Sense, Think, Act, Reflect: Distilling Fast and Interpretable Decision Functions from LLM-Driven Crowds

YICHI ZHANG, Nanyang Technological University, Singapore, Singapore
PHILIPP ANDELFINGER, Nanyang Technological University, Singapore, Singapore
WEN JUN TAN, Nanyang Technological University, Singapore, Singapore
WENTONG CAI, Nanyang Technological University, Singapore, Singapore

Large language models (LLMs) have been shown to be capable of generating human-like agent behavior in diverse scenarios, making them useful building blocks for agent-based simulations. However, the substantial inference cost restricts the crowd sizes that can be tackled, and the opaque nature of LLM-based decision making raises reliability concerns. To address these issues, we propose the approach of Decision Function Distillation (DFD), which extracts strategies underlying the decision making of LLM agents in a rule-based and interpretable form. The final decision function is determined in an iterative process during which intermediate insights gathered from historical agent trajectories are refined and finally translated into commented code. In two variants of the approach, the intermediate insights are either generated directly as text based on few-shot examples or explicitly formalized into code snippets. Unlike black-box symbolic regression (SR), the gradual and transparent refinement process allows modelers to understand the strategies captured in the final commented decision function. We demonstrate DFD on agent-based crowd evacuation scenarios, showing that DFD outperforms both classical and a state-of-the-art LLM-based SR.

CCS Concepts: • **Computing methodologies → Agent / discrete models**; **Natural language generation**; **Rule learning**.

Additional Key Words and Phrases: Crowd simulation, Large language model, Fast simulation, Symbolic regression, In-context learning, Social simulation

1 Introduction

Simulating crowd behavior during evacuations is essential for emergency planning and building design, e.g., for appropriate placement of emergency exits [38]. To capture the individual behaviors and interactions, agent-based modeling (ABM) [79, 84, 93, 94] simulates human behavior in a given environment according to specified agent attributes, behaviors, and mechanisms [29, 58]. Typically, the agents' behaviors must be manually defined for each scenario and are thus difficult to generalize across applications.

Recently, LLMs have attracted growing attention. LLMs are able to ingest and generate natural language, can reproduce a degree of world knowledge, and have demonstrated human-like reasoning ability in various tasks. Increasingly, LLMs play the role of agents in simulated environments by perceiving their environment through natural-language descriptions, making decisions, and interacting between individuals and the environment. While there are still known limitations such as the tendency for LLM-driven agents to behave overly homogeneously [3, 87], this approach has been successfully applied to task-driven decision making [72], social games [49], and crowd simulations [17].

Authors' Contact Information: Yichi Zhang, Nanyang Technological University, Singapore, Singapore; e-mail: yichi.zhang@ntu.edu.sg; Philipp Andelfinger, Nanyang Technological University, Singapore, Singapore; e-mail: philipp.andelfinger@ntu.edu.sg; Wen Jun Tan, Nanyang Technological University, Singapore, Singapore; e-mail: wjtan@ntu.edu.sg; Wentong Cai, Nanyang Technological University, Singapore, Singapore; e-mail: aswtcai@ntu.edu.sg.

ACM 1558-1195/2026/6-ART
https://doi.org/10.1145/3818686

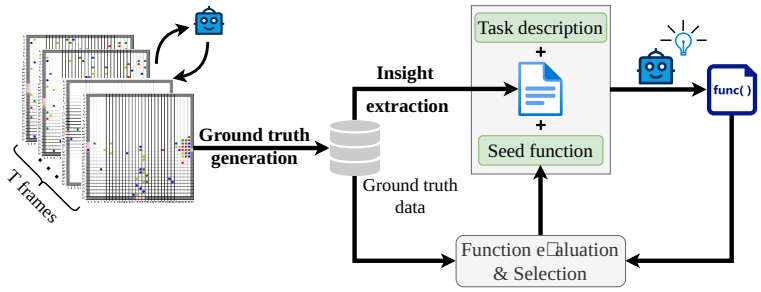


Fig. 1. Overview of our approach for distilling mechanistic decision functions from LLM-driven agent behavior

However, the rich behavior of LLM-driven agents and the reduced need for manual modeling come with two major drawbacks. Firstly, the language-based interaction and the underlying transformer-based architecture of LLMs entail enormous computational costs and memory demands [88]. Compared to traditional mechanistic simulations, this limits the scalability beyond relatively modest numbers of individuals. Secondly, while chain-of-thought prompting approaches can shed some light on an LLM's decision process, the underlying text generation remains a black box, making agent behaviors difficult to predict, validate, or interpret [46]. Although the validation of LLM-driven agent behaviors has been explored in some works [13, 95], systematic and rigorous validation approaches have not been established yet [87].

The aim of the present paper is to combine the capability of LLMs to flexibly react to an environment in a human-like manner with the speed and reliability of mechanistic agent models. To this end, we explore to what degree the rules that guide an LLM-driven agent's decision making can automatically be distilled into human-readable and commented functions. Our approach, which we refer to as **Decision Function Distillation (DFD)**, can be viewed as a form of symbolic regression that approximates LLM-generated ground truth decisions by a mechanistic function. Unlike existing work on LLM-based symbolic regression [73], DFD explicitly employs and formalizes domain knowledge to guide the gradual refinement and documentation of candidate decision functions. The final decision function can be integrated into a traditional mechanistic agent-based simulation framework.

As illustrated in Figure 1, DFD comprises three phases:

a **Ground Truth Generation.** An LLM-driven crowd simulation [88] generates pairs of the simulation state as perceived by agents and the resulting decisions.
b **Insight Extraction.** The ground truth data is analyzed to gradually extract language-based insights on the agents' decisions, which are optionally translated to an intermediate code-based representation.
c **Function Refinement.** Based on intermediate language or code-based insights, the LLM iteratively proposes decision function skeletons, whose coefficients are numerically optimized to fit the ground truth data.

We evaluate the fidelity of the generated decision functions against those generated by traditional and LLM-based symbolic regression using several qualitative and quantitative criteria.

Our contributions can be summarized as follows.

(1) We propose the approach of DFD, which tackles the computational cost of LLM-driven agent-based simulations by approximating the dialogue-based decision making of an LLM by an explainable and computationally inexpensive decision function.
(2) We propose an extension of DFD that augments the decision function refinement by converting insights on the agent behavior into intermediate commented code snippets.
(3) In an extensive evaluation, we show that DFD outperforms function generation using traditional symbolic regression and state-of-the-art LLM-based symbolic regression.

2 Related Work

Our approach provides a bridge between LLM-driven agent models and traditional mechanistic modeling. In the following, we briefly summarize existing work on the use of LLMs for crowd modeling and subsequently position our work in the existing literature on agent-based modeling, symbolic regression, explainable artificial intelligence, knowledge distillation, and insight generation from LLMs.

2.1 LLM-Driven Crowd Modeling

Agent-based crowd simulations are commonly constructed using the ABM paradigm [51], typically generating mobility in a two-dimensional space. In continuous spaces, Helbing's social force model [33] is often employed, whereas cellular automata-based models are used in discretized spaces (e.g., [70]). While models such as social force handle the so-called operational level of distance-keeping among members of a crowd, the higher-level decision making on the tactical level is task-specific and cannot easily be generalized.

Recently, many authors explored the use of LLMs to steer agents in various scenarios to leverage the rich general-purpose knowledge encoded in these models. A key advantage of this approach is that LLMs empower the simulated agents to flexibly carry out naturalistic reasoning and interactions [26, 36, 59]. However, directly immersing LLMs in a simulation [44, 88] raises the question of validity and reliability. Due to the black-box nature of LLMs, classical approaches to calibration and validation of ABMs cannot easily be applied. Furthermore, LLM-based agents lack the interpretability and repeatability of simulations based on mechanistic models [46].

Our decision function distillation approach aims to combine the benefits of LLM-driven agents with mechanistic ABM. In a form of symbolic regression [99], decision functions are determined from LLM-generated ground truth trajectories. In contrast to traditional symbolic regression, the LLM-driven iterative and insight-driven decision function distillation allows for a better fit and a higher degree of interpretability by generating commented code.

2.2 Accelerating Agent-based Simulations

Since the outcomes of agent-based simulations emerge from applying behavioral rules and interactions at the level of individuals in an overall agent population [32], the computational costs can be substantial. Approaches to reduce execution times of individual simulation runs substitute or simplify the original simulation model, or improve the efficiency and hardware utilization of the simulation engine used to exercise the model.

Surrogate modeling is a form of model reduction that substitutes an original model by a counterpart at a lower level of fidelity [34], often reducing the execution time substantially. Surrogate models are typically generated by sampling the original model in order to determine the relationships between model inputs and outputs. Successful amortization of the initial sampling cost depends on the complexity of the underlying input-output relationships and the fidelity requirements of the simulation study. Common model classes used as surrogates include Gaussian regression models and artificial neural networks [4].

Our distillation approach shares the data-driven approach of surrogate modeling. However, where surrogate modeling approximates the end-to-end relationship between simulation inputs and outputs, DFD frames the

distillation as a symbolic regression problem to identify per-agent decision-making processes, i.e., the mechanisms that lead to the overall simulation behavior. Hence, in place of largely opaque mappings from the simulation's input space to its output space, DFD generates a human-readable decision function that can serve as part of a mechanistic simulation model.

