towards RL-driven human-in-the-loop engineering methods.

Keywords human guidance · machine learning · uncertainty

[1] McMaster University, Canada
[2] McMaster Centre for Software Certification (McSCert), McMaster University, Canada

1 Introduction

Modeling activities are often more complex than an atomic model transformation (MT) and rely on *sequences of MTs*. Pertinent examples can be found in model synchronization [1], model refactoring [2], and rule-based design-space exploration [3]. Typically, there might be more than one MT sequence that can successfully transform the source model into the target state, and choosing the most appropriate (cost-effective, efficient, safe) one manually is not tractable. This raises the **need for automated methods for developing complex MTs**, in which MTs are chained in sequences.

Reinforcement learning (RL) is a machine learning (ML) paradigm, in which an agent explores the state space, and through trial and error, learns beneficial action sequences [4]. This ability positions RL as a potential automation method for developing complex MTs. The benefits of RL have been recognized in MDE before, e.g., in model repair [5], in-place MTs [6], and model-driven service migration [7]. However, RL methods tend to perform poorly in the early phases of learning due to suboptimal decisions before converging to better policies [4]. That is, RL agents may require substantial training time and resources before learning an optimal sequence of MTs. This shallow learning curve limits the applicability of RL, particularly in cases where early mistakes can be costly, e.g., synchronizing models in model-based design of safety-critical systems. While the performance of RL agents improves over time, this improvement often comes too late to be useful.

Human guidance is an apt technique to mitigate problems, such as the shallow learning curve of RL agents [8]. In such a setup, human advice acts as

a heuristic to hint the otherwise autonomous agent towards desirable goals, thereby improving its performance. Involving human guidance in automated MDE methods has been explored before, e.g., driving RL-based MTs for model repair by personalized preferences [9], and driving search-based methods for MT-based design space exploration [10]. A common theme of existing techniques is the assumption that firm advice can be produced by the human in a timely and cost-effective fashion. This is a strong assumption that cannot always be guaranteed, especially when the human is asked to provide guidance about complex MTs (e.g., identifying consistent intermediate states in a long refactoring, or known locally optimal configurations in design-space exploration). Human advice, though often uncertain, can still be of high value in driving RL agents [11]. This raises the **need for incorporating human advice into RL-driven MTs through sound modeling and quantification of uncertainty**.

In this paper, **we develop an approach and technical framework to infer complex MTs through RL, guided by potentially uncertain human advice**. The sound formalization and assessment of uncertainty allow for producing useful human advice early in the MT learning process. Depending on the specific situation and engineering problem, early uncertain advice might be of higher value than late certain advice. We evaluate our approach in synthetic and real human guidance scenarios, including collaborative guidance with multiple humans. Our results indicate that human guidance, even if uncertain, substantially improves RL performance, and allows for the efficient inference of complex MTs.

Through a trade-off between the certainty and timeliness of guidance, our method takes a firm step towards ML-driven human-in-the-loop engineering methods. Our approach alleviates the manual labor required for developing complex MTs while enabling domain experts to steer the inference process. This convergence of human and artificial intelligence is much sought-after in complex engineering domains.

Novelty statement This article is an extended version of our conference paper [12] presented at the 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems (MODELS). We extend the original paper as follows.

- We provide more detail on Subjective Logic and compare it to other uncertainty frameworks (Sec. 2).
- We provide more detail on the approach and illustrate it through additional examples (Sec. 5).
- We report additional evaluation of the approach; specifically, in collaborative guidance (Sec. 6).
- We contribute a more detailed discussion of the results and their implications (Sec. 7).

Replicability For independent verification, we publish a replication package containing the data and analysis scripts of our study.[1]

Structure The rest of this article is structured as follows. In Sec. 2, we review the background topics of our work and the related work. In Sec. 3, we provide an example to illustrate our approach. In Sec. 4, we provide an overview of the approach, with a focus on the framework. In Sec. 5, we detail our approach to guide RL agents by opinions. In Sec. 6, we provide an evaluation and results of our experiments. In Sec. 7, we discuss the implications of our work. Finally, in Sec. 8, we draw the conclusion and identify future work.

2 Background and Related work

In this section, we review the background areas of our work, reinforcement learning (Sec. 2.1) and subjective logic (Sec. 2.2), and review the related work (Sec. 2.3).

2.1 Reinforcement learning

RL is a machine learning paradigm where an agent learns through sequential decision making by interacting with an environment and updating its strategy through the feedback received from the environment. While RL can perform autonomously, human input can be incorporated into the learning process in the form of guidance [8]. Such guidance can be used to bias the agent's exploration strategy to improve performance, reduce unsafe or inefficient behaviour, or align the agent's behaviour with human values. The ability to autonomously explore while also incorporating domain knowledge as human guidance is particularly relevant in MDE. The use of automation in MDE has been steadily increasing [13], with RL in particular being used as an automated method to learn MTs, such as learning MTs for model repair [9], and in-place MTs [6]. In this work, we use RL to learn complex MTs that are guided by human advice.

[1] The replication package of this extended version of our study is available at `https://github.com/ssm-lab/rl4mt-replication-package-sosym` and will be published on Zenodo after considering the reviewers' remarks. The replication package of the conference publication [12] is available at `https://doi.org/10.5281/zenodo.16321499`.

2.1.1 Formal underpinnings

In RL, an *agent* learns optimal control of an *environment* by taking sequential actions to explore the environment and update its strategy based on feedback in the form of *rewards* [4]. This can be formalized as a Markov decision process [14] $\langle S, A, \pi(a|s), r_t(s,a)\rangle$. S denotes the set of observable states the environment can produce, and A is the set of actions the agent can take. $r_t(s,a) \in \mathbb{R}$ is the reward the environment produces when the agent chooses action $a \in A$ while observing state $s \in S$, at time step t. At time step t, the agent observes a state s_t and chooses an action a_t. In response, at time step $t+1$, the environment produces a reward r_{t+1} and a new state s_{t+1}. The agent maintains a policy $\pi(a|s)$, which gives the conditional probability of choosing action a in state s. The agent's goal is to learn and act according to an optimal policy π_* that maximizes the sum of future rewards.

2.1.2 Taxonomy of RL Methods

While all RL algorithms aim to maximize expected return, various methods achieve this goal differently. *Value-based methods* aim to maximize either the state-value function $v_\pi(s) = \mathbb{E}_\pi[G_t|s]$, the action-value function $q_\pi(s,a) = \mathbb{E}_\pi[G_t|s,a]$, or both. Value-based approaches are deterministic because they select actions greedily when maximizing the value function, which might lead to under-exploration. In contrast, the goal of *policy-based* methods is to learn a parameterized policy that maximizes the reward function in every state without using value functions. Policy-based methods explore the state space more thoroughly than value-based methods, but produce estimates with more noise, potentially resulting in unstable learning processes. Finally, *actor-critic* methods provide a trade-off between value-based and policy-based methods, in which the actor implements a policy-based strategy, and the critic criticizes the actions made by the actor based on a value function. Different methods offer various benefits and perform with unique utility in specific learning problems.

In particular, *policy gradient* is a subset of the aforementioned policy-based methods. In policy gradient, the policy is parameterized through a parameter vector $\boldsymbol{\theta}$. For reasonably sized action/state spaces, the policy may be parameterized as numerical preferences for all state-action pairs $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$, and actions can then be chosen using a soft-max distribution. This is called *discrete policy gradient*, and often, a matrix-like representation of a look-up table is used. In contrast, the policy may be parameterized via a neural network such that $\boldsymbol{\theta}$ is a vector of network connection weights or linear in-features, which defines *deep policy gradient*. In either case, the policy is defined as $\pi(a|s,\boldsymbol{\theta})$, the probability of taking action a in state s with parameter $\boldsymbol{\theta}$. The policy may be parameterized in any manner, subject to the condition that it is differentiable with respect to $\boldsymbol{\theta}$. This way, $\boldsymbol{\theta}$ may be updated to maximize some scalar performance metric $J(\boldsymbol{\theta})$. **In this work,** we use discrete policy gradient so we can directly incorporate human guidance into the policy. The lookup table representation allows us to precisely identify where human guidance is to be incorporated, and explicitly control how it modifies the corresponding policy values.

2.1.3 Human guidance in RL

While RL algorithms can operate autonomously, evidence suggests that human guidance is beneficial to learning dynamics. Guidance is a method in which human input is used to bias the agent's exploration strategy. Techniques differ by when and how advice is provided, as described in the taxonomy of Najar and Chetouani [8]. For example, aligned with our approach, Cruz et al. [15] emphasize the role of *early advice*.

Strategies to incorporate guidance into RL are called *policy shaping* methods, which are defined by which part of the RL process they are incorporated in. In value shaping, human advice acts as a function of action preferences [16]. In reward shaping, advice is translated into numerical rewards given to the agent [17]. In decision biasing, human advice is used to directly affect the policy output [18]. Finally, in policy shaping, human advice is directly incorporated into the agent's policy to bias the exploration strategy. For example, Griffith et al. [19] estimate the human's feedback policy and incorporate this information into the agent's policy.

By this taxonomy, our work falls under the category of guidance, and specifically, policy shaping, since we aim to bias the agent's exploration by human guidance.

2.2 Uncertainty and Subjective Logic

Using human guidance to bias the RL agent's policy requires a formal approach to appropriately capture human opinions. In particular, the epistemic uncertainty of human opinions must be modeled appropriately to make guidance meaningful. Subjective Logic (SL) [25] provides such a formalism for reasoning under uncertainty that defines the construct of an opinion. An opinion explicitly separates belief from uncertainty, allowing human guidance to express both what the human knows, and their confidence about

Framework	R1 Epistemic uncertainty	R2 Separation of components	R3 Probability compatibility	R4 Multiple advisors
Dempster-Shafer Theory [20]	●	●	◑	◑
Beta Distribution [21]	◑	○	●	◑
Bayesian Probability [22]	◑	○	●	◑
Imprecise Probability [23]	◑	○	◑	◑
Fuzzy Logic [24]	○	○	◑	◑
Subjective Logic [25]	●	●	●	●

Table 1: Comparison of frameworks. (●: Full support. ◑: Partial support. ○: No support.)

their guidance. Additionally, SL is grounded in and compatible with traditional probability theory. This makes it particularly well suited to express guidance to policy-based RL agents, whose policies are represented as probabilities. The utility of SL in MDE has been demonstrated, e.g., in modeling with uncertain types [26–28] and prioritizing inconsistency resolution steps [29]. **In this work**, we use SL to model the uncertainty of guidance for RL in MT development.

2.2.1 Formal underpinnings

SL [25] separates and quantifies *belief* and epistemic *certainty* to model an opinion. By promoting opinions to first-class citizens, SL improves over traditional probabilistic logic. Opinions are indicators of emerging knowledge, thus, considering them promises better-informed RL. Given a boolean predicate x, a binomial opinion regarding the truth of x is given by $\omega_x = (b_x, d_x, u_x, a_x)$, where b_x represents the belief in the truthfulness of x; d_x represents the disbelief in the truthfulness of x; u_x quantifies the degree of epistemic uncertainty of the truthfulness of x; and a_x represents the base rate, i.e., the probability of x being true in the absence of (dis)belief. The following hold.

$$0 \leq b_x, d_x, u_x, a_x \leq 1 \tag{1}$$

$$b_x + d_x + u_x = 1 \tag{2}$$

$$P_x = b_x + a_x u_x. \tag{3}$$

Equation 3 expresses the *projected probability* of an opinion, effectively transforming opinions into the probability domain, where $P(x)$ is the probability that boolean predicate x holds. As uncertainty u_x increases, projected probability P_x is closer to base rate a_x. As uncertainty u_x decreases, projected probability P_x is closer to the belief b_x. A traditional probability p corresponds with $u_x = 0$, and can thus be transformed into a binomial opinion as follows: $\omega_x = (p, 1-p, 0, p)$.

A joint opinion of multiple advisors can be devised through the fusion of individual opinions. Fusion is a function $f : \Omega \times \Omega \to \Omega$ that maps a pair of opinions onto a new opinion. The appropriate fusion operator should be chosen based on its fit for purpose [30].