Outside of surrogate modeling, model reduction for agent-based simulations typically involves model-specific transformations to move to more coarse-grained and thus less computationally intensive representations, while maintaining adequate levels of fidelity [47, 57, 64, 103].

Approaches that target the *simulation engine* aim to reduce execution times by more efficient use of the hardware. Typically, these approaches preserve the original simulation's behavior. As an example, a common optimization in agent-based simulations is to employ spatial indexing structures to reduce the cost of identifying each agent's neighbors. A broad class of engine-level approaches parallelizes the agent state updates across multiple processing elements [14]. In recent years, methods for parallel agent-based simulations have moved from CPU-based approaches towards the use of hardware accelerators such as GPUs and FPGAs [90]. In the current implementation of our approach, the original LLM-driven simulation runs on a GPU, whereas the distilled decision function is evaluated on a CPU. Although the LLM's token generation is a highly parallelized procedure in itself [82], each LLM-driven agent update is several orders of magnitude slower compared to our distilled decision function. While out of scope for our present work, the lightweight nature of the distilled function would allow for straightforward parallelization across different agents' updates to further accelerate the simulation.

2.3 Symbolic Regression

Symbolic regression (SR) [43] is a class of approaches that aim to determine symbolic expressions that recreate the relation between a set of reference input-output pairs. Most SR approaches rely on sparse regression techniques or incrementally construct expression trees [52]. Sparse regression-based approaches generate weight coefficients to combine elements from a library of linear or non-linear functions. A common traditional method targeting dynamical systems is sparse identification of nonlinear dynamics (SINDy) [8], which fits the weight coefficients to construct an overall function that best matches the time derivatives of the reference system's state variables. While SINDy is limited to linear combinations of functions, employing the elementary functions from the library as the activation functions of neural networks allows complex nonlinearities to be captured [39]. A limitation of sparse regression is that the possible outcomes are restricted by the predefined function structure.

Constructing an expression tree from a library of operations allows for arbitrarily complex functions to be generated. Most approaches employ evolutionary algorithms, specifically genetic programming (GP) [52], to heuristically explore the enormous search space, which grows exponentially with the tree depth. While purely GP-based approaches are still among the state-of-the-art [20], neural networks are increasingly employed in this context. Recurrent neural networks are a natural architecture to model the conditional probabilities for operations to appear at each level of the expression tree [35, 61]. More recently, transformers [82] pre-trained on large numbers of example pairs of functions and generated datapoints have been used for this purpose. Constants that appear in the generated functions are either calibrated numerically in a separate step [6, 80] or included directly in the transformer's prediction [37, 81]. In combinations of neural and GP-based SR, an outer loop invokes a pre-trained transformer to generate seed functions, which are refined using GP in an inner loop [45, 56].

Recent work recognizes prior knowledge as a key ingredient to steer the search towards promising areas of the expression space. In unified deep SR [45], manually specified constraints on the possible symbols are encoded as numerical vectors and used to adjust the logits emitted by the neural network. An alternative is to identify prior probabilities in a data-driven way from datasets of domain-specific examples [35].

Due to their pre-training on text from various domains and their capability to extract insights from natural language, LLMs offer novel opportunities for integrating domain-specific prior knowledge. One approach is

to integrate LLMs in traditional SR pipelines to steer the search. This has been explored by instructing the LLM to choose the library of operations and the optimization algorithm for SINDy [55]. Another approach is to employ LLMs to score the generated functions according to pre-defined criteria such as physical realism [77]. By incorporating the score in the loss function used for traditional SR, the search is accelerated over a purely numerical evaluation.

A number of recent works explored the use of LLMs to directly generate candidate functions [30, 50, 53, 73, 91]. Our work takes inspiration from LLM-SR [73], which evolves disjoint sets of candidate functions in an iterative fashion. Subsequent work explored the prompting of LLMs to identify and describe regularities in the data as well as provide feedback to subsequent function generation rounds [85]. Data-aware insights are generated by analyzing the influence of the changes in the values of different independent variables on the dependent variable. Unlike this work, we prompt an LLM with concrete trajectory data to capture insights on the underlying behaviors. Further, in contrast to this existing work's focus on scientific equation discovery, instead of relying on ground truth data representing phenomena of the real world directly, we aim to capture LLM-driven crowd behaviors in mechanistic functions. To achieve this, we facilitate the function refinement using LLM-generated, explicit, and optionally formalized insights on observed decision-making patterns.

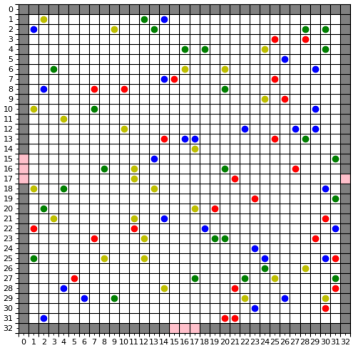
Finally, some existing works have applied SR for identifying the underlying rules that guide the behavior of populations of agents. SR via gene expression programming was applied to identify crowd movement rules from real-world data [100]. More recently, graph neural networks and a custom evolutionary algorithm were used to extract mathematical expressions for the behavior of animal swarms [63], inspired by seminal work targeting particle interactions [16]. However, we are not aware of any existing work on extracting behavior beyond immediate physical movement rules, targeting higher-level decision making.

2.4 Explainable Artificial Intelligence

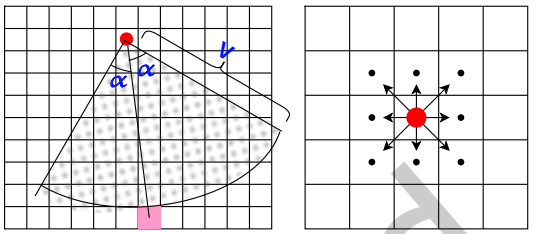
Explainable Artificial Intelligence (XAI) [22] refers to methods that allow humans to understand the predictions or behaviors of AI models. A broad categorization differentiates XAI methods by their focus on data, features, or models. Our work falls under feature-based methods as our symbolic decision functions are generated from feature-decision pairs produced by the LLM-driven ground truth model. However, in contrast to other methods in this category, our aim extends beyond isolated aspects of the feature-decision relationship such as feature importance (e.g., using Shapley values [54]) and beyond generating local explanatory surrogates around specific feature combinations (e.g., using LIME [65]). Instead, we generate a global surrogate that approximates the LLM-guided decision making in a human-interpretable form. Existing approaches usually construct interpretable models in the form of rules [5, 18] or decision trees [7]. In the context of automated driving, some imitation learning approaches apply inductive logic programming to generate logic formulae that capture ground truth driver behavior [48, 71]. In comparison to these works, our reliance on SR is less restrictive regarding the expressions that can be generated.

Some recent works applied SR to distill neural reinforcement learning policies into mathematical expressions [1, 48]. These works focus on control problems and low-level mobility in robotics rather than high-level decision making.

Only few works have employed LLMs to provide ground truth behavior or for identifying interpretable models. In retrieval-based generation, traditional rule mining was applied to identify the relationship between LLM-driven document retrievals and predictions [68]. Finally, in biology, LLM-driven SR using Google's FunSearch [66] has been applied to determine cognitive models from human and animal behavior [10]. Our work builds on LLM-SR, which has been shown to vastly outperform FunSearch in SR tasks [74]. Going beyond LLM-SR's implicit reliance on LLMs' priors, we explicitly generate intermediate insights from agent trajectories to facilitate the function generation.



(a) Grid environment.



(b) Left: visible area when estimating exit crowdedness, right: Movement directions and selection order.

Fig. 2. The considered grid-based simulation space, agent perception, and mobility.

2.5 Symbolic Knowledge Distillation

Knowledge distillation extracts knowledge from a large "teacher" model to a "student" model. Different from XAI methods, the focus is on model efficiency rather than interpretability. In the LLM context, a common goal is to approximate a large pre-trained model using a faster, smaller model [12, 19]. *Symbolic* knowledge distillation extracts knowledge graphs and other structured data [2]. A small number of works have considered the extraction of symbolic *rules* from LLMs in the form of decision trees or in task-specific rule languages [11, 41]. Fang et al. proposed a method to extract proof strategies from LLMs [23]. Successful LLM-generated proofs in natural language are translated into lemmas that can then be employed in a rule-based fashion by a symbolic theorem solver. Different from our problem setting, a correct proof is necessarily a sequence of steps whose individual correctness can be proven directly. In contrast, extracting rules that align with the less well-defined space of plausible crowd behaviors considered in our work calls for a data-driven and iterative approach. We are not aware of existing work on knowledge distillation from LLMs for ABM.

2.6 Insight Generation for Self-Prompting

Finally, in many recent works, LLMs extract underlying strategies from generated text [24, 25, 96, 101]. By including the strategies in future prompts, the LLM's capability to complete tasks is improved. In line with this idea, our approach iteratively extracts the exit choice strategies in natural language to guide the decision function distillation. However, the final output of our approach is a decision function in the form of commented and deterministically executable code.

3 Model Formulation

We consider building evacuation scenarios based on [84] as illustrated in Figure 2(a). The simulation space is a 33×33 cellular grid representing the interior of a building, each agent occupying one cell. There are exits spanning one to three cells at the left, bottom, and right. The environment state perceived by the agents comprises the occupation status in their immediate neighbor cells as well as each exit's distance and estimated crowdedness level.

The agent state comprises the current position and two static attributes that influence the decision making: the agent's level of physical ability and its mental condition. The mental condition is either 0 for "positive", or 1 for "negative", and the physical ability is either 0 for "positive and full of energy" or 1 for "negative and afraid of difficulties". To model different personas, we classify agents into four groups: red represents "mental = 0, physical = 0", green represents "mental = 1, physical = 0", yellow represents "mental = 0, physical = 1", and blue represents "mental = 1, physical = 1".

At each time step, a Bernoulli trial with a configurable probability determines whether an agent makes a new decision on its target exit. This allows for repeated reconsideration of the exit choice without introducing symmetries and oscillations among agents. As the final action, each agent moves to one of its surrounding eight cells or remains in its current cell. Since our focus is on the agents' high-level decision making, the subsequent low-level action is to deterministically choose the cell that minimizes the distance to the target exit. If an agent becomes stuck, it permanently switches from the greedy movement to following the shortest path computed via $A^\star$ [31].

We formalize the model as follows: Initially, N agents are placed uniformly at random on unobstructed grid cells. The exit cells are denoted $e_m \in \mathbf{E}$, with $\mathbf{E} = \{(15, 0), (16, 0), (17, 0), (0, 15), (0, 16), (0, 17), (17, 32)\}$. Adjacent exit cells are aggregated to a single exit direction q. The *perceived environment state* for an agent i at time t includes the crowdedness $w_t^i \in \{1, 2, 3, ..., 9\}$ near the agent, the perceived number of agents $l_{t,q}^i$ towards exit direction $q \in \{\text{left, right, bottom}\}$, the three exits comprised of 3, 1, *and* 3 cells, and the distance $d_{t,m}^i$ to each exit cell m. Once an exit has been chosen, the greedy action choice is made by choosing the closest cell to the exit from a list of unoccupied neighboring cells.

The perceived number of agents $l_{t,q}^i$ is estimated based on all agents inside a $\alpha = 45°$ cone towards the exit direction within a limited visible range V (cf. Figure 2(b)).

The *agent state* includes the current position (x_t^i, y_t^i), the physical ability c_i, and the mental condition h_i, which are determined before the simulation begins. The number of remaining agents is N_t, which decreases each time an agent escapes through one of the exits.

The simulation terminates after T time steps or when $N_t = 0$. The overall simulation state is:

$$\begin{aligned}
&\mathbf{P}_t(i, m, q) = \{\mathbf{X}_t, \mathbf{Y}_t, \mathbf{W}_t, \mathbf{C}, \mathbf{H}, \mathbf{L}_t, \mathbf{D}_t \mid t, i, m, q\},\\
&\quad \text{where } \mathbf{X}_t, \mathbf{Y}_t, \mathbf{W}_t, \mathbf{C}, \mathbf{H}, \mathbf{L}_t \textit{ and } \mathbf{D}_t \text{ are the vectorized } x_t^i, y_t^i, w_t^i, c_i, h_i,\\
&\quad l_{t,q}^i, d_{t,m}^i.\ \mathbf{X}_t, \mathbf{Y}_t, \mathbf{W}_t, \mathbf{C}, \mathbf{H} \in \mathbb{N}_0^{N_t}, \mathbf{L}_t \in \mathbf{R}^{N_t \times 3}, \mathbf{D}_t \in \mathbb{R}^{N_t \times 7}
\end{aligned}$$

Each state transition involves an initial check for successfully escaped agents, for which the corresponding vector elements are removed from $\mathbf{P}_t$. For each of the N_t agents, ordered by ascending distance from their current target exits, the agent decision logic generates an exit choice $\mathbf{e}_t^i \in \mathbf{E}$ if $t = 0$ or a Bernoulli trial is successful. Finally, an action choice $\mathbf{v}_t^i \in \{(-1, 0), (-1, -1), (0, -1), (0, 0), (0, 1), (1, 1), (1, 0), (1, -1), (-1, 1)\}$ is generated, choosing the unoccupied cell in the direction that minimizes the distance to the respective agent's target exit.

A state transition can be written as $\mathbf{P}_t \xrightarrow{\mathbf{V}_t} \mathbf{P}_{t+1}$. Note that for a given environment state, the action is fully determined by the exit choice.

As described in detail below, our symbolic regression task is to determine a function that maps from the numerical input data of perceived environment state and agent state to the scores for different exit cells.

4 Method

In the following, we present our method for distilling decision functions from LLM-generated agent behaviors. The section is organized according to the three main phases of ground truth generation, insight extraction, and decision function refinement.

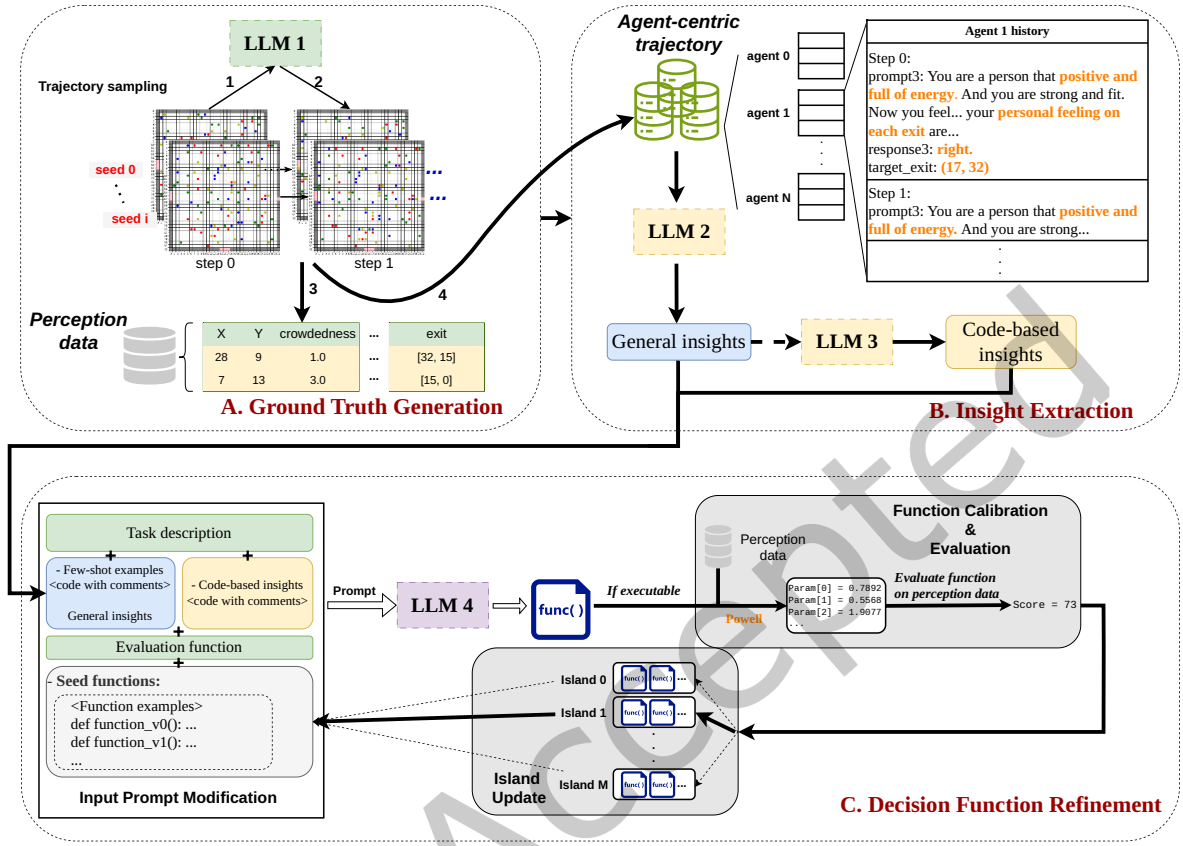


Fig. 3. Overview of the decision function distillation workflow.

4.1 Overview

The overall workflow is comprised of the three phases shown in Figure 3.

A. Ground Truth Generation. At each time step, the agents' perception data is passed to the LLM1 (1), which returns the exit choices according to which actions are taken (2). After the simulation terminates, the agents' trajectories are processed to form structured perception data (3). Further, the trajectories are also translated to an agent-centric form that represents the input and output of the decision making (4).

B. Insight Extraction. The LLM2 analyzes the agent-centric trajectories and identifies general behavioral rules, which we refer to as *general insights*. Between phase B and C, the identified general insights are optionally used as a basis for generating intermediate commented code snippets by LLM3, which will support the subsequent decision function refinement process.

C. Decision Function Refinement. Based on a structured prompt that includes the task description, insights, and evaluation function as well as the current set of candidate decision functions, the LLM4 iteratively proposes new candidate functions. The candidate functions are evaluated based on their fidelity in recreating the agents' decisions based on the perception data, updating the set of candidates whenever a new function outperforms all existing ones. Once a termination criterion is reached, the best-performing function is returned.