2.2.2 Comparison to alternative formalisms

Requirements To compare SL to alternative uncertainty formalisms, we derive requirements for preferentist uncertainty quantification from [31]. An ideal formalism shall exhibit each of these abilities.

R1 That is, the advisor's level of knowledge (or lack, thereof) shall be explicitly captured. In particular, the semantics of "*I don't know*" and "*I'm unsure*" shall be supported, as such statements are manifestations of epistemic uncertainty.

R2 *Separation of concerns.* That is, (i) belief and (ii) uncertainty of advice shall be distinct factors. Requirements **R1** and **R2** respond to the known limitation of the often used scalar-based representation of value and reward, which falls short of representing the existence of incomplete preferences [31].

R3 *Compatibility with probability.* That is, advice shall possess a natural translation to probabilistic logic. This requirement responds to the often unclear operationalization of human preferences for artificial agents [31]. As an RL agent's policy is typically represented as a probability distribution, clear translation to probabilistic logic allows opinions to be integrated into the agent's policy systematically.

R4 *Support for multiple advisors.* That is, the formalism shall support expressing joint opinions of multiple advisors. In other words, the algebraic structure formed by individual pieces of advice shall be closed under composition, i.e., merging two pieces of advice shall result in an advice. Supporting multiple advisors is important because it helps mitigate uncertainty of opinions. Furthermore, the composition semantics of opinions shall be able to handle situations in the absence of agreement [31].

Tab. 1 compares uncertainty modeling formalisms in terms of the requirements. SL offers a blend of abilities that positions **SL as an appropriate formalism for uncertain advice**, and consequently, for uncertain human guidance in RL. Other formalisms are limited in one or more requirements.

Closest to SL in its ability to model uncertain opinions is the Dempster-Shafer theory [20], which explicitly

captures uncertainty through the assignment of a belief mass (**R1**), and allows for the expression of complete or partial ignorance (**R2**). However, Dempster-Shafer has only an approximate translation to probability [32] (**R3**). Dempster-Shafer uses a rule of combination to combine belief functions (**R4**), but this is only a partial support compared to the semantically rich opinion fusion operators of SL.

A class of formalisms—including Beta distribution [21] and Bayesian probability [22]—are fully compatible with probability (**R3**) thanks to being probabilistic by nature. Such formalisms represent belief and certainty through simple mathematical concepts. Bayesian probability represents belief as a single probability value and uncertainty as its variance; Beta distribution represents belief by the mean of a distribution and uncertainty by its variance. At best, this is a partial approximation of epistemic uncertainty (**R1**), without the separation of concerns (**R2**). While distributions from different sources can be merged, this is limited support for multiple advisors (**R4**).

For completeness, we mention formalisms that are not suited for our purposes, but are widely used in simple uncertainty problems. Imprecise probability [23] represents uncertainty through the size of the probability interval. While this representation can capture uncertainty (**R1**), it is also possible that the width of the interval represents the values the user is expressing belief about. Thus, belief and uncertainty are not separated (**R2**). Fuzzy logic [24] does not support modeling of epistemic uncertainty, as it expresses degrees of truth rather than uncertainty (**R1**, **R2**). Thus, semantics of partial informedness are not supported. Compatibility with probability is partial (**R3**) and requires interpretations rather than direct translation. Multiple advisors (**R4**) are partially supported, but similar to the previous formalisms, operator semantics are limited.

2.3 Related Work

Closest to our work are approaches that use RL for MTs, particularly with human in the loop for guidance or intervention. The PARMOREL framework [9, 33] uses RL to learn user-preferred sequences of model transformations to fix invalid models. While this approach features human input for choosing an appropriate reward function, it does not accommodate the uncertainty of human guidance.

Eisenberg et al. [6, 34] use RL for learning in-place MTs. In this approach, the RL agent learns a sequence of MTs that optimizes objectives in the final model. The authors demonstrate that RL is a feasible alternative to traditional search heuristics, e.g., genetic algorithms. Our work improves over these approaches by including human guidance in RL.

Basciani et al. [35] introduce the CITRIC tool to recommend developers optimal MT sequences between a source and target metamodel. The approach ranks MT sequences through shortest path algorithms by two relevant criteria: metamodel coverage and information loss. Our approach is similar in aims but differs in technological solution, including the role of the human, who is an active advisor in our approach.

David and Syriani [36] translate simulator model engineering to RL processes and encode sequences of modeling actions in RL policies. Their approach allows for the direct, in-place engineering of DEVS simulation models, which is particularly useful, e.g., in handling concept drift in digital twins. However, their work does not make use of explicit MTs and by that, the inferred, often complex model editing sequences are not amenable to analysis or verification.

Burgueno et al. [37] propose a neural network architecture to infer heterogeneous MTs. Their approach leverages an encoder-decoder architecture that uses long-short term memory and attention mechanisms. In their approach, a neural network is trained using datasets of input-output models, facilitating the ability to learn MTs from example rather than preprogrammed rules. In contrast to their neural network based approach to learn MTs, our work uses RL to learn MTs and explicitly incorporates human guidance to improve the learning process.

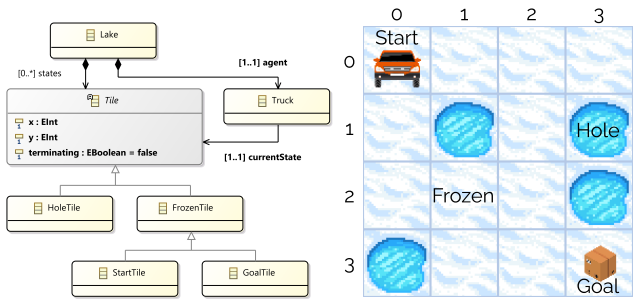
Finally, some additional works that automate MT development include the following. Eisenberg et al. [38] use multi-objective optimization to explore MT sequences using meta-heuristic search algorithms. Abdeen et al. [3] use a genetic algorithm for multi-objective optimization in rule-based design-space exploration. Mokaddem et al. [2] learn refactoring rules from examples using genetic programming.

3 Illustrative example

To illustrate our approach, we rely on the following running example of a self-driving vehicle through a frozen lake (Fig. 1). The example is analogous to Open AI's Gym's Frozen Lake environment[2], but it is fully modeled in EMF and its operational semantics are defined by MTs, akin to the Pacman case, a frequently used example in MDE [6, 39, 40].

The vehicle explores the lake by choosing to move up, down, left, or right one tile at a time. It begins its

[2] `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`



(a) Metamodel (b) Example model

Fig. 1: The Frozen Lake running example

exploration at the top leftmost tile (*Start*). Its aim is to reach the bottom rightmost tile (*Goal*) while avoiding terminating states (*Holes*).

3.1 Model transformations

Operational semantics are given by MTs. As shown in Fig. 2, moving to the tile to the right is achieved by executing the `moveRightRule` MT rule, detailed in Listing 1. To illustrate our approach, we use the VIATRA framework [41].

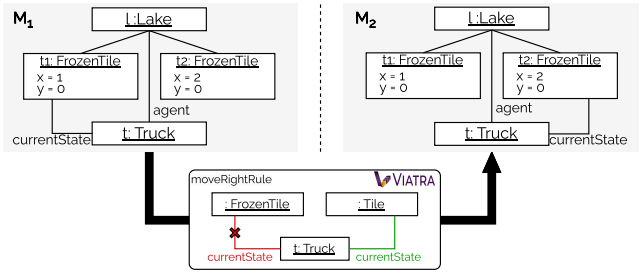


Fig. 2: Example MT rule and its execution

To successfully navigate through the lake, the vehicle must find a MT sequence that brings it to the goal state while avoiding terminating states (holes).

```
1 val moveRightRule =
2   createRule(agentOnFrozenTile)
3  .name("MoveRightRule")
4  .action[
5     val newTile = tiles.findFirst[
6       it.x==tile.x+1 && it.y==tile.y].head
7     truck.tile = newTile].build
```

Listing 1: Example VIATRA MT rule moving the agent right

3.2 Reinforcement learning

In RL terms, the MT sequence will generate *reward* upon encountering different states as follows.

Frozen tile Nothing happens. Reward = 0.
Hole The episode ends unsuccessfully. Reward = 0.
Goal The episode ends successfully. Reward = 1.

Through trial-and-error, the agent gradually learns which actions are beneficial in a specific situation. Eventually, it will develop a policy that allows for navigating from *Start* to *Goal* without falling into *Hole*s. By that, the policy encodes the MT sequence of interest, which can be materialized by selecting the actions (MTs) with the highest probability in each state.

The complexity of the problem prevents the agent from learning at a high pace. Typically, the agent struggles at the beginning of the learning process [4], especially when obstacles hinder exploration (e.g., the *Hole* in [1, 1], and the *Hole*s around [2, 1], forming a narrow passage).

3.3 Human guidance

Human guidance can be of significant value to accelerate the agent's learning rate. For example, a human advisor might suggest the agent to avoid [1, 1] as there is a hole (negative advice); or to visit [2, 1] or [1, 2], because those tiles are part of every viable path between *Start* and *Goal* (positive advice).

The advisor might **not be completely certain** in the advice. For example, advisors might be more certain about the vicinity of their location (situational expertise), or might be experts in recognizing *Hole*s but not *Goal*s (domain-specific expertise).

3.4 Result of the learning process

The result of the RL process is a policy that is able to assist the execution of MTs by providing priority information about which MT to execute in specific states, i.e., which direction to take while standing on specific frozen tiles. For example, the `moveRightRule` MT rule should be severely deprioritized when the vehicle is on tile [0, 1] as it would lead to a termination by falling into a hole.

4 Overview of the approach

4.1 Framework prototype

To implement and evaluate our approach, we prototyped a framework. The goal of the framework is to

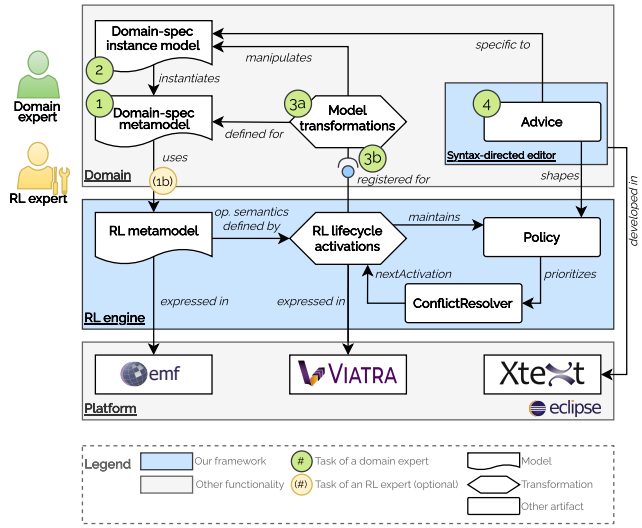


Fig. 3: Architectural overview

support the registration of MTs or model manipulation primitives from which complex MT sequences are inferred via RL. The framework is available as an open-source software.[3]

Fig. 3 presents the architectural view of the framework (in blue), in the context of the typical domain concepts, activities, and users. The core element of the framework is the *RL Engine*. The engine is fully modeled: the *RL metamodel* captures the domain-agnostic RL concepts (e.g., agent, policy, state) in the Eclipse Modeling Framework (EMF), and the operational semantics of RL (i.e., the underlying Markov Decision Process) is captured in MTs implemented on the VIATRA platform. These MTs span the *RL lifecycle activations* that drive the engine. (Described in detail in Sec. 4.4.) The *Policy* is maintained by specific activations. To resolve ambiguous MT activations (i.e., when multiple MTs can fire), we implemented a *ConflictResolver* that chooses the activation of the MT that is preferred in the Policy (i.e., has the highest probability in the state the agent exhibits).

4.2 Domain-specific (meta)modeling

The framework allows for executing MTs as RL programs in an MDE process, with minimal development effort.

In step (1), a *Domain-specific metamodel* is developed by the *Domain expert* that reuses some key concepts of the *RL metamodel*. Specifically, this step might be aided by an *RL expert* in case the domain expert is not familiar with RL concepts. These cases are typical when a domain expert works with RL for the first time; the concepts to work with (e.g., terminating states and rewards) are trivial and could be circumvented by generative techniques.