4.2 Ground Truth Generation

The ground truth data can originate from an existing model or real-world observations. In the present work, we rely on an existing work in which the agents' exit selection and immediate movement is generated using an LLM [88]. Their original simulation workflow comprises four phases: (a) A communication phase, in which the individuals communicate with each other to help with their information gathering and decision making. (b) A planning phase, in which agents decide which exit to target based on the perceived environment state, personal feelings, and gathered interactions. (c) An actuation phase, in which the agents take actions according to their previous decision. (d) An updating phase, in which the system determines the new status ("escaping" or "escaped") of the agents and writes to a log file. Since we are interested in high-level decision making, we focus on the exit selection and substitute the LLM-driven immediate movement by a simple heuristic, which substantially reduces the time required for data generation.

The simulation is executed several times for sampling trajectories $\tau_s = \{\mathbf{P}_0, \mathbf{V}_0, \mathbf{P}_1, \mathbf{V}_1, \mathbf{P}_2, \mathbf{V}_2, ..., \mathbf{P}_{u_s}, \mathbf{V}_{u_s}, \}$, where u is the trajectory length for a specific random number generator seed s, P and V represent the simulation state and action choice. The collected trajectories τ are stored as the training dataset. The input data in the symbolic regression task is the *perception data*, which is the tuple of numerical data $(x_t^i, y_t^i, w_t^i, c_i, h_i, \mathbf{d}_t^i, \mathbf{l}_t^i)$. The symbolic regression target is the exit choice e_t^i corresponding to each perception data point.

From the training dataset, we extract the *agent-centric trajectory data*, which describes the individuals' behavior in the form of inputs and outputs of the decision making. For an agent i, the trajectory includes the following information: physical ability c_i, mental condition h_i, history exit selection choice, surrounding conversations between individuals, the individual's feelings about the situation, about the different exits, and the textual representation of exit choice.

The individual's feelings are reflections of their static physical ability and mental condition under the current perceived environment state, as well as surrounding conversations and individual feelings obtained in the planning phase during the ground truth simulations. The interactions among agents reveal their individual exit choice and emotional state to nearby agents in natural language [88]. The agent-centric trajectory data combines the perception data with this textual data. Table 1 shows the features included in the perception data and agent-centric trajectory data.

4.3 Insight Generation

Given that an LLM is capable of analyzing past experiences to identify high-level patterns, extracting general behavioral rules can facilitate the generation of executable functions that replicate the LLM-driven agent behavior. Following [60], we refer to these general rules as *insights* defined as "meaningful and actionable information derived from raw data" that help agents to solve problems effectively. In the task considered in the present work, insights are meta-rules applicable to a range of situations. They are extracted from historical agent trajectories to explicate and enrich in-domain knowledge, supporting the decision function generation.

Table 1. Components of the perception data and agent-centric trajectory data. "Distances" are between agents and exit cells. "Exit crowdedness" is the perceived agent count around three exit directions.

Perception data (Numerical)	Target (Numerical)	Agent-centric trajectory data (Textual)
x, y coordinates	Exit choice	Crowdedness
Crowdedness		Mental
Capability		Historical exits
Mental		Conversations
Distances		Feelings about situation
Exit crowdedness		Feelings about exits
		Exit choice

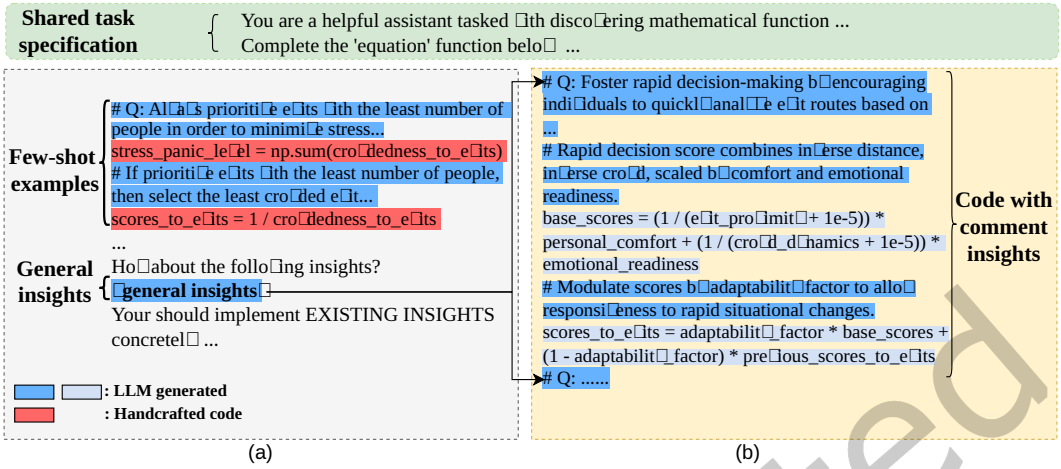


Fig. 4. DFD (a) bases the decision function refinement on handcrafted few-shot examples and LLM-generated general insights, whereas DFD-C (b) generates intermediate commented code snippets from general insights.

To extract initial high-level insights in natural language, we make use of the Expel approach [96], which employs a voting mechanism to gradually improve a finite set of candidate insights. However, the insights generated by Expel take the form of general observations rather than concrete rules suited for formalization. Due to the limited reasoning ability of current LLMs, translating general observations to accurate decision functions is a challenging problem. To facilitate the end-to-end translation from trajectories to decision functions, we clearly separate the insight generation from the function refinement. We distinguish two variants of our approach: DFD is prompted with handcrafted few-shot examples that indicate how insights can be translated to commented code, and directly generates decision functions from general insights. In contrast, similar to existing work in other problem domains [27, 42, 89], DFD-C (Decision Function Distillation - Code) introduces an additional step in which the LLM is instructed to generate intermediate commented code snippets that formalize partial insights. The code snippets are included in subsequent prompts to support the refinement of overall decision functions. In the following, we discuss the insight generation process in more detail. Figure 4 illustrates the two variants of our approach.

The key challenge the insight generation must tackle is to bridge the enormous gap between general high-level strategies and specific actions. For instance, the insight of "optimizing escape efficiency through clear communication and collaboration" indicates a general strategy but does not prescribe concrete actions. As a step towards a directly executable rule, a parameter *escape efficiency* could be introduced to quantify the perceived merit of each exit. Similarly, a strategic insight such as "Foster a proactive and anticipatory mindset during emergencies by continuously assessing surroundings, potential hazards,..." indicates important decision-making factors, but is too general to be applied directly. By formalizing the components of the insight and introducing numerical scores for aspects such as hazards, an overall decision function comes within closer reach. The specific prompts used for generating insights in natural language and commented code are shown in Appendix B.2.

4.4 Decision Function Refinement

In this final phase of our approach, the decision function is generated through an iterative refinement process. We take inspiration from LLM-SR [73], a closed-loop scientific equation discovery approach that exploits LLMs' inherent scientific prior knowledge and robust coding ability. Their core idea is to prompt an LLM to generate

function hypotheses, optimize and evaluate the candidate functions on a training dataset, and to evolve sets of potential functions iteratively. The approach employs an island-based evolutionary model [15, 67] from classic genetic programming [9, 76], which maintains several distinct populations of candidate functions in which functions can be updated or discarded depending on their fitness scores. While the selection of functions to retain is carried out in a rule-based manner, combination and mutation is performed opaquely by an LLM. Within each island, functions with identical scores are clustered. During subsequent sampling, this allows shorter functions among a set of functions with the same score to be preferred.

In contrast to LLM-SR's target area of scientific equations, our problem domain involves human behavior and interactions, which may be challenging for an LLM to infer based on its inherent knowledge and the provided data. Hence, while we follow LLM-SR in its basic principles, we facilitate the function refinement by prompting the LLM with the explicit, and optionally formalized, insights generated in the previous phase.

As shown in Figure 3, the function refinement phase consists of three parts: *island update*, *input prompt modification*, and *function calibration and evaluation*.

To initialize the overall procedure, λ independently evolving islands are initialized by the seed function ϕ_0, which outputs scores for the seven exit cells as a simple weighted sum of the features. The prompt used for function refinement consists of four parts: (a) task description, (b) few-shot examples with general insights or code-based insights, depending on the approaches, (c) evaluation function for function calibration, and (d) seed function. They are concatenated as input prompt during decision function refinement. At each round n, we input the aggregated prompt and get the candidate function from LLM.

The input prompt modification step involves selecting example functions from the islands to be included in the next prompt. The function sampling consists of two steps. First, one of the m islands i_n is sampled uniformly. Second, k functions are selected from the chosen island.

Since, as in LLM-SR, functions with identical scores are clustered together within each island, we first sample a cluster, then sample a function inside the cluster. The cluster is sampled using Boltzmann sampling, preferring higher scores by defining the sampling probability as $P_i = \frac{exp(s_i/\tau_i)}{\sum_j exp(s_j/\tau_j)}$ where s_i represents the average score for the i-th cluster. Here, $\tau_i = T_0(1 - \frac{u \bmod N}{N})$ with hyperparameters T_0, N, and u is the number of functions in the island. When sampling specific functions, shorter functions are preferred via $P(f_j) \propto exp(-l'_j)$ where l'_j is the function length normalized to $[0, 1]$. Finally, the selected k function samples are added as seed functions to the next prompt.

Once the decision functions have been sampled and non-executable functions discarded, calibration of any function coefficients and the evaluation of the function are carried out. At each round n, the coefficients in the decision function ϕ_n are initialized to all-ones, and the evaluation function ψ calculates the mean squared error s_n of ϕ_n between the positions of the predicted exit $\tilde{\mathbf{e}}_n^i$ and the true exit $\mathbf{e}_n^i$ over all training data points. If several exits share the same score, we prefer the first exit in counter-clockwise order, beginning from the top left. As the generated decision functions may not be differentiable, we adopt Powell's method [62] to adjust the coefficients in order to get the maximized $s_n^\star$. To avoid overfitting, we terminate once the relative change in the solution vector between iterations falls below 10^{-4}. $f_n^\star$ represents the optimized decision function consisting of ϕ_n and corresponding coefficients.

Calibrated functions that improve over their respective island's best scores are incorporated in the island: $\mathcal{F}_n^{i_n} \leftarrow \mathcal{F}_n^{i_n} \cup \{(\phi_n, s_n^\star) | s_n^\star > s_{i_n}^\star\}$ where $\mathcal{F}_n$ is the function set of source island i_n, $s_{i_n}^\star$ is the largest score in i_n. Subsequently, the islands' clusters are updated. The pseudocode of DFD is shown in Alg. 1.

As a final step, we simplify the generated decision function $f^\star$ by discarding parameters that do not appear in the function. Based on the obtained reduced feature set Σ', the parameters of the adjusted function skeleton $f^{\star\prime}$ are once again calibrated on the training dataset. The calibrated decision function is subsequently used to steer the simulated agents' decisions. As shown in Figure 5, at each time step, each agent invokes the decision function

Algorithm 1: Decision Function Distillation

Data: Round n, Maximum round $N \geq 1$, Collected ground truth τ including **P** and **A**, DFD $\mathcal{I}_f$, DFD-C $\mathcal{I}_c$, Evaluation function ψ, Local LLM $\mathcal{L}$, Island No. λ, Seed function ϕ_0, Island function sets $\mathcal{F}$, Insight type $\mathcal{T}$
Result: $f^\star, s^\star$
$\mathcal{F} = \{(\phi_0); (\phi_0); ...\}$, $n \leftarrow 1$;
$Prompt = Task\ Description$ `// cf. Fig. 4`
$f^\star, s^\star \leftarrow null, -\infty$;
`// Add insights and evaluation function`
$Prompt \leftarrow Prompt \cup \mathcal{I}_\mathcal{T} \cup \psi$
while $n \leq N$ **do**
  `// Sample k examples from` λ `islands`
  $E,\ i \leftarrow SampleFun(\mathcal{F}, k)$;
  $Prompt \leftarrow Prompt \cup E$;
  `// Feed into LLM to generate new hypotheses`
  $\phi_n \leftarrow \mathcal{L}(Prompt)$;
  if ϕ_n *is Executable* **then**
    `// Calibrate the parameters of function skeleton`
    $f_n^\star, s_n^\star \leftarrow FuncOptim(\phi_n)$;
    $\mathcal{F}_i \leftarrow IslandUpdate(\phi_n, s_n^\star)$;
    if $s_n^\star > s^\star$ **then**
      $f^\star, s^\star \leftarrow f_n^\star, s_n^\star$;
    end
  end
  $n \leftarrow n + 1$;
end

with probability p to obtain the exit choice. After determining the exit choice, the greedy policy or $A^\star$ routing algorithm is used to determine the action choice V_t. By enacting V_t, the simulation state transitions from P_t to P_{t+1}.

5 Experiments

We conducted extensive experiments to assess the effectiveness of DFD and DFD-C. After introducing the benchmark setup, we present qualitative and quantitative comparison results against several baseline methods on an in-domain and out-of-domain dataset. An ablation study explores the impact of each component of our approach.

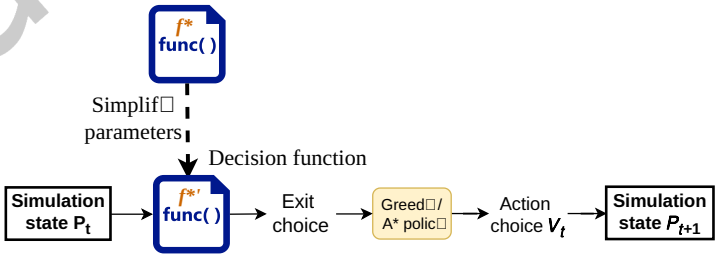


Fig. 5. In the final fast simulation, each agent uses the generated decision function to select its target exit. f^* to $f^{*'}$ represents the feature reduction. The diagram shows one simulation step of a single simulation run.

5.1 Benchmark Setup

Datasets. Our experiments are based on three datasets: an in-domain training and test dataset, and an out-of-domain test dataset, each of which is generated by the LLM-driven agent-based simulation from [88]. The out-of-domain test dataset is intended to evaluate whether the generated decision functions can generalize beyond the specific scenario layout of the training dataset. Each simulation was executed for a total of $T = 51$ timesteps. The number of participating agents was $N = 100$ in all experiments. The visible range V for each agent was set to 10 cells. The number of general and code-based insights was set to 20. For DFD, we handcrafted extra four few-shot example pairs of general insights and corresponding commented code. The number of independently evolving islands λ was set to 10.

The ground truth data is generated from simulations using five different random number seeds that define the individuals' starting positions. Constrained by the enormous execution time of the LLM-driven simulations, we used five different seeds for the in-domain training dataset, two seeds for the test dataset, and three seeds for the out-of-domain test dataset. The distilled functions are evaluated on the simulation data of the same sets of random seeds for training, test and generalization. In the following, we report the upper bound and lower bound across test seeds in the plot.

At each time step, there is one (perception data, exit choice) pair for each agent. The training dataset is comprised of 9955 such pairs, which are further split into a training and validation set in the ratio $4:1$. Different from the training dataset, the test dataset and the out-of-domain test dataset are composed of historical positions $(\mathbf{X}, \mathbf{Y})$ from simulation states of each agent for evaluating the similarities in the crowd behavior. All reported evaluation metrics are averaged across the different seeds.

Evaluation metrics. We evaluate the fidelity of the generated functions both qualitatively by visualizing crowd densities and evacuation times, and quantitatively according to the following metrics:

(1) To assess the raw prediction accuracy, we employ the mean square error (MSE) between positions of the predicted and true exit, which captures the fine-grained prediction loss. In our experiments on the training and test datasets, the loss values per exit prediction ranged between 0 and 32.06.
(2) To quantify the difference in crowd distribution, we compare the number of agents per subregion. As in [98], we split the initial 33×33 grid into subregions of size $\omega \times \omega$ with $3 \leq \omega \leq 11$. Zero padding is applied when the scenario cannot be evenly divided into blocks. The error metric is the mean absolute error (MAE) in the agent counts over all sub regions and time steps.
(3) To evaluate the similarity in crowd distributions with respect to the exits, we plot the number of agents escaped via the three exit directions over time. Dynamic Time Warping (DTW) [69] is employed to score the similarity between the time series.

Experiment setup. In our experiments, we leverage several different LLMs for ground truth generation, insight extraction, and decision function refinement. The ground truth data is generated using **Gemma3:27b** [78]. Observing that Gemma3 was insufficient for the subsequent steps, we used two different versions of GPT via the OpenAI API, choosing the smallest model that yielded well-structured results. The general insights for DFD and DFD-C were generated using **GPT-4o-mini**, whereas DFD-C employed **GPT-4.1-mini** to generate the intermediate insights as commented code. Finally, in accordance with [74], we use **Llama-3.1-8B-Instruct** in the decision function refinement phase for each method.

As baselines for the quantitative comparison, we use traditional symbolic regression (SR) using the PySR library [15] and the state-of-the-art LLM-based symbolic regression approach LLM-SR [73]. For fairness, we set PySR's iteration limit to roughly obtain the same execution time as our own approaches. However, PySR terminated after two days, compared to five days with DFD and DFD-C. PySR proposes functions at different levels of complexity, of which we select the most complex one, aiming to emphasize the fit to the data.

Table 2. Overview of approaches evaluated in our experiments.

		LLM-SR	LLM-SR-GI	DFD	DFD-P	DFD-C	DFD-M	DFD-CM
Insight Generation:	From task description		✓	✓	✓	✓	✓	✓
	From trajectories		✓	✓		✓	✓	✓
Function Generation:	Handcrafted insights as code			✓			✓	
	Intermediate insights as code					✓		✓
	Agent memory						✓	✓

Our evaluation includes four forms of ablation: Firstly, the comparison between DFD and DFD-C elucidates the benefits of generating intermediate code-level representations of insights. Secondly, we modify LLM-SR to leverage the same general insights as DFD and DFD-C (LLM-SR-GI), allowing us to gauge the level of fidelity reachable without our gradual function generation and refinement mechanisms.