4.2.1 Mapping MTs to RL

Related works have provided a clear mapping between MTs and RL [5, 6]. Here, we only recall that graph morphisms or MT left-hand sides (LHS) correspond to states in RL programs and model manipulations correspond to actions in RL. Reward is automatically calculated (e.g., number of violations in model repair [5]), or assumes a reward structure (e.g., quality of solutions in model exploration [3]).

In the running example, the domain expert may choose to refine the concept of the agent into the concept of the *Truck*, and define the tiles on the lake as RL states, with frozen tiles being non-terminating states (the agent may continue the exploration) and holes and the goal being terminating states (the agent will stop exploring).

In Step (2), the domain expert instantiates the domain metamodel, e.g., as illustrated in Fig. 1.

4.3 Defining MTs

In Step (3a), the domain expert defines the MTs for the domain model. In the running example, this means defining the four movements (left, down, right, up) for the autonomous vehicle. Once the MTs have been defined, they are registered in the RL engine in Step (3b) to be executed as an RL program.

Typically, these MTs react to matched graph patterns expressed in terms of the domain-specific metamodel. This is why in the previous step, the domain expert (and in some cases, the RL expert) needed to ensure that the domain-specific metamodel uses some key concepts of the RL metamodel. Alternatively, users of the framework may opt for defining plain model manipulations without an LHS, instead of complete MTs. Upon registering model manipulations, the framework packages them into MTs with a pattern in the RHS that describes non-terminating states in the RL agent's state space. This allows for the execution of model manipulations as long as the RL program does not terminate.

4.4 Operational semantics

We developed our RL engine in a fully modeled fashion, in which the RL metamodel is operationalized by MTs. These MTs are internal to the RL engine and different

[3] https://github.com/ssm-lab/rl4mt

from the registered domain-specific MTs. For example, updating the policy upon encountering a terminating state is formulated as an internal MT. The activations of domain-specific and internal MTs span a lifecycle, shown in Fig. 4.

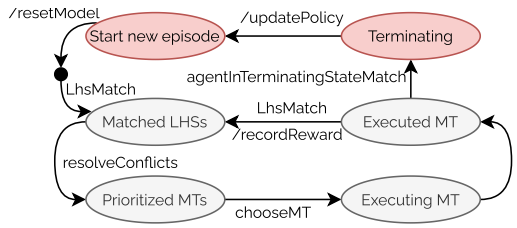


Fig. 4: RL lifecycle for learning MT sequences

The execution of the externally registered MTs starts by matching their LHS rules, resulting in a number of *Matched LHSs*. Typically, multiple MTs' LHSs are matched, which requires resolving conflicts. In our platform of choice, VIATRA, conflict resolution is achieved through a conflict resolver over a conflict set.[4] We implement our conflict resolution logic in a way that MTs are dynamically prioritized by their probability in the policy. Listing 2 shows the relevant excerpt of the `PolicyBasedConflictSet` class, which returns the next LHS activation.

```
override getNextActivation() {
    val nextActivation =
          policyService.chooseAction(container)
    container.clear
    return nextActivation
}
```

Listing 2: Excerpt of the *PolicyBasedConflictSet* class through which the policy-based conflict resolver prioritizes MTs

After resolving conflicts, the engine has a list of *Prioritized MTs*. After choosing one of the prioritized MTs, the engine starts *Executing the MT*. After execution, the result of the *Executed MT* determines how the lifecycle proceeds.

If the agent is situated in a terminating state at this point, the lifecycle continues by *Terminating* the episode, which includes updating the policy and *Starting a new episode* by resetting the model (e.g., placing the agent in the initial state). This internal MT is shown in Listing 3 and Listing 4.

[4] See `ConflictResolver.java` and `ConflictSet.java` on the API of VIATRA in `https://github.com/eclipse-viatra/org.eclipse.viatra`.

```
pattern agentInTerminatingState(
  agent: Agent,
  state: TerminatingState,
  environment: Environment){
    find agentInState(agent, state);
    Environment.agent(environment, agent);
}
```

Listing 3: Domain-agnostic graph query that matches when the agent is in a terminating state

```
val agentIsTerminatedRule =
  createRule(agentInTerminatingState)
 .name("AgentIsTerminatedRule")
 .action[agent.terminate].build
```

Listing 4: Domain-agnostic MT rule terminating the RL episode upon encountering a terminating state

If the agent is situated in a non-terminating state, the execution continues by recording the reward for the successful step and matching LHSs in the new state.

Through this lifecycle, the RL agent learns beneficial MT sequences based on the rewards for successful MT executions. The result of the RL process is a policy that can guide subsequent executions of the MTs to form the most efficient complex MT.

5 Guiding RL agents by opinions

In this section, we elaborate on Step 4. Our previous work [11, 42] presents the general method in great detail. Here, we focus on adopting its essential elements required for this paper. We refer to Fig. 5 in the remainder of this section.

5.1 Advisor input

First, the advisor provides advice to the agent. This step (shown in 4.1 in Fig. 5) is the only manual step in the method; the rest of the steps are automated in our technical framework.

The advisor expresses their subjective belief about a state being beneficial to the agent. We define advice as $\alpha : s \in S' \mapsto v$, i.e., a mapping from a state $s \in S'$ to a value v. Here, $S' \subseteq S$ is the subset of the problem space that the advisor can feasibly provide advice about, and v is a value that describes the benefit of occupying or exhibiting that state. In the running example, a domain expert could formulate the advice "*Cell (1,1)* [a hole] *is likely not beneficial*". Here, s refers to cell [1,1] in the grid world, and "not beneficial" is the value of occupying cell `[1,1]`. This advice is used to update the probability of choosing the action in adjacent states that leads to the advised cell, thereby shaping the policy.

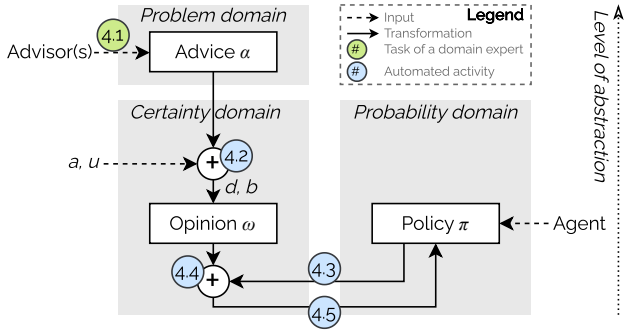


Fig. 5: Method for incorporating advice into RL

Advice is provided through an appropriate domain-specific language (DSL) [43] and ideally, in a *Syntax-directed editor*. To demonstrate principles and conduct our experiments, we developed a DSL and an Xtext-based editor for the running example. Such editors can be developed or generated for the specific case at hand.

Example GridWorld DSL for the Frozen Lake[5]

The DSL to provide advice about the grid world of the running example is shown in Listing 5. An example set of advice expressed in the DSL about the running example (Fig. 1) is shown in Listing 6.

```
<AdviceList> ::= <Advice>+
<Advice> ::= <AdviceLocation> ',' <AdviceValue>
<AdviceLocation> ::= '['[0-9]+, [0-9]+']'
<AdviceValue> ::= ^-?[0-2]
```

Listing 5: GridWorld grammar in EBNF

```
[1,1], -2
[3,1], -2
[3,0], -1
[3,3], +2
```

Listing 6: Example opinions for the problem in Fig. 1

The value of the advice represents the advisor's idea about how beneficial the given cell can be for the agent. In the example in Listing 6, the list of advice corresponds to the following cells of the running example (Fig. 1): two holes on the grid, which are not beneficial (`[1,1], -2`) and (`[3,1], -2`); the goal, which is beneficial (`[3,3], +2`); and a frozen tile that is not hazardous but still rather disadvantageous (`[3,0], -1`) due to the holes in its vicinity.

Uncertainty in advice may arise due to various reasons. Typical to multi-view modeling settings, a subject-matter expert may be required to give advice about adjacent domains' concepts. For example, a mechanical engineer may need to express the quality of the electro-mechanical design.

[5] For the reminder of this manuscript, grey boxes such as this one will contain illustrative examples.

5.2 Compiling advice into opinion

Next, in step 4.2, advice is transformed into opinions by mapping to the parameters of **subjective logic** [25]—briefly introduced in Sec. 2.2. This step is completely automated and is achieved in three sub-steps as follows.

5.2.1 Calibrating base rate

First, we set base rate a_i, which represents the prior probability of the agent taking an action in a state. For a set of actions A, the base rate is derived as

$$a_i = \frac{1}{|A|}. \tag{4}$$

In the running example, there are four actions the agent can choose in any state, thus $|A| = 4$ and the base rate $a_i = 0.25$.

5.2.2 Calibrating uncertainty

Next, we calibrate uncertainty u_i. While uncertainty can be set manually by the advisor, it can prove difficult to quantify, and thus, we recommend deriving uncertainty using an automated method. For example, uncertainty may be derived from an appropriate distance metric, such that uncertainty increases with distance. To calibrate uncertainty using a distance measure, it is essential to choose (i) an appropriate distance metric and (ii) a discount function that successively discounts uncertainty as distance increases.

In general the discount function is given as $\gamma : (u_{max}, \delta) \mapsto u_i \in (0, 1)$, where δ is a distance measured by a distance measure Δ, and u_{max} is the upper bound of uncertainty. From the identity of $u_i = 1 - (b_i + d_i)$ (Sec. 2), it follows that $u_{max} = 1 - (b_i + d_i)$. In the simplest case, a linear discount function can be chosen to calibrate uncertainty as follows.

$$u_i = \left(\frac{\delta}{\delta_{max}}\right) \times u_{max} \tag{5}$$

Here, δ/δ_{max} is the distance relative to the maximum distance, that maps onto the (0,1) domain.

In problems where uncertainty should be scaled only to a subset of the problem space, we use a threshold $0 < \tau \in \Delta < 1$ to accelerate the discounting of certainty

and reach u_{max} in τ. Then, Equation 5 is adopted as follows.

$$u_i = \begin{cases} {}^{1}\!/\!_{\tau} \times {}^{\delta}\!/\!_{\delta_{max}} \times u_{max} & \text{if } \delta \leq (\tau \times \delta_{max}); \\ u_{max} & \text{if } \delta > (\tau \times \delta_{max}). \end{cases} \quad (6)$$

5.2.3 Computing disbelief (d) and belief (b) to form an opinion

Finally, the belief b_i and disbelief d_i can be computed, and the parameters of the opinion ω_i can be set. As per the parameter constraints explained in Sec. 2.2, $u_i + b_i + d_i = 1$, and $0 \leq u_i, b_i, d_i \leq 1$, so the remaining weight of $1 - u_i$ is distributed between b_i and d_i based on the value of the advice α. Given n, the length of the advice value scale, and the value of the advice j (in ascending order of confidence from least to most), the calculation of b_i and d_i is given as

$$b_i = ({}^{j-1}\!/\!_{n-1}) \times (1 - u_i) \mid j \in \{1..n\} \quad (7)$$

$$d_i = (1 - u_i) - b_i. \quad (8)$$

By using Equations 4–8, the opinion ω_i is compiled as

$$\omega_i = \left(\frac{j-1}{n-1} \times (1 - u_i), \frac{n-j}{n-1} \times (1 - u_i), u_i, \frac{1}{|A|} \right). \quad (9)$$

The first parameter is the belief, b_i as formulated in Equation 7. The second parameter is the disbelief d_i (Equation 8). The third parameter u_i is the uncertainty calculated automatically by an appropriate metric. The final parameter a_i is the base rate calculated by the dynamics of the problem space (Equation 4). The corresponding algorithm is shown in Algorithm 1.