Further, DFD-P generates insights purely from the task description to determine whether the LLM's prior knowledge suffices for successful insight generation. For fair comparison, we use the same LLM (GPT-4o-mini) to obtain insights directly from the task description, after which we use these insights for DFD-P. Finally, in DFD-M, we implement per-agent memory, creating stateful agents that recall the history of previous exit choices. DFD-CM corresponds to DFD with code-based intermediate insights and agent memory. Table 2 provides an overview of the considered approaches. Additional details on DFD-P, DFD-M, and DFD-CM are provided in Appendix B.1.

As a reference point for the computational cost of the DFD-generated decision functions, we employ the handcrafted evacuation model from [84].

5.2 Generated Decision Functions

Figure 6 shows the MSE loss representing the fit to the training data over the course of the function refinement process. All methods are able to gradually improve over the initial solution.

The best result is achieved by DFD-M.

We observe that even if LLM-SR is augmented with general insights (LLM-SR-GI), it is still not competitive with our approach. This indicates that our gradual translation from high-level insights to the final commented code facilitates the refinement of high-fidelity decision functions.

We now qualitatively assess the concrete decision function code generated by the different methods. Figure 7 lists the functions $f^{\star\prime}$ (cf. Section 4.4) after eliminating non-contributing features and calibrating their parameters to the training dataset.

As expected, the decision functions generated by the LLM-driven methods are more amenable to interpretation by humans. LLM-SR, LLM-SR-GI, DFD, and DFD-C all provide explicit explanations of the variables and expressions used in the function. The purely search-based PySR lacks this type of interpretability. Notably, the decision functions generated by DFD and DFD-C are constructed from documented strategies that indicate how the final decision is obtained.

Considering DFD-M, DFD-CM, in which the function distillation has access to a history of the agent's previous decisions, we observed that the final generated function did not make use of the history. The generated functions are roughly in line with those generated via DFD and DFD-C.

Comparing the structure of the functions between DFD and DFD-C emphasizes the methodological difference between the two variants. DFD, which relies on few-shot prompting as a basis for the translation from insights to decision functions, generated a function that compactly combines the parameters to determine an overall score.

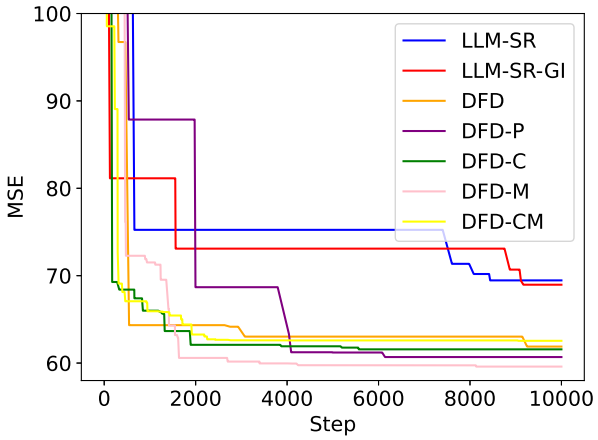


Fig. 6. MSE loss curve during decision function refinement.

In contrast, DFD-C, which translates insights into intermediate code-level representations, produced a decision function that progressively determines the score in a sequence of separately commented steps.

5.3 Crowd Densities

We first evaluate the fidelity of the agent behavior in terms of the crowd densities across the simulation space. Figure 8 compares the cumulative total number of agents that visited each cell over one simulation run. The ground truth shows an increased crowd density between the central area and the three exits. The highest density is observed near the exit on the right-hand side, which in contrast to the other exits spans only one cell.

Comparing the behavior generated by the different methods' decision functions, we observe that LLM-SR-GI, DFD, DFD-C, DFD-P, DFD-M and DFD-CM all correctly reproduce the tendencies in the ground truth. In contrast, PySR heavily overemphasizes evacuations through the bottom exit, whereas LLM-SR favors the exits on the right and bottom, without capturing the crowd density on the paths to the exits. As the model by Wang et al. was not crafted based on our LLM-based ground truth, the observed substantial deviation is expected.

Table 3 quantifies the differences in the individual counts in subregions of size 11×11. For each method, we show the result for the decision-making probability p that led to the best performance. With lower p, agents tend to adhere to their current exit choice for longer periods of time, which may prevent oscillations under changing scenario conditions and compensate for the memoryless nature of the decision functions. The results demonstrate that DFD and DFD-C consistently outperform the baseline methods PySR, LLM-SR, and LLM-SR-GI both on the in-domain training and test dataset and on the out-of-domain generalization dataset. We note that DFD-C is competitive with DFD even though in DFD-C, the function refinement phase purely relies on the previously generated intermediate code-based insights. In fact, it can be observed in Figure 7, that DFD repeatedly recites commented code from its original prompt verbatim, failing to convert the current insights into the corresponding code. In contrast, the function refinement in DFD-C can rely on numerous previously generated code-based insights, facilitating the refinement of the candidate functions.

DFD-P, which draws only on the LLM's prior knowledge to generate insights, performs worse than DFD in three datasets. This indicates that the consideration of agent trajectories improves the quality of the generated functions.

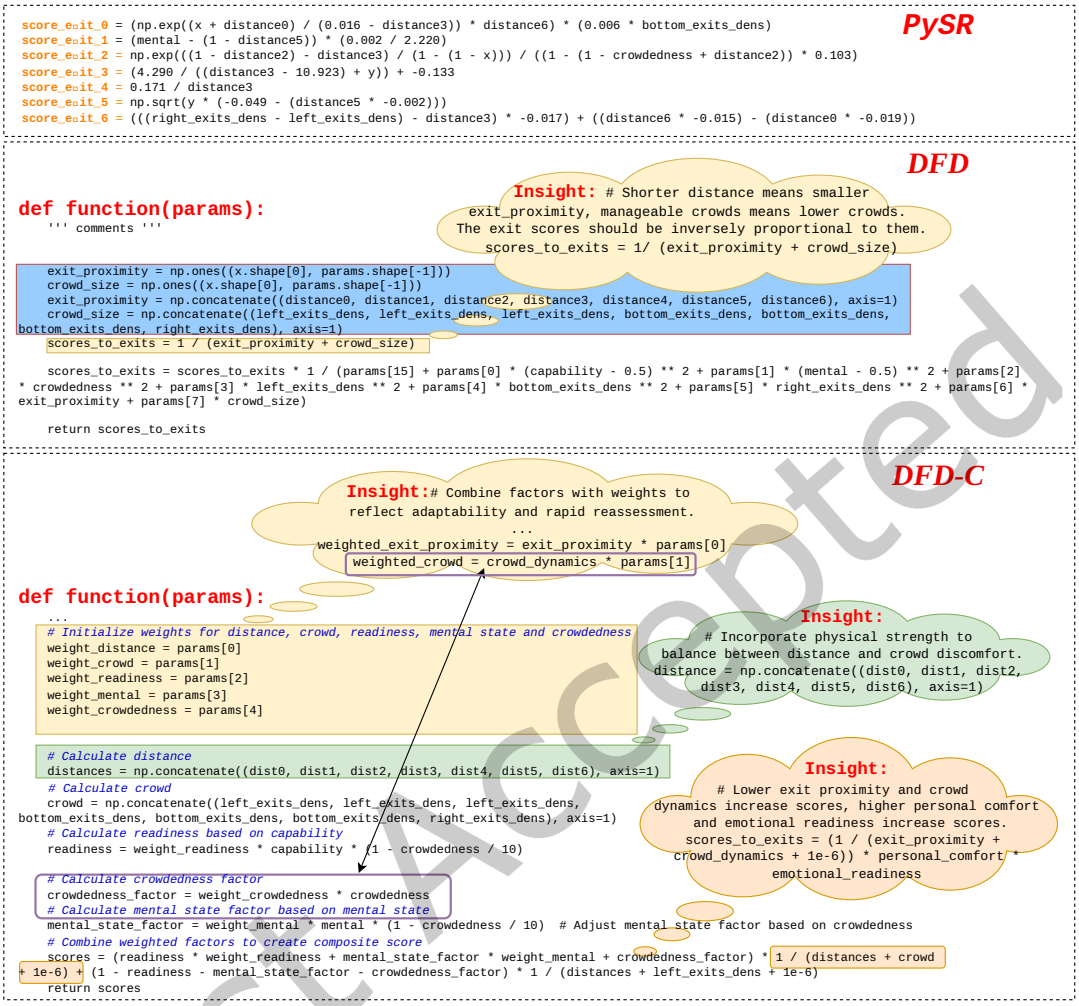
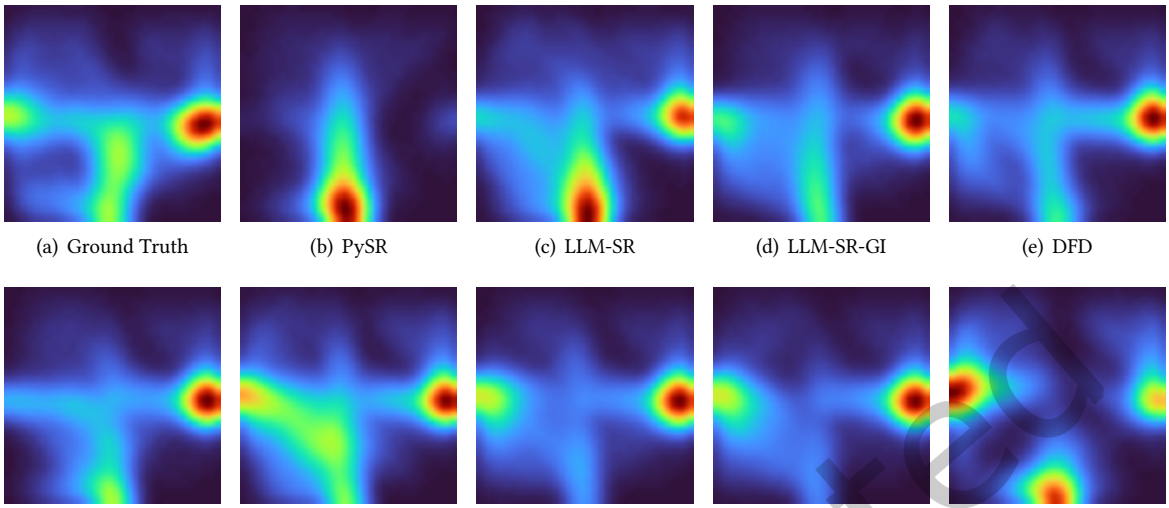


Fig. 7. Best-performing decision functions generated by PySR, DFD, and DFD-C.The clouds represent possible source insights for expressions used in the function. The text in the blue area was recited verbatim by DFD from the handcrafted examples.