Algorithm 1 Compiling advice into opinion

Require: *Advice* α
Require: a ▷ inferred base rate
Require: u ▷ automatically calibrated uncertainty
Require: n ▷ advice scale length
  Opinion ω
  $\omega.a \leftarrow a$
  $\omega.u \leftarrow u$
  $\omega.b \leftarrow (order(\alpha.value) - 1)/(n - 1) \times (1 - \omega.u)$
  $\omega.d \leftarrow 1 - (\omega.b + \omega.u)$
  return ω

Example From advice to opinions in the Frozen Lake

Consider the advice from line 4 of Listing 6. Using an advice scale with length $n = 5$, the advisor indicates that the goal is beneficial $\alpha =$`([3,3], 2)`. Advice value $v = 2$ corresponds with $j = 5$. The agent can take one of the four actions in each cell in Fig. 1, so $a_i = {}^{1}\!/\!_{4} = 0.25$. This advisor has limited understanding of the environment, and is assigned uncertainty $u_i = 0.5$. Using Equation 9, we calculate ω_i as

$$\begin{aligned} \omega_i &= \left(\frac{j-1}{n-1} \times (1 - u_i), \frac{n-j}{n-1} \times (1 - u_i), u_i, \frac{1}{|A|} \right) \\ &= \left(\frac{5-1}{5-1} \times (1 - 0.5), \frac{5-5}{5-1} \times (1 - 0.5), 0.5, \frac{1}{|4|} \right) \\ &= (0.5, 0, 0.5, 0.25). \end{aligned}$$

5.3 Policy shaping via opinion fusion

In the fully automated steps 4.3 and 4.4, the agent's policy is shaped by the advisor's opinions as follows.

5.3.1 Transforming the policy into the certainty domain

As described in Sec. 2.1, the agent maintains a policy $\pi(a|s)$, which gives the conditional probability of choosing action $a \in A$ while in state $s \in S$. Since the entries are probabilities, $\forall s \in S, a \in A : 0 \leq \pi(a|s) \leq 1$. Before exploration, the probability of choosing any action from a state is equal, $\forall s \in S, a_i, a_j \in A, i \neq j : \pi(a_i|s) = \pi(a_j|s)$.

Example Default policy in the Frozen Lake problem

Tab. 2 shows the default policy of the running example. As shown, in each state (here, each cell), there are four actions available for the agent (left, down, right, up). Before training, the default policy is uniform. Each action is equally as likely to be taken in each state. Here, this probability is 0.25 as there are 4 possible actions to take in each state. Through trial and error, the agent will modify these probabilities.

Table 2: Default policy in the running example

	Left	Down	Right	Up
[0,0]	0.25	0.25	0.25	0.25
[0,1]	0.25	0.25	0.25	0.25
		...		
[3,2]	0.25	0.25	0.25	0.25
[3,3]	0.25	0.25	0.25	0.25

To fuse opinions described in the certainty domain, with a policy described in the probability domain, we choose to transform the agent's policy to the certainty domain (step 4.3). Our choice is motivated by the well-defined and sound fusion operators that are readily

available in the toolbox of subjective logic [25]. This translation is achieved as follows: $f_{P\to\Omega}(\pi) : S \times A \to \Omega \mid P$. As explained in Sec. 2.2, probability p translates to opinion as follows:

$$\omega : p \mapsto (p, 1-p, 0, p). \tag{10}$$

Algorithm 2 Transforming a policy in the probability domain (π_p) to a policy in the certainty domain (π_c)

```
Require: π_p                          ▷ Policy in the probability domain
  π_c                                 ▷ Policy in the certainty domain
  for Each state s ∈ S do
      for Each action a ∈ A do
          p ← π_p[s, a]
          π_c[s, a] ← (p, 1 − p, 0, p)
      end for
  end for
  return π_c
```

The corresponding algorithm is shown in Algorithm 2. For clarity, we refer to a policy expressed in the probability domain as π_p, and to a policy expressed in the certainty domain as π_c.

Example Default policy of the Frozen Lake problem in the certainty domain

Tab. 3 shows the default policy in the certainty domain transformed from the default policy in the probability domain shown in Tab. 2.

As shown, the previous probabilities of $p = 0.25$ are now expressed as opinions, following Equation 10 as $\omega = (0.25, 1-0.25, 0, 0.25) = (0.25, 0.75, 0, 0.25)$.

5.3.2 Policy shaping by opinion fusion

Next, in step 4.4, the advisor's opinions are fused with the agent's policy in the certainty domain. As described in Sec. 2.2, combined opinions can be fused into one joint opinion. By fusion, we are referring to a mapping $\Omega \times \Omega \to \Omega$. For fusion, we require two elements: locating the policy entries that are affected by the advice (in the running example: the neighboring states of the advised state), and choosing an appropriate fusion operator from the toolbox of subjective logic.

Locating policy entries to be fused Since the advisor gives advice about the value of a particular state being beneficial, this advice will affect the agent's policy in states from which the advised state is reachable. The following definitions are required to formalize this step.

Definition 1 (Neighboring states) States $s_i, s_j \in S$ are said to be neighbors if there exists a state-action pair in the policy for which $\pi(a|s_i) \mapsto s_j$, i.e., choosing action $a \in A$ in state s_i will transition the agent to state s_j.

Definition 2 (Neighborhood (of a state)) $N(s_i \in S) = \bigcup_{s_j \in S} \exists a \in A : \pi(a|s_i) \mapsto s_j$.

That is, neighborhood N of a state is the set of all neighbors. In the running example, the state space is topological; thus, the neighborhood of a cell comprises the cells that can be reached from the given cell through one of the agent's actions.

Advice α about state s_i is introduced to policy π by $\forall a \in N(s_i).A : \omega(\alpha) \odot \omega(a)$. That is, opinion $\omega(\alpha)$ formed from the advice is fused with every opinion formed from actions ($\omega(a)$) that lead from its neighboring states $N(s_i)$ to s_i.

Choosing a fusion operator To fuse opinions, Jøsang [25] defines several fusion operators that are classified based on the situation. In this work, the agent and advisor formulate their opinions separately, and it is fair to assume they will not compromise. Thus, we use the Belief Constraint Fusion (BCF) operator, as it is most appropriate when compromise is not sought, and those giving opinions have made up their minds. Joint opinion $\omega^{\odot} = (b^{\odot}, d^{\odot}, u^{\odot}, a^{\odot})$ under BCF is calculated by first finding the harmony and conflict of opinions: $Harmony = b_1u_2 + b_2u_1 + b_1b_2$; $Conflict = b_1d_2 + b_2d_1$. Then, the components of the joint opinion are calculated as follows.

$$b^{\odot} = \frac{Harmony}{(1 - Conflict)}; \quad u^{\odot} = \frac{u_1u_2}{(1 - Conflict)};$$

$$d^{\odot} = 1 - (b^{\odot} + u^{\odot}); \quad a^{\odot} = \frac{a_1(1-u_1) + a_2(1-u_2)}{2 - u_1 - u_2}.$$

As described in Sec. 2.2, more than two opinions may be fused into a joint opinion. Due to this and the fact that fusion operators are commutative, our approach does not limit the number of advisors that may provide advice to the agent.

Example Shaped policy of the Frozen Lake problem in the certainty domain

Tab. 4 shows the impact of a single advice `[1,1]→-2` on the default policy shown in Tab. 3. As defined in Theorem 5.3.2, shaping the policy impacts the neighbors of the advised state.

Locating opinions to be fused From Definition 2, $N[1,1] = \{[0,1], [1,0], [1,2], [2,1]\}$, with the respective actions of $\{Down, Right, Left, Up\}$ leading to

Table 3: Default policy of the Frozen Lake running example in the certainty domain

	Left	Down	Right	Up
[0,0]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)
[0,1]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)
		...		
[3,2]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)
[3,3]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)

Table 4: Advice `[1,1]→-2` fused into the policy of the Frozen Lake running example

	Left	Down	Right	Up
[0,0]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)
[0,1]	(0.25, 0.75, 0.0, 0.25)	**(0.143, 0.857, 0.0, 0.25)**	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)
			...	
[1,0]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	**(0.143, 0.857, 0.0, 0.25)**	(0.25, 0.75, 0.0, 0.25)
			...	
[1,2]	**(0.143, 0.857, 0.0, 0.25)**	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)
			...	
[2,1]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	**(0.143, 0.857, 0.0, 0.25)**
			...	
[3,3]	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)	(0.25, 0.75, 0.0, 0.25)

`[1,1]`. The corresponding opinions will be fused with the opinion formed from the advice.

Fusing opinions As shown in Tab. 5, advice `[1,1]→-2` translates to $\omega(\alpha) = (0.00, 0.50, 0.50, 0.25)$. In the default policy, each opinion is given by $\omega(a) = (0.25, 0.75, 0.00, 0.25)$. The fused opinion under Belief Constraint Fusion is calculated as follows.

$$\begin{aligned} \mathit{Harmony} &= b_1 u_2 + b_2 u_1 + b_1 b_2 \\ &= 0.00 \times 0.00 + 0.25 \times 0.50 = 0.125. \\ \mathit{Conflict} &= b_1 d_2 + b_2 d_1 \\ &= 0.00 \times 0.75 + 0.25 \times 0.50 = 0.125. \end{aligned}$$

$$b^{\odot} = \frac{\mathit{Harmony}}{(1 - \mathit{Conflict})} = \frac{0.125}{1 - 0.125} = 0.143.$$

$$u^{\odot} = \frac{u_1 u_2}{(1 - \mathit{Conflict})} = \frac{0.50 \times 0.00}{1 - 0.125} = 0.$$

$$d^{\odot} = 1 - (b^{\odot} + u^{\odot}) = 1 - (0.143 + 0) = 0.857.$$

$$a^{\odot} = \frac{a_1(1 - u_1) + a_2(1 - u_2)}{2 - u_1 - u_2} = \frac{0.25 \times (1 - 0.50) + 0.25 \times (1 - 0.00)}{2 - 0.50 - 0.00} = 0.25.$$

Thus, the resulting fused opinion is

$$\omega^{\odot} = (0.143, 0.857, 0.000, 0.250)$$

as highlighted in Tab. 4.

Table 5: The four pieces of advice and their opinion equivalents of the running example

Cell	**Advice**	**SL**			
		u	b	d	a
[1,1]	-2	0.500	0.000	0.500	0.250
[3,1]	-2	0.833	0.000	0.167	0.250
[3,0]	-1	1.000	0.000	0.000	0.250
[3,3]	+2	0.500	0.500	0.000	0.250

5.4 Transforming opinion-infused policies from the certainty domain to the probability domain

Finally, in automated step 4.5, the agent's policy—now shaped by advisor opinions—is transformed from the certainty domain back to the probability domain. First, as per Sec. 2.2, every opinion is transformed into a probability as follows:

$$p : \omega \mapsto b + au. \tag{11}$$

Second, the policy needs to be normalized. State-specific policy entries form a probability space, and thus the following invariant must hold:

$$\forall s \in S : \sum_{a \in A} p(a|s) = 1. \tag{12}$$

Normalization is defined as:

$$\forall s \in S, a \in A : p(a|s) := p(a|s) \times \frac{1}{\sum_{a \in A} p(a|s)}. \tag{13}$$

The agent's policy now reflects the advice, and the agent may begin the exploration process.

Table 6: Shaped policy of the Frozen Lake running example in the probability domain after advice `[1,1]→-2` affecting the actions of neighboring states `[0,1]`, `[1,0]`, `[1,2]`, `[2,1]` that lead to state `[1,1]`

(a) Before normalization (yellow: probabilities impacted by advice)

	Left	Down	Right	Up
[0,0]	0.25	0.25	0.25	0.25
[0,1]	0.25	0.143	0.25	0.25
		...		
[1,0]	0.25	0.25	0.143	0.25
		...		
[1,2]	0.143	0.25	0.25	0.25
		...		
[2,1]	0.25	0.25	0.25	0.143
		...		
[3,3]	0.25	0.25	0.25	0.25

(b) After normalization (green: increased probability, red: decreased probability)

	Left	Down	Right	Up
[0,0]	0.25	0.25	0.25	0.25
[0,1]	0.28↑	0.16↓	0.28↑	0.28↑
		...		
[1,0]	0.28↑	0.28↑	0.16↓	0.28↑
		...		
[1,2]	0.16↓	0.28↑	0.28↑	0.28↑
		...		
[2,1]	0.28↑	0.28↑	0.28↑	0.16↓
		...		
[3,3]	0.25	0.25	0.25	0.25

Example Shaped policy of the Frozen Lake problem in the probability domain

Calculating probabilities We take the fusion of the advice and the policy in Tab. 4 and for each tuple, we apply Equation 11. Tab. 6 shows the result of this transformation and the new policy that considers advice `[1,1]→-2`. Consider, e.g., the `Down` action in state `[0,1]`. Here, $\omega = (0.143, 0.857, 0.0, 0.25))$, and from Equation 11, $p = b + au = 0.143 + 0.25 \times 0.0 = 0.143$ follows.

As seen by the highlighted values, the advice affects the actions of neighboring states that lead to state `[1,1]`, i.e., `[0,1]` – `Down`, `[1,0]` – `Right`, `[1,2]` – `Left`, `[2,1]` – `Up`. Tab. 6a and Tab. 6b show the policy before and after normalization, respectively.

Normalization The resulting probabilities in row `[0,1]` violate invariant Equation 12 as $\sum_{a \in A} p(a|s = [0,1]) = 0.893$. Thus, we normalize by Equation 13 and multiply each probability by $\frac{1}{\sum_{a \in A} p(a|s=[0,1])}$, resulting in the probability vector (0.28, 0.16, 0.28, 0.28), as shown in the `[0,1]` row of Tab. 6b.

6 Evaluation

In this section, we evaluate our approach by answering the following research question: **how does human guidance affect the performance of RL in inferring MTs?** We conduct evaluations with on a scaled-up version of the running example; both with a single advisor, and with two cooperating advisors in the loop.

6.1 Evaluation setup

We conduct a comparative study in which we first drive MTs by an unadvised RL agent, and then, provide advice at various levels of uncertainty. We use the 12×12 scaled-up, randomized version of the Frozen Lake environment (Fig. 1). The map contains 20% *Hole* tiles, which is a standard ratio in OpenAI Gym's Frozen Lake implementation. The exact map is visualized in the replication package. We assess the performance of RL agents by the cumulative reward they collect. We measure both the amount of the cumulative reward and the pace at which the agent collects it. Our hypothesis is that human guidance has a measurable positive impact on both aspects, even at moderate levels of uncertainty.

The RL-specific parameters of our experiments are shown in Tab. 7. For the RL method, we choose discrete policy gradient [4] as policy-based methods explicitly represent the policy and allow for the direct investigation of the effects of advice. *Learning rate* α affects how much the agent updates its policy parameters during training. The closer it is to 0, the smaller the policy parameter updates, and the closer it is to 1, the larger the policy parameter updates. *Discount factor* γ determines how much the agent values future rewards. The closer γ is to 0, the more the agent prioritizes immediate rewards, and the closer it is to 1, the more the agent values long-term rewards. Each experiment runs for a *number of episodes*. An episode starts by resetting the environment and placing the agent in the initial position. Within the same experiment, the policy is continuously maintained. An episode lasts until the agent either finds a terminating state (hole or goal) or reaches the number of *maximum steps per episode*. To mitigate threats to validity, we tuned these hyperparameters to favor the learning dynamics of the unadvised agent.

6.1.1 Experiment parameters

Experiment parameters are shown in Tab. 8. We conduct 30 experiments for combinations of parameters

Table 7: RL hyperparameters and settings

Parameter	Value
RL method	Discrete policy gradient
Learning rate (α)	0.9
Discount factor (γ)	1.0
Number of episodes	10 000
Maximum steps per episode	100
State-action space	$12^2 \times 4$

Table 8: Experiment parameters

Parameter	Value
Agent type	{Random, Unadvised, Advised}
Source of advice	{Oracle, Single human, Cooperating humans}
Fusion operator	BCF
Advice quota – Oracle	{100%, 20%}
Advice quota – Single human	{10%, 5%}
Advice quota – Coop. human	{10% each, 5% each}
Uncertainty (oracle and single human)	$\{0.2k \mid k \in 0..4\}$
Uncertainty (cooperative humans)	2D Manhattan distance
Cooperative advice type	{Sequential, Parallel}

and calculate their average to identify systemic tendencies. We compare the performance of different *agent types*. The random agent chooses available actions randomly. The advised agent maintains and updates a policy. The unadvised agents also maintain and update a policy, and before exploration advice is incorporated into the policy.

We guide advised agents by different *sources of advice* and *advice quotas*. To form a ground truth, we first experiment with an idealized oracle with full information of the problem space, which provides advice based on pre-programmed rules. The oracle has two quotas; one where it provides advice about *all* states (100%) and another where it provides advice about the *holes and goal* on the map (20%).

In the *single human mode*, one of the authors acted as the advisor with full information of the problem space. This mode has two advice quotas: one where advice is provided about 10% of the state space, and the other where advice is provided about 5% of the state space. For both the *oracle* and *single human* advisors, the *degree of uncertainty* is modulated synthetically by setting $u = 0.0, 0.2, 0.4, 0.6, 0.8$.

In the *cooperative human mode*, both authors acted as advisors, each with partial information of the problem space. In this mode, uncertainty is calculated from the distance between the advisor and the location of the advice, in accordance with Sec. 5.2.

We use the two dimensional Manhattan distance, defined as follows.

$$D((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|.$$

Here, (x_1, y_1) is the location of the advisor. We assume a linear certainty discount function with $\tau = 1$, as explained in Equation 5. That is, uncertainty of an advisor in one of the corners of the grid reaches its maximum in the opposing corner. Thus, uncertainty at any point (x_2, y_2) is defined as follows.

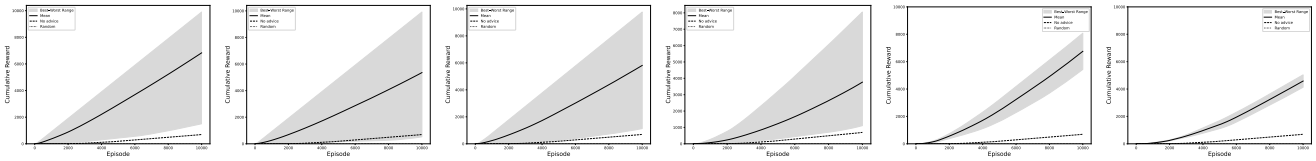
$$u(x_2, y_2) = D((x_1, y_1), (x_2, y_2)) \cdot \frac{1}{D((x_1, y_1), (x_n, y_n))}.$$

Specifically, we define two types of *cooperative advice with partial information*. In *sequential cooperation*, one human advises the agent in the first half of its mission, and subsequently, the other human advises the agent in the second half of its mission. To achieve this, we place the first human advisor in the top-left corner of the grid (start), and the second human advisor in the bottom-right corner of the grid (goal). As the agent follows the main diagonal of the grid world, the first advisor's input will gradually lose its influence due to the distance explained above; and the second advisor's input will gradually gain more influence. In *parallel collaboration*, both humans provide advice throughout the agent's entire mission. To achieve this, we place one human advisor in the top-right corner of the grid, and the other human advisor in the bottom-left corner of the grid. By that, both advisors are able to provide advice throughout the entire start-goal trajectory of the agent. However, due to the distance metric defined above, both advisors will focus on their half of the grid world, i.e., the triangle under and above the diagonal, respectively. Clearly, this advice mode is the most *realistic one* of the three as in real settings, uncertainty pertains to individ-

Table 9: Cumulative rewards by experiment. Random = 0.033, Unadvised = 698.266. Bold is best at the given level of certainty. Cooperative modes aligned with the best matching uncertainty level of the comparable oracle or single human mode.

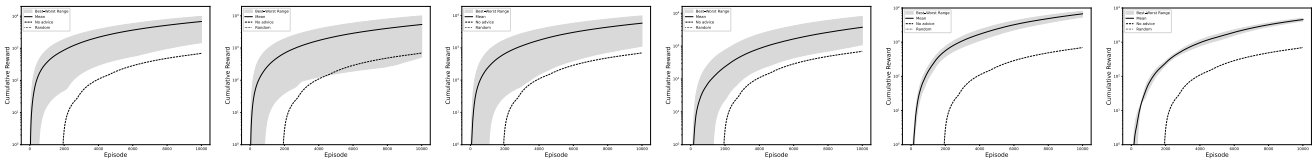
	Oracle		Single human		Coop. – Sequential		Coop. – Parallel	
u	100% (All)	20% (H&G)	10%	5%	10%	5%	10%	5%
0.0	9 900.100	**9 914.900**	9 768.000	8 051.300	8 078.366			
0.2	**9 685.900**	8 948.933	8 538.266	5 287.833			5 429.466	
0.4	**7 974.066**	5 216.433	6 121.033	2 134.966				
0.6	**5 094.333**	2 177.633	3 488.700	2 246.733		5 037.066		4 130.666
0.8	**1 502.500**	523.633	1 126.300	1 108.666				

Linear scale



(a) Oracle 100% (b) Oracle 20% (c) Human 10% (d) Human 5% (e) Coop 10% (f) Coop 5%

Log scale



(g) Oracle 100% (h) Oracle 20% (i) Human 10% (j) Human 5% (k) Coop 10% (l) Coop 5%

Fig. 6: Mean cumulative rewards (black) and best-worst performance range across experiments (shaded))

ual pieces of advice, and cooperation among stakeholders and experts is the way to solve complex problems.

6.2 Results and key takeaways

In the following, we report the results of our experiments by reviewing the performance of guidance by the oracle (Sec. 6.3), a single human advisor (Sec. 6.4), and cooperative human advisors (Sec. 6.5).

In addition, we evaluate statistical significance between each advice mode at different levels of uncertainty. We use independent samples t-tests for the pairwise comparison. The replication package contains the complete list of t-test results.

Main takeaways. Tab. 9 and Fig. 6 report the cumulative rewards in different evaluation modes and highlights the key takeaways of our work.

We observe that **advised agents perform better than unadvised ones**, as in all but one case (Fig. 6b), the worst-case performance of advised agents is better than the unadvised agent. In particular, we note that the mean cumulative reward of advised agents is substantially higher than the cumulative reward of the unadvised agent, even with limited advice quota.

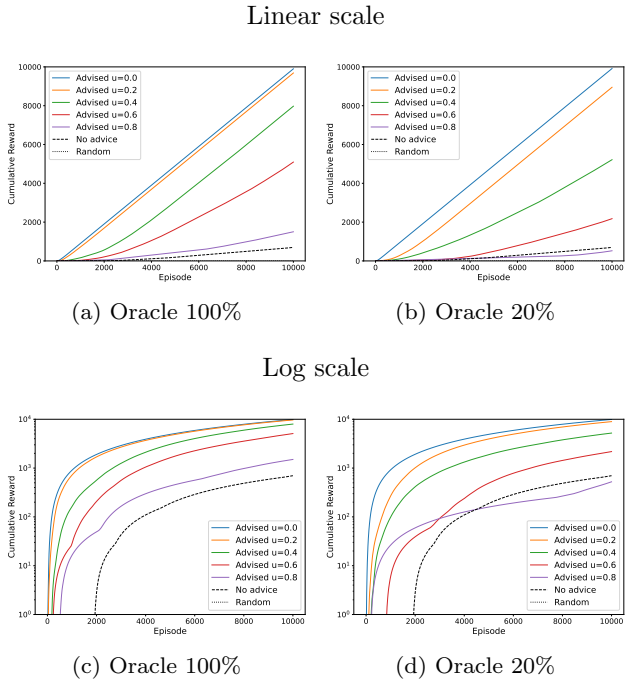
We also note that higher advice quotas tend to result in better performance. This is evident, as agents advised with higher quotas tend to have a higher mean cumulative reward. In particular, we see that the best performance from guidance by an oracle with 100% quota Fig. 6a. However, we note that with decreased quotas, other advising scenarios have comparable performance. In particular, we note that a single human with a substantially lower advice quota of 10%(Fig. 6c) is able to achieve comparable performance to an oracle with quotas of 100% and 20% (Fig. 6a, Fig. 6b).

Reducing the advice quota of a single human advisor to 5% (6d) decreases performance compared 10%, but we find this scenario has similar performance to cooperative advice settings with 10% and 5% advice quotas (Fig. 6e, Fig. 6f). Overall, these results indicate that **guidance, even if uncertain, improves the**

performance of the RL agent learning complex MTs, even if in a limited quantity or from a partially informed source.

6.3 Performance of guidance by an oracle

Fig. 7 shows the cumulative reward of agents provided with synthetic advice from an oracle under the two advice quotas.