To provide a more comprehensive evaluation, Figure 9 plots the MAE across different subregion sizes ω. In line with the previous result, DFD and DFD-C consistently outperform baseline methods on all datasets. An interesting observation is that MAE decreases as the size of subregion ω increases. Since deviations are counted on the level of subregions, more coarse-grained subregions lead to minor deviations being disregarded in the metric. More detailed results are provided in Appendix B.1.

5.4 Exit Choices

In Figure 10, we visualize the evacuations of test dataset via each of the exit directions over time. ADone.s the overall evacuation times differ between the methods, we quantify the curves' fidelities via Dynamic Time Warping



(a) Ground Truth (b) PySR (c) LLM-SR (d) LLM-SR-GI (e) DFD

(f) DFD-P (g) DFD-C (h) DFD-M (i) DFD-CM (j) Wang et al.

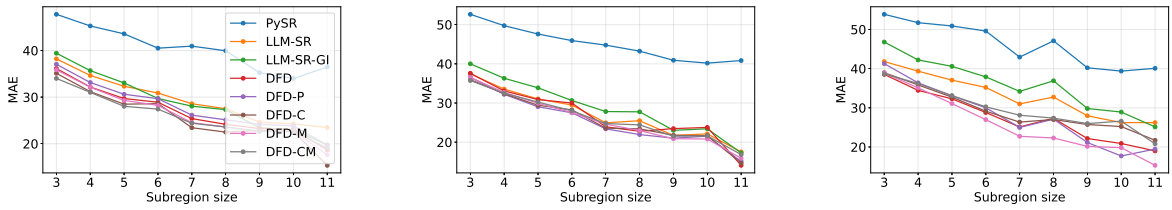
Fig. 8. Accumulated density maps comparing the different methods against the ground truth on test data.

Table 3. Performance comparison on crowd evacuation task. Lower absolute mean indicates better subregion density similarity. We also list the best-performing exit selection probability per step.

Model	Crowd evacuation			Probability		
	train↓	test↓	generalize↓	train	test	generalize
PySR	36.46	39.07	40.08	0.02	0.01	0.38
LLM-SR	23.48	19.61	26.21	0.05	0.04	0.04
LLM-SR-GI	19.71	17.79	25.13	0.01	0.01	0.08
DFD	18.67	**15.62**	19.01	0.06	0.04	0.49
DFD-P	19.73	17.42	19.47	0.06	0.05	0.19
DFD-C	**15.36**	15.64	21.72	0.04	0.04	0.12
DFD-M	17.63	16.63	**15.32**	0.08	0.04	0.33
DFD-CM	19.07	18.26	20.81	0.06	0.04	0.31

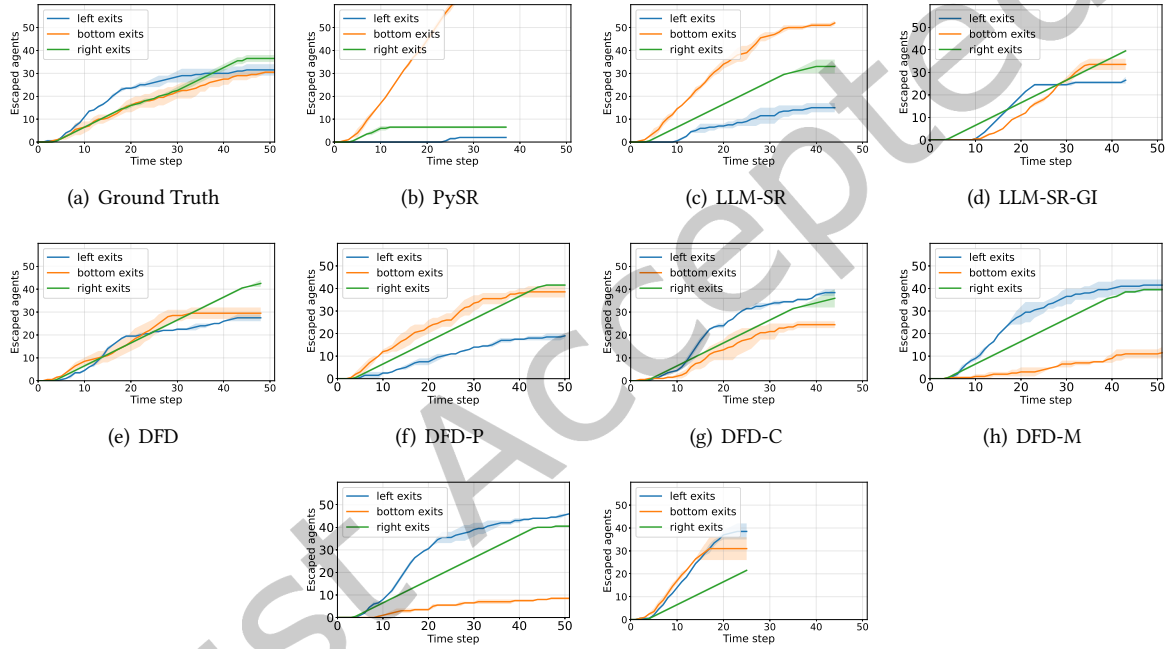
(DTW) [69], which obtains the minimum accumulated alignment cost along evacuation time (horizontal) and evacuated agents distribution (vertical). The results in Table 4 indicate that the DFD variants best capture the ground truth evacuation times and evacuated agents distribution both on the in-domain training and test dataset, as well as on the out-of-domain generalization dataset. The handcrafted model by Wang et al. results in faster evacuation, but does not accurately capture the behavior of our ground truth model.

To summarize, we observed that our methods can distill the ground truth data into commented and fast decision functions that closely represent the original LLM-driven agent behavior. By comparing to LLM-based symbolic regression, we observed that introducing contextual information through our insight-based refinement leads to more accurate decision functions. Moreover, the generation of code-level insights as an intermediate step led to decision functions that were more clearly structured into sequences of commented steps.



(a) Train (b) Test (c) Generalization

Fig. 9. MAE with different subregion sizes ω.



(a) Ground Truth (b) PySR (c) LLM-SR (d) LLM-SR-GI

(e) DFD (f) DFD-P (g) DFD-C (h) DFD-M

(i) DFD-CM (j) Wang et al.

Fig. 10. Comparison of the number of escaped agents at the exits across the approaches. The solid lines are averages, and the shadow around the lines indicates the range. The overall evacuation time varies across the decision functions.

5.5 Computational Cost

While the decision function distillation is a time-consuming process, the subsequent simulations do not rely on an LLM and are thus accelerated by several orders of magnitudes. A comparison to the handcrafted model by Wang et al. showed execution times in the same range of 1 to 2 seconds per time step.

In the following, we briefly describe the components of the overall time spent to identify when amortization of the distillation cost is achieved. The execution times are denoted as follows: $t_{\text{sim,LLM}}$ and $t_{\text{sim,DF}}$ for an LLM-driven or decision function-driven simulation run, t_{insight} for the insights generation, and t_{distill} for the function

Table 4. DTW loss for training, test and generalization datasets (lower is better).

Model	DTW loss		
	train↓	**test↓**	**generalization↓**
PySR	171.38	184.93	118.92
LLM-SR	66.13	55.36	12.00
LLM-SR-GI	18.04	5.19	11.02
DFD	**4.65**	**2.87**	**1.11**
DFD-P	10.45	30.99	6.86
DFD-C	18.45	9.48	4.01
DFD-M	23.73	39.19	1.97
DFD-CM	19.11	49.63	13.83
Wang et al.	5.35	5.15	39.38

distillation. The time required for the trajectory generation is $N_{\text{traj}} \cdot t_{\text{sim,LLM}}$, where N_{traj} is the number of reference simulation runs. If N_{sim} runs of the final simulation model are to be carried out, amortization requires

$$N_{\text{traj}} \cdot t_{\text{sim,LLM}} + t_{\text{insight}} + t_{\text{distill}} + N_{\text{sim}} \cdot t_{\text{sim,DF}} < N_{\text{sim}} \cdot t_{\text{sim,LLM}}$$

Hence, the required number of simulations is

$$N_{\text{sim}} > \frac{N_{\text{traj}} \cdot t_{\text{sim,LLM}} + t_{\text{insight}} + t_{\text{distill}}}{t_{\text{sim,LLM}} - t_{\text{sim,DF}}}$$

In our measurements, $N_{\text{traj}} = 5$, $t_{\text{sim,LLM}} \approx 1\text{d}$, $t_{\text{insight}} \approx 1\text{m}$, $t_{\text{distill}} \approx 5\text{d}$, and $t_{\text{sim,DF}} \approx 1\text{m}$, and thus $N_{\text{sim}} > \sim 10$.