(a) Oracle 100% (b) Oracle 20%

(c) Oracle 100% (d) Oracle 20%

Fig. 7: Cumulative reward with an oracle as the advisor

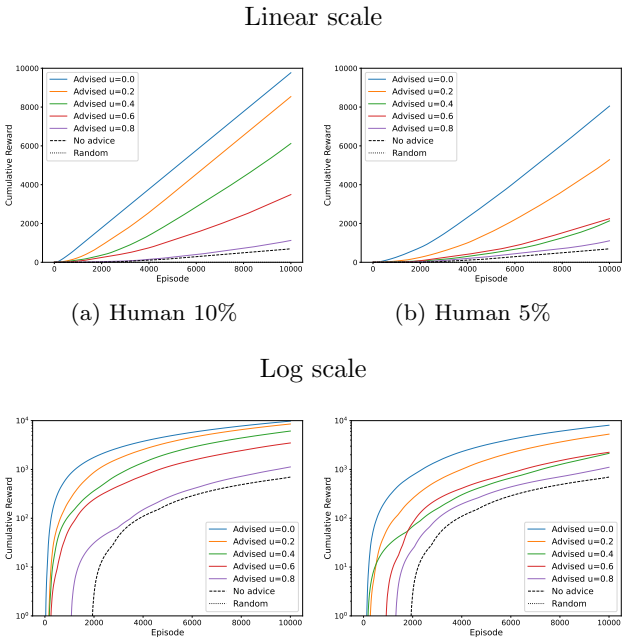
We observe that the performance of advised agents improves as certainty increases, as cumulative reward is higher, and increases at a higher level for higher certainty advice. This is evident as the lines corresponding with agents given advice with higher certainty are consistently higher than the lines corresponding with agents given advice with lower certainty.

We observe that the oracle with 100% quota has the best overall performance, accumulating the most cumulative reward for all but one $u = 0.0$, where the oracle with 20% quota is slightly better. This can be seen in Tab. 9, where, for $u = 0.0$, the oracle with 100% quota obtains final cumulative reward of 9 900.100, and the oracle with 20% quota obtains a final cumulative reward of 9 914.900.

We also observe that the unadvised agent obtains more cumulative reward than the oracle with 20% quota for $u = 0.8$. As seen in the log plots Fig. 7d, while the advised agent accumulates reward earlier than the unadvised agent, at about 4 5000 episodes, the unadvised agent continues to gain reward, while the advised agent plateaus. The advised agent recovers and begins to approach the cumulative reward of the unadvised agent near the end of the training, but still achieves less. This finding indicates that indicates that at very high levels of uncertainty ($u \geq 0.8$), a modest quota of synthetic advice may help agents in early learning phases, but hinder later performance.

6.4 Performance of guidance by a single human advisor

Fig. 8 shows the cumulative reward of agents provided with advice from a single under the two advice quotas.



(a) Human 10% (b) Human 5%

(c) Human 10% (d) Human 5%

Fig. 8: Cumulative reward with a single human advisor

We see that key performance trends retained, as agents advised perform better than unadvised agents, and that performance of advised agents improves as certainty increases.

While the oracle with 100% advice quota (Fig. 7a) significantly outperforms the human advisor with 10% quota (Fig. 8a) at low levels of uncertainty ($u \leq 0.4$), this difference is not significant at higher levels of uncertainty ($u \geq 0.6$). That is, **in uncertain situations, a**

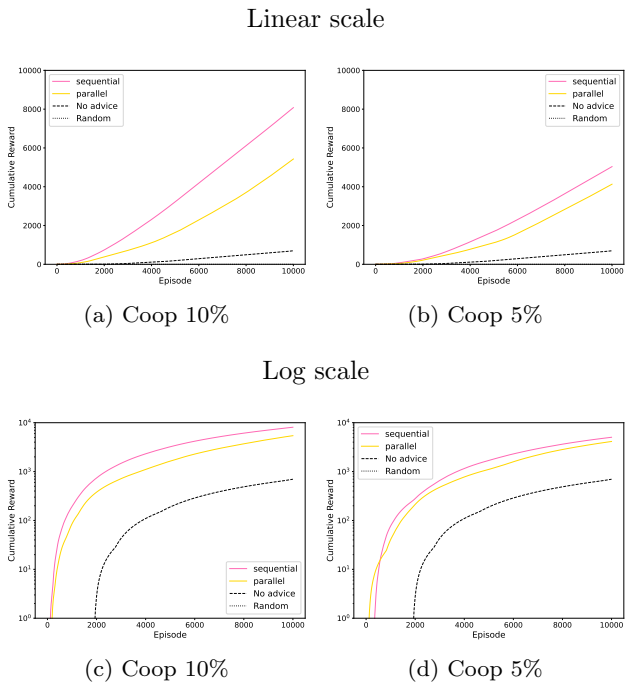
human advisor, even with a lower advice quota, does not perform systematically worse than a hypothetical oracle. This finding highlights the utility of human advice in uncertain situations.

This trend is even more pronounced when the oracle is given less advice quota (Fig. 7b). With 20% quota, the oracle does not significantly outperform a human advisor with 10% quota when $u > 0.0$.

At low levels of uncertainty ($u \leq 0.4$) a human advisor with 10% quota significantly outperforms the human advisor with 5% quota. However, at higher levels of uncertainty ($u \geq 0.6$), the difference is not significant. These findings suggest that **different uncertainty levels require different advice strategies from human advisors**. Specifically, when advisors are certain, more advice is justified; however, when uncertain, advisors should be more frugal with their advice.

6.5 Performance of guidance by cooperating human advisors with partial information

Fig. 9 shows the cumulative reward of agents provided with advice from two cooperating human advisors under the two advice quotas.



(a) Coop 10% (b) Coop 5%

(c) Coop 10% (d) Coop 5%

Fig. 9: Cumulative reward with two cooperating humans the advisors

We again see key performance trends being retained, as advised agents perform better than unadvised agents, despite decreased advice quotas, and human advisors with partial knowledge about the problem.

We observe that for both advice quotas, that sequential cooperation out performs parallel cooperation. This is evident as the lines corresponding with sequential cooperation are consistently higher than line corresponding with parallel cooperation. These findings indicate that having one advisor guide an agent through the first portion of a problem, then handing off to a second advisor when the first advisor's uncertainty peaks, is a more effective strategy than both advisors guiding an agent through the problem together.

We also observe that the results of the cooperative experiments are comparable to the performance of an oracle, and single human advisor at particular levels of uncertainty. As indicated in Tab. 9, we note that sequential cooperation where each advisor has a 10% quota is comparable to an oracle with a 20% quota at no uncertainty $u = 0.0$. We see that sequential cooperation where each advisor has a 5% quota is comparable to an oracle with a 100% quota at moderate uncertainty $u = 0.6$.

Parallel cooperation where each advisor has a 10% quota is comparable to a single human advisor with a 5% quota at low uncertainty ($u = 0.2$). Finally, parallel cooperation where each advisor has a 5% quota is comparable to a single human advisor with a 10% quota at moderate uncertainty ($u = 0.6$).

These findings indicate that even with partial information, cooperating advisors can achieve similar performance to low to moderate uncertainty in fully-informed situations. As these cooperative experiments emulate realistic settings, where several advisors operate with partial information, it is plausible to expect the performance improvements of uncertain guidance can be retained in real applications of learning complex MTs.

Our observations indicate that human guidance contributes to significant performance improvement in RL-based inference of complex MTs, even at moderate-to-high uncertainty. Human advisors, even with substantially lower advice quotas, can perform at the level of an oracle, and might even outperform it. In realistic settings with cooperating humans with partial information about the problem, we find that guidance, even if uncertain and with low advice quotas, still results in improvements over unadvised agents. This results in faster and more efficient inference of complex MTs.

6.6 Assumptions, limitations, and threats to validity

6.6.1 Assumptions and limitations

We implicitly assumed that the reward structure could be effectively formulated and rewards are observable. In some MDE applications, this is true. For example, Barriga et al. [9] formulate reward by a user-preferred quality characteristic, such as maintainability and understandability. Still, formulating reward might be challenging in open problems, such as design space exploration, in which reward often must be asserted to hypothetical, yet-to-encounter model configurations. In such situations, advanced RL techniques can be considered, e.g., inverse RL [44] that extracts a reward function of an agent by observing its behavior.

6.6.2 Threats to validity

A fortunately aligned reward structure may result in better, albeit non-systemic performance of the RL agent, threatening the *internal validity* of our study. To mitigate this threat, (i) we have chosen the least supportive reward function in which the goal reward is minimal, there are no intermediate rewards, and terminating states do not provide negative feedback; and (ii) tuned the hyperparameters to favor the learning dynamics of the unadvised agent. To assess the *external validity* of our study, we remark that we only tested policy-based RL. While it is reasonable to expect the method to be applicable to other RL methods (e.g., value-based RL, a popular choice in MDE [6, 9, 45]), our study does not provide evidence for this. Similarly, a minor threat to external validity stems from the technological choices, especially the MT platform. We rely on the reactive capabilities of Viatra, and it is reasonable to expect that MT frameworks with support for similar MT execution semantics can adopt our approach.

7 Discussion

We now discuss the implications of our work and outline areas of interest for MDE researchers.

7.1 Towards RL-enabled MT platforms

The result of the RL process is a policy that encodes beneficial MT executions in specific graph morphisms. To utilize this information, we foresee two main directions: runtime policy exploitation in MT chains (Sec. 7.1.1) and generating standalone MTs from the learned policy (Sec. 7.1.2).

7.1.1 Runtime policy exploitation in MT chains

The RL engine can be used to resolve MT activations based on the policy. The policy assigns higher probabilities to specific MTs in a model state when those MTs have yielded higher positive rewards in previous occurrences of similar states. Such scenarios can be supported through appropriate conflict resolvers that factor in the policy information, as shown in Fig. 3 and Fig. 4. The benefit of this approach is that the RL engine can continuously maintain the complex MT and guide the execution of complex MTs in a non-intrusive way.

This mode is particularly useful when MT execution is not a one-shot activity, but part of a continuously used model management process. Examples include repeated model repair [46], synchronization [1], refactoring [2], or design-space exploration [3] tasks, where the same transformation rules are executed over related but not identical models over time. In such settings, the policy can act as a learned conflict resolution strategy: instead of replacing the MT engine, it prioritizes among otherwise valid rule activations. This allows existing transformation engines, such as Viatra, to retain their pattern matching and execution semantics, while the RL component contributes learned operational knowledge about which activation is likely to be most beneficial in the current model state [41].

Runtime policy exploitation also provides a natural point for incorporating new human advice. Since the policy remains available during execution, it can be reshaped when new opinions are provided, when uncertainty changes, or when the model reaches states that were not sufficiently covered during training. This is important for practical RL-enabled MT platforms because engineering contexts are rarely static: constraints may change, new model variants may appear, and previously beneficial MT sequences may become suboptimal. In such cases, policy-guided conflict resolution supports a feedback loop in which execution experience and human guidance can continue to refine the complex MT after initial training. This makes the approach well suited to interactive and human-in-the-loop MDE settings, where advice is expected to influence learning dynamics rather than merely define a fixed reward function [8, 15, 18].

The main cost of this mode is platform integration. The MT platform must keep the RL engine, the policy, and the transformation execution infrastructure connected at runtime. It must also expose suitable hooks for observing states, recording rewards, resolving conflicts, and updating the policy. This integration challenge is consistent with broader software-engineering

work on RL frameworks, which highlights the need for reference architectures to compare, evaluate, and integrate recurring RL framework components [47]. However, this integration cost is balanced by greater adaptability: the learned complex MT remains open to further training, additional advisors, and changing engineering objectives.

7.1.2 Generating standalone MT chains

Alternatively, standalone MT chains can be generated to be used independently from the RL engine. In such scenarios, the RL engine is used only for training purposes and does not maintain the MT chain.

In this mode, the learned policy is treated as an intermediate artifact from which an executable MT sequence, macro-rule, or transformation script is extracted. For each relevant state or graph morphism, the platform can select the highest-ranked action in the policy and materialize the corresponding MT activation as part of a reusable chain. The resulting artifact can then be executed by a standard MT engine without requiring the RL infrastructure to be present. This makes standalone generation attractive for settings in which the learned transformation needs to be reviewed, versioned, certified, shared, or deployed in an environment where an online RL component would be too heavyweight.

Standalone MT chains also make the result of learning more transparent to MT engineers. While a policy is useful for runtime decision making, it is not necessarily the artifact that MDE practitioners expect to inspect or maintain. By materializing the policy in an explicit transformation chain, learned behaviour can be analyzed as an MDE artifact: engineers can inspect the selected rule sequence, compare it to existing handwritten transformations, validate it on representative models, and incorporate it into existing model management workflows. This connects our approach to seminal works on recommending or learning MT sequences [37, 38], while preserving the distinctive role of RL and uncertain human guidance during the training phase.

The limitation of standalone generation is that the extracted chain freezes a particular interpretation of the learned policy. Once detached from the RL engine, the chain no longer adapts to new rewards, new advice, or model states that differ substantially from those observed during training. Consequently, this mode is best suited to relatively stable problem classes, recurring transformation scenarios, or deployment contexts in which predictability and ease of integration are more important than continued learning. For more dynamic settings, standalone chains may need to be regenerated periodically from an updated policy.

We anticipate MT platform developers to adopt techniques such as ours and promote RL-enabled MTs to first-class citizens in their platforms. While runtime policy exploitation leverages our approach to a greater extent, generation of standalone MT chains is less intrusive and requires a smaller leap for the user base of MT platforms. Together, the two modes suggest complementary adoption paths: runtime policy exploitation targets adaptive MT platforms, whereas standalone chain generation targets integration with existing MT tooling and engineering processes.

7.2 Typical applications

To better contextualize our approach within MDE, we explain how some characteristic MDE problems map onto RL and can make use of our approach.

7.2.1 Model synchronization and inconsistency management

These problems are related and both often require identifying complex MTs [48–50]. In these settings, the RL **state** encapsulates morphisms over the union of the source and target models, and the **reward** is typically implied by the negative distance between the source and target models. The lower the distance, the better the solution. Termination semantics are asserted to states that minimize the distance or, equivalently, maximize the reward. Distance can be measured at the syntactic level (e.g., by counting inconsistent model elements) [51] or at the semantic level (e.g., by counting inconsistent properties that are not being jointly satisfied by the two models) [52].

The role of our approach. **Actions** are typically expressed as domain-specific MTs, but are limited in complexity. Our approach allows for inferring complex MTs that enable more sophisticated synchronization semantics between models. **Human guidance** is of particular value in these problems as multi-domain settings challenge the understanding of how exactly consistency should be restored [29, 53].

7.2.2 Model repair and refactoring

This class of problems is similar to model synchronization; however, there are some key differences. RL **state** in this case encapsulates only a single model's morphisms, and **reward** cannot be automatically calculated due to the lack of a suitable oracle. That is, it

cannot be structurally defined when a refactoring is successful or sufficient. Instead, the reward structure must be defined by the domain expert. Typically, qualitative metrics are used for this purpose, either defined at the general syntactic level (e.g., number of dangling edges or isolated components) [46] or at the domain-specific syntactic level [40]. Occasionally, personalized metrics are considered [9].

The role of our approach. The challenge is similar to the previous class of problems: **actions** are usually expressed as domain-specific MTs with limited complexity. Here again, an approach such as ours would allow for inferring complex MTs that, in turn, enable more complex refactorings and model repair actions. **Human guidance** helps prevent models from growing out of size under the premise of repair.

7.2.3 Design space exploration

Design space exploration (DSE) is the process of identifying favorable candidate designs under various constraints and soft objectives [54]. Starting from an initial design, a DSE process gradually applies MTs to uncover alternative designs. In such scenarios, the **state** encapsulates a specific graph morphism in a model instance. **Actions** are typically defined in domain-specific terms [10] rather than domain-agnostic CRUD primitives. The **reward** structure is given through constraints and soft objectives.

The role of our approach. Optimal solutions are often found deep in the design space and are computationally demanding to identify. Our approach can aid the computational load of DSE by learning complex MTs that have led to beneficial design alternatives. These complex MTs, then, can be used as fixed points in the design space from which subsequent DSE executions may continue deeper explorations. **Human guidance** helps by providing known favorable design decisions and by that, drive the RL agent to favorable states and accelerate the inference of complex MTs. The challenge of uncertainty in DSE has been recognized before [55]. Our approach allows for the explicit modeling and management of uncertainty that stems from human guidance.

7.3 Research opportunities for the MDE community

There are some key challenges to be addressed in order to materialize the opportunities our approach creates. In the following, we elaborate on the ones that align particularly well with the expertise and knowledge of the MDE community.

7.3.1 Domain-specific languages for guidance

A key advantage of opinion-based guidance is the rapid pace at which opinions emerge. To leverage this advantage to its fullest, intuitive languages are needed to express opinions. Domain-specific languages (DSLs) [43] have been widely used to capture expert opinions at high levels of abstraction, narrowing the gap between the modeling language and human cognition. However, DSLs typically operate with reduced syntax to remain truly specific to the problem domain, and therefore, **automated methods for DSL engineering** are in high demand [56]. This is true for guided RL as well. We envision language workbenches that allow for the automated construction of DSLs through which advice can be provided by domain concepts. In the running example (Sec. 3), one might want to use descriptive quantifiers, such as “hazardous” or “beneficial” instead of relying on an integer scale. Unfortunately, there is little research on **domain-specific languages for RL**, and existing languages mostly focus on programming RL procedures [57] rather than expressing opinions at high cognitive levels. We see plenty of room for research targeting domain-specific modeling languages for RL, inspired by the advancements in the model-driven engineering community [58, 59].

7.3.2 (Domain-specific) Tool support

Our approach can be aided by domain-specific tool support at key points. For example, the domain expert needs a **domain-specific editor** with an intuitive user interface to efficiently provide advice. We implemented the prototype of such an editor on top of Eclipse and Xtext, as discussed in Sec. 5.1. Instead of case-by-case tool development, syntax-directed editors could be generated from high-level specifications of the RL problem and the application domain [60], and equipped with automatically generated specialized concrete syntax [61]. Adaptive modeling languages [62] offer an apt solution through a product line-inspired approach to customizing DSLs to different problems.

Additional tooling considerations relate to **model management** and RL execution. In our work, we executed the RL agent’s learning process on an instance model that should likely be prevented from automated complex MTs until the RL agent is trained. Instead, an improved version of our tool could create replicas of the real model and treat those as sandbox models for the agent to train. Such sandbox models need not be arbitrary copies: they can be managed as families of RL training environments, where environment variants are generated from model-level mutations and con-

straints [63]. The result of the training, i.e., the policy, can be applied to the real model. Such utility functionality will pave the way for adoption.

7.3.3 Interactive advice

In our experiments, we assumed that advice is provided once, before the agent begins exploring the environment. However, given adequate interaction protocols, our approach can accommodate advice at any point during the RL process. We see opportunities in developing more **interactive advice protocols** through advanced human-computer interaction (HCI) [64]. MDE can be of high value in such situations, as it can support **domain-specific interfaces** that integrate user models in the system design. This would allow for interactive advising systems that are tailored to the user's level of expertise about the problem space [65]. Blended modeling—the activity of interacting seamlessly with a single model (i.e., abstract syntax) through multiple notations (i.e., concrete syntaxes), allowing a certain degree of temporary inconsistencies [66]—can be of high utility in facilitating seamless advising workflows in multi-view settings.

Finally, interaction protocols amenable to **verification and validation** can be generated, e.g., from state chart models [67] or sequence diagrams [68].

8 Conclusion

We presented a method for engineering complex model transformation (MT) sequences by reinforcement learning (RL), guided by potentially uncertain human advice. Our results show that human advice, even if uncertain, can be a major contributor to more efficient MT development. Through a combination of subjective logic (SL), appropriate domain-specific languages, and a suitable distance function, opinions can be introduced into RL as the means of guidance. The sound mathematical framework of SL allows for multiple sources of opinions to be uniformly fused with the agent's original policy, effectively achieving policy shaping through opinions. Our results open the door for the RL-based human-in-the-loop treatment of key MDE problems, e.g., model synchronization and design-space exploration. We envision MT platform developers adopting techniques such as ours to promote RL-backed MTs to first-class citizens in MT platforms. This will enable efficient MT development, execution, and maintenance.

Future work will focus on evaluating our approach for different flavors of RL and developing domain-specific support for RL-driven human-in-the-loop MT engineering.

References

[1] K. Marussy, O. Semeráth, and D. Varró, "Incremental view model synchronization using partial models," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '18, Copenhagen, Denmark: ACM, 2018, pp. 323–333. DOI: 10.1145/3239372.3239412.

[2] C. E. Mokaddem, H. Sahraoui, and E. Syriani, "Recommending model refactoring rules from refactoring examples," in *Proc. of the 21th ACM/IEEE Intl. Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '18, Copenhagen, Denmark: ACM, 2018, pp. 257–266. DOI: 10.1145/3239372.3239406.

[3] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, and Á. Horváth, "Multi-objective optimization in rule-based design space exploration," in *Proc. of the 29th ACM/IEEE Intl Conference on Automated Software Engineering*, ACM, 2014, pp. 289–300. DOI: 10.1145/2642937.2643005.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] A. Barriga, A. Rutle, and R. Heldal, "Personalized and automatic model repairing using reinforcement learning," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 175–181. DOI: 10.1109/MODELS-C.2019.00030.

[6] M. Eisenberg, H.-P. Pichler, A. Garmendia, and M. Wimmer, "Towards reinforcement learning for in-place model transformations," in *2021 ACM/IEEE 24th Intl Conf. on Model Driven Engineering Languages and Systems (MODELS)*, 2021, pp. 82–88.

[7] M. Dehghani, S. Kolahdouz-Rahimi, M. Tisi, and D. Tamzalit, "Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning," *Soft. Sys. Mod.*, vol. 21, no. 3, pp. 1115–1133, 2022. DOI: 10.1007/s10270-022-00977-3.

[8] A. Najar and M. Chetouani, "Reinforcement learning with human advice: A survey," *Frontiers in Robotics and AI*, vol. 8, 2021, ISSN: 2296-9144. DOI: 10.3389/frobt.2021.584075.

[9] A. Barriga, R. Heldal, A. Rutle, and L. Iovino, "Parmorel: A framework for customizable model repair," *Soft. Sys. Mod.*, vol. 21, no. 5, pp. 1739–1762, 2022.

[10] Á. Hegedüs, Á. Horváth, and D. Varró, "A model-driven framework for guided design space exploration," *Autom Softw Eng*, vol. 22, no. 3, pp. 399–436, 2015. DOI: 10.1007/s10515-014-0163-1.

[11] K. Dagenais, "Towards model repair by human opinion–guided reinforcement learning," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24, Linz, Austria: ACM, 2024, pp. 192–195. DOI: 10.1145/3652620.3676878.

[12] K. Dagenais and I. David, "Complex model transformations by reinforcement learning with uncertain human guidance," in *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2025. DOI: 10.1109/MODELS67397.2025.00025.

[13] L. Burgueño, D. Di Ruscio, H. Sahraoui, and M. Wimmer, "Automation in model-driven engineering: A look back, and ahead," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, May 2025, ISSN: 1049-331X. DOI: 10.1145/3712008. [Online]. Available: https://doi.org/10.1145/3712008.

[14] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[15] F. Cruz, P. Wüppen, S. Magg, A. Fazrie, and S. Wermter, "Agent-advising approaches in an interactive reinforcement learning scenario," in *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, 2017, pp. 209–214. DOI: 10.1109/DEVLRN.2017.8329809.

[16] A. Najar, O. Sigaud, and M. Chetouani, "Training a robot with evaluative feedback and unlabeled guidance signals," in *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2016, pp. 261–266. DOI: 10.1109/ROMAN.2016.7745140.

[17] A. C. Tenorio-Gonzalez, E. F. Morales, and L. Villaseñor-Pineda, "Dynamic reward shaping: Training a robot by voice," in *Advances in Artificial Intelligence – IBERAMIA 2010*, Springer, 2010, pp. 483–492, ISBN: 978-3-642-16952-6.

[18] A. L. Thomaz and C. Breazeal, "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *Proc of the 21st National Conf on Artificial Intelligence - Volume 1*, ser. AAAI'06, Boston, Massachusetts: AAAI Press, 2006, pp. 1000–1005, ISBN: 9781577352815.

[19] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[20] G. Shafer, *A Mathematical Theory of Evidence*. Princeton: Princeton University Press, 1976, ISBN: 9780691214696. DOI: doi : 10 . 1515 / 9780691214696. [Online]. Available: https://doi.org/10.1515/9780691214696.

[21] A. K. Gupta and S. Nadarajah, *Handbook of beta distribution and its applications*. CRC press, 2004.

[22] B. A. Olshausen, "Bayesian probability theory," *The Redwood Center for Theoretical Neuroscience, Helen Wills Neuroscience Institute at the University of California at Berkeley, Berkeley, CA*, 2004.

[23] T. Augustin, F. P. Coolen, G. De Cooman, and M. C. Troffaes, *Introduction to imprecise probabilities*. John Wiley & Sons, 2014.

[24] L. A. Zadeh, "Fuzzy logic," *Computer*, vol. 21, no. 4, pp. 83–93, 1988.

[25] A. Jøsang, *Subjective Logic*. Springer International Publishing, 2016, ISBN: 9783319423371. DOI: 10.1007/978-3-319-42337-1.

[26] L. Burgueño, M. F. Bertoa, N. Moreno, and A. Vallecillo, "Expressing confidence in models and in model transformation elements," in *Proc of the 21th ACM/IEEE Intl Conference on Model Driven Engineering Languages and Systems*, ACM, 2018, pp. 57–66.

[27] E. Bagheri and A. A. Ghorbani, "A belief-theoretic framework for the collaborative development and integration of para-consistent conceptual models," *J. Sys. & Soft.*, vol. 82, no. 4, pp. 707–729, 2009, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2008.10.012.

[28] J. Troya, N. Moreno, M. F. Bertoa, and A. Vallecillo, "Uncertainty representation in software models: A survey," *Soft. Sys. Mod.*, vol. 20, no. 4, pp. 1183–1213, 2021, ISSN: 1619-1374. DOI: 10.1007/s10270-020-00842-1.

[29] R. Jongeling and A. Vallecillo, "Uncertainty-aware consistency checking in industrial settings," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2023, pp. 73–83.

[30] A. Jøsang, P. C. Costa, and E. Blasch, "Determining model correctness for situations of belief fusion," in *Proceedings of the 16th International Conference on Information Fusion*, 2013, pp. 1886–1893.

[31] T. Zhi-Xuan, M. Carroll, M. Franklin, and H. Ashton, "Beyond preferences in ai alignment," *Philosophical Studies*, Nov. 2024. DOI: 10.1007/s11098-024-02249-w.

[32] F. Cuzzolin, "On the relative belief transform," *International Journal of Approximate Reasoning*, vol. 53, no. 5, pp. 786–804, 2012, ISSN: 0888-613X. DOI: https://doi.org/10.1016/j.ijar.2011.12.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888613X11001861.

[33] L. Iovino, B. Angela, R. Adrian, and H. Rogardt, "Model repair with quality-based reinforcement learning.," *JOT*, vol. 19, no. 2, 17:1, 2020. DOI: 10.5381/jot.2020.19.2.a17.

[34] M. Eisenberg and M. Wimmer, "From single-objective to multi-objective reinforcement learning-based model transformation," *Soft. Sys. Mod.*, 2024, ISSN: 1619-1374. DOI: 10.1007/s10270-024-01233-6.

[35] F. Basciani, D. Di Ruscio, M. D'Emidio, D. Frigioni, A. Pierantonio, and L. Iovino, "A tool for automatically selecting optimal model transformation chains," in *Proceedings of the 21st ACM/IEEE Intl. Conf. on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '18, Copenhagen, Denmark: ACM, 2018, pp. 2–6. DOI: 10.1145/3270112.3270123.

[36] I. David and E. Syriani, "Automated Inference of Simulators in Digital Twins," in *Handbook of Digital Twins*. CRC Press, 2024, pp. 122–148. DOI: 10.1201/9781003425724-11.

[37] L. Burgueno, J. Cabot, S. Li, and S. Gérard, "A generic lstm neural network architecture to infer heterogeneous model transformations," *Software and Systems Modeling*, vol. 21, no. 1, pp. 139–156, 2022.

[38] M. Eisenberg, A. Sahay, D. Di Ruscio, L. Iovino, M. Wimmer, and A. Pierantonio, "Multi-objective model transformation chain exploration with MOMoT," *Inf Softw Technol*, vol. 174, p. 107 500, 2024, ISSN: 0950-5849.

[39] R. Heckel, "Graph transformation in a nutshell," *Electronic Notes in Theoretical Computer Science*, vol. 148, no. 1, pp. 187–198, 2006, ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2005.12.018.

[40] E. Syriani, R. Bill, and M. Wimmer, "Domain-specific model distance measures," *Journal of Object Technology*, vol. 18, no. 3, 3:1–19, 2019, The 12th International Conference on Model Transformations, ISSN: 1660-1769. DOI: 10.5381/jot.2019.18.3.a3.

[41] G. Bergmann, I. David, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró, "VIATRA 3: A reactive model transformation platform," in *Theory and Practice of Model Transformations - 8th International Conference, ICMTSTAF 2015, L'Aquila, Italy. Proceedings*, ser. LNCS, vol. 9152, Springer, 2015, pp. 101–110. DOI: 10.1007/978-3-319-21155-8_8.

[42] K. Dagenais and I. David, "Opinion-guided reinforcement learning," Tech. Rep., 2024. arXiv: 2405.17287 [cs.LG].

[43] D. C. Schmidt, "Model-driven engineering," *Computer-IEEE Computer Society*, vol. 39, no. 2, p. 25, 2006.

[44] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00, Morgan Kaufmann Publishers Inc., 2000, pp. 663–670.

[45] M. Sharbaf, B. Zamani, and G. Sunyé, "Automatic resolution of model merging conflicts using quality-based reinforcement learning," *Journal of Computer Languages*, vol. 71, p. 101 123, 2022. DOI: https://doi.org/10.1016/j.cola.2022.101123.

[46] G. Taentzer, M. Ohrndorf, Y. Lamo, and A. Rutle, "Change-preserving model repair," in *Fundamental Approaches to Software Engineering*, Springer, 2017, pp. 283–299, ISBN: 978-3-662-54494-5.

[47] X. Liu and I. David, "A reference architecture of reinforcement learning frameworks," in *2026 IEEE 23rd International Conference on Software Architecture (ICSA)*, 2026. DOI: 10.1109/ICSA66085.2026.00016.

[48] T. Mens and R. Van Der Straeten, "Incremental resolution of model inconsistencies," in *Recent Trends in Algebraic Development Techniques*, Springer, 2007, pp. 111–126, ISBN: 978-3-540-71998-4.

[49] M. El Hamlaoui, S. Bennani, M. Nassar, S. Ebersold, and B. Coulette, "A mde approach for heterogeneous models consistency," in *Proc of the 13th Intl Conference on Evaluation of Novel Approaches to Software Engineering*, ser. ENASE 2018, Funchal, Madeira, Portugal: SCITEPRESS, 2018, pp. 180–191. DOI: 10.5220/0006774101800191.

[50] I. David, J. Denil, K. Gadeyne, and H. Vangheluwe, "Engineering process transformation to manage (in)consistency," in *Proceedings*

of the 1st International Workshop on Collaborative Modelling in MDE (COMMitMDE 2016) co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2016), St. Malo, France, October 4, 2016, ser. CEUR Workshop Proceedings, vol. 1717, CEUR-WS.org, 2016, pp. 7–16. [Online]. Available: http://ceur-ws.org/Vol-1717/paper5.pdf.

[51] L. Marchezan, M. Homolka, A. Blokhin, W. K. G. Assunção, E. Herac, and A. Egyed, "A tool for collaborative consistency checking during modeling," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24, Linz, Austria: ACM, 2024, pp. 655–659, ISBN: 9798400706226. DOI: 10.1145/3652620.3688558.

[52] I. David, H. Vangheluwe, and E. Syriani, "Model consistency as a heuristic for eventual correctness," *Journal of Computer Languages*, vol. 76, p. 101 223, 2023, ISSN: 2590-1184. DOI: https://doi.org/10.1016/j.cola.2023.101223.

[53] K. Dagenais and I. David, "Driving requirements evolution by engineers' opinions," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C*, ACM, 2024. DOI: 10.1145/3652620.3688566.

[54] S. J. I. Herzig, S. Mandutianu, H. Kim, S. Hernandez, and T. Imken, "Model-transformation-based computational design synthesis for mission architecture optimization," in *2017 IEEE Aerospace Conference*, 2017, pp. 1–15. DOI: 10.1109/AERO.2017.7943953.

[55] M. Földiák, "Probabilistic graph queries for design space exploration under uncertainty," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24, Linz, Austria: ACM, 2024, pp. 142–148, ISBN: 9798400706226. DOI: 10.1145/3652620.3688199.

[56] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: An analysis of the state of the research," *Soft. Sys. Mod.*, vol. 19, no. 1, pp. 5–13, 2020, ISSN: 1619-1374. DOI: 10.1007/s10270-019-00773-6.

[57] T. Molderez, B. Oeyen, C. De Roover, and W. De Meuter, "Marlon: A domain-specific language for multi-agent reinforcement learning on networks," in *Proc of the 34th ACM/SIGAPP Symposium on Applied Computing*, ACM, 2019, pp. 1322–1329. DOI: 10.1145/3297280.3297413.

[58] G. Kulagin, I. Ermakov, and L. Lyadova, "Ontology-based development of domain-specific languages via customizing base language," in *2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT)*, 2022, pp. 1–6. DOI: 10.1109/AICT55583.2022.10013619.

[59] H. Wu and J. Gray, "Automated generation of testing tools for domain-specific languages," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ACM, 2005, pp. 436–439. DOI: 10.1145/1101908.1101993.

[60] E. Syriani, D. Riegelhaupt, B. Barroca, and I. David, "Generation of custom textual model editors," *Modelling*, vol. 2, no. 4, pp. 609–625, 2021, ISSN: 2673-3951. DOI: 10.3390/modelling2040032.

[61] M. Ben Chaaben, O. Ben Sghaier, M. Dhaouadi, N. Elrasheed, I. Darif, I. Jaoua, B. Oakes, E. Syriani, and M. Hamdaqa, "Toward intelligent generation of tailored graphical concrete syntax," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '24, Linz, Austria: ACM, 2024, pp. 160–171. DOI: 10.1145/3640310.3674085.

[62] J. de Lara and E. Guerra, "Adaptive modelling languages: Abstract syntax and model migration," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 3, 2025, ISSN: 1049-331X. DOI: 10.1145/3702975.

[63] X. Liu and I. David, "A model-driven approach for developing families of reinforcement learning environments," in *2026 ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2026.

[64] B. Shajari, X. Liu, K. Dagenais, and I. David, "Trust the AI, doubt yourself: The effect of urgency on self-confidence in Human-AI interaction," in *2nd Workshop on Human-Centered AI for Software Engineering*, 2026.

[65] S. Abrahão, F. Bourdeleau, B. Cheng, S. Kokaly, R. Paige, H. Stöerrle, and J. Whittle, "User experience for model-driven engineering: Challenges and future directions," in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2017, pp. 229–236. DOI: 10.1109/MODELS.2017.5.

[66] I. David, M. Latifaj, J. Pietron, W. Zhang, F. Ciccozzi, I. Malavolta, A. Raschke, J.-P. Steghöfer, and R. Hebig, "Blended modeling in commercial and open-source model-driven software engineering tools: A systematic study," *Soft Sys Mod*, vol. 22, no. 1, pp. 415–447, 2023. DOI: 10.1007/s10270-022-01010-3.

[67] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987. DOI: https://doi.org/10.1016/0167-6423(87)90035-9.

[68] D. Kundu, D. Samanta, and R. Mall, "Automatic code generation from unified modelling language sequence diagrams," *IET Software*, vol. 7, no. 1, pp. 12–28, 2013. DOI: https://doi.org/10.1049/iet-sen.2011.0080.