6 Discussion

Our experiments showed the successful distillation of fast and interpretable decision functions from an LLM-driven crowd evacuation model. In the following, we discuss the implications of our reliance on LLM-driven reference data, the opportunities that alternative data sources may present, and the generalizability of the approach.

6.1 Validity of Agent Behavior

As our approach is agnostic of the realism of the ground truth trajectories, the validity of the distilled function depends directly on the reference data. The realism of LLM-driven models of human behavior is the subject of active research, with many authors voicing concerns regarding reliability and thus trust in the models [75, 92, 102]. Our work contributes to this field of inquiry by allowing candidate models to be "frozen" and scrutinized for their validity in the form of interpretable and deterministically executable decision functions.

Since the focus of our paper is on the distillation methodology, we relied on the simulation environment and LLM prompts from an existing study focused on LLM agents' capability to generate human-like behavior directly based on environmental cues [88]. The underlying scenario and definition of the simulation space stems from Wang et al.'s work on evacuation under panic states [84]. As the area of LLM-driven agent modeling progresses, studying the degree to which our method can maintain the validity of complex LLM-driven behaviors will be an important avenue for future work.

6.2 Alternative Sources for Ground Truth

As a consequence of our symbolic regression-based approach, we do not require the reference trajectories to be generated by LLM-driven agents. For instance, reference trajectories could also be generated by a traditional

handcrafted mechanistic agent-based model. In this setting, our distillation method is a form of model reduction, translating between mechanistic models. By extracting the "essence" of the existing model, potential benefits may lie in reduced execution times and in complexity, facilitating insights into the decisive factors for observed behavior.

An interesting alternative is to distill decision functions directly from trajectories extracted from real-world observations, e.g., from video footage, possibly augmented by individual-level context information. Where our present approach can be regarded as model distillation, surrogate modeling, or model reduction (cf. Section 2), foregoing the intermediate stage of LLM-driven simulation would more directly support the process of model creation. However, an important benefit of an intermediate LLM-driven model lies in its capability to generate plausible human-like behavior with respect to agent decisions for which data is sparse or unavailable, and to justify these decisions in natural language.

6.3 Generalization beyond Crowd Models

The applicability of our distillation approach rests on two assumptions, one pertaining to the structure of the trajectory data and the generated decision functions, and one pertaining to the LLM's pre-training.

A structural assumption of our distillation approach is that agents follow a sense-think-act cycle [86], in line with common high-level agent models such as belief-desire-intention [28]. In the *sense* stage, the agent observes the state of its surrounding environment, which may include nearby agents. Based on the observed situation, the *think* stage involves deliberations on actions to take. Finally, the chosen actions are carried out in the *act* stage. Our approach relates partial environment states observed in the *sense* stage to actions taken in the *act* stage in order to infer the deliberations that occurred in the *think* stage. Since the sense-think-act cycle is closely linked to the notion of "agents" [86], our method is not restricted to specific types of agent-based models.

Of course, successful inference of the agents' decision making requires a level of understanding of the types of considered agents and the environment in which they are situated. While the prompts used in our experiments provided a general description of these aspects, the distillation approach is beneficial only if the LLM succeeds in determining the relation between environmental features and the actions taken without requiring detailed scenario-specific prompting. Hence, the quality of the inferred decision functions may vary with the problem domain. As LLMs are commonly pre-trained mostly on human-written text, domains involving human decision making can be expected to be particularly favorable. In more specialized areas such as biology and chemistry, each agent may represent a cell or molecule, and an emergent overall behavior may depend on intricate agent interactions being modeled at high fidelity. Here, the "commonsense knowledge" [97] of current LLMs may not suffice to identify the underlying relationships. Fine-tuning based on suitable domain-specific corpora could improve the coverage of such specialized domains.

Beyond these general considerations, applying DFD to a new scenario requires problem-specific prompts to be adjusted as follows:

- **Insight generation**. The specification of the task the agents should complete, existing insights as well as the optimization objective are tailored to the scenario. In addition, the agent-centric trajectory data is inserted into the prompt (cf. Figure 17).
- **Generation of code-based insights**. A small number of handcrafted few-shot pairs of natural language insights and their counterparts in code should be provided (cf. Figure 18).
- **Decision function refinement**. A scenario description in accordance with the insight generation prompt should be provided (cf. Figure 4). The evaluation and seed function (cf. Figure 13) should be adapted to reflect the feature and action spaces of the agents' decision making.

All other prompts and procedures generically apply independently of the considered scenario.

7 Conclusions

We presented DFD, an approach to distill mechanistic decision functions from LLM-based agent simulations. Unlike existing symbolic regression approaches, DFD employs in-domain contextual information to gradually refine generated candidate functions. In our experimental results on emergency evacuation scenarios, the DFD-generated decision functions more closely approximate the ground truth data. In addition to outperforming traditional and LLM-based symbolic regression according to several quantitative evaluation criteria, DFD's commented decision functions also provide human-readable explanations of the agents' behavior.

Extensive ablation experiments demonstrated the benefit of generating intermediate insights from the ground truth data as a basis for the decision function generation. Further, formalizing insights into code intermediate snippets led to final decision functions that were more clearly structured as sequences of distinct commented steps. We observed that in our experiments, making a history of previous decisions available as a feature did not improve the quality of the distilled decision functions. Future work could explore the use of such memory mechanisms for scenarios in which path dependencies on previous decisions dominate, e.g., outcomes of past trades in financial models.

Encoding context information on agent interactions as features could further increase the fidelity with respect to the ground truth behavior. To move beyond scenarios that can be tackled by isolated decision functions, the generated functions could be integrated into more comprehensive mechanistic models structured according to principled decision-making frameworks such as recognition-primed decision [40]. This would allow the approach to be extended towards more complex and dynamic scenarios that require nuanced and multi-step decision-making processes.

Acknowledgments

This research is supported in part by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031) and in part by MoE AcRF Tier 1 Grant RG22/25.

References

[1] Fernando Acero and Zhibin Li. 2024. Distilling reinforcement learning policies for interpretable robot locomotion: Gradient boosting machines and symbolic regression. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6840–6847.

[2] Kamal Acharya, Alvaro Velasquez, and Houbing Herbert Song. 2024. A survey on symbolic knowledge distillation of large language models. *IEEE Transactions on Artificial Intelligence* (2024).

[3] Gati V Aher, Rosa I Arriaga, and Adam Tauman Kalai. 2023. Using large language models to simulate multiple humans and replicate human subject studies. In *International conference on machine learning*. PMLR, 337–371.

[4] Claudio Angione, Eric Silverman, and Elisabeth Yaneske. 2022. Using machine learning as a surrogate model for agent-based simulations. *Plos one* 17, 2 (2022), e0263150.

[5] M Gethsiyal Augasta and Thangairulappan Kathirvalavakumar. 2012. Reverse engineering the neural networks for rule extraction in classification problems. *Neural processing letters* 35, 2 (2012), 131–150.

[6] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. 2021. Neural symbolic regression that scales. In *International Conference on Machine Learning*. Pmlr, 936–945.

[7] Olcay Boz. 2002. Extracting decision trees from trained neural networks. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 456–461.

[8] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. 2016. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences* 113, 15 (2016), 3932–3937.

[9] Erick Cantú-Paz et al. 1998. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis* 10, 2 (1998), 141–171.

[10] Pablo Samuel Castro, Nenad Tomasev, Ankit Anand, Navodita Sharma, Rishika Mohanta, Aparna Dev, Kuba Perlin, Siddhant Jain, Kyle Levin, Noémi Éltető, et al. 2025. Discovering symbolic cognitive models from human and animal behavior. *bioRxiv* (2025), 2025–02.

[11] Tao Chen, Luxin Liu, Xuepeng Jia, Baoliang Cui, Haihong Tang, and Siliang Tang. 2022. Distilling Task-specific Logical Rules from Large Pre-trained Models. *arXiv preprint arXiv:2210.02768* (2022).

[12] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2024. MINI-LLM: Memory-Efficient Structured Pruning for Large Language Models. *arXiv preprint arXiv:2407.11681* (2024).
[13] Yun-Shiuan Chuang, Krirk Nirunwiroj, Zach Studdiford, Agam Goyal, Vincent V. Frigo, Sijia Yang, Dhavan V. Shah, Junjie Hu, and Timothy T. Rogers. 2024. Beyond Demographics: Aligning Role-playing LLM-based Agents Using Human Belief Networks. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 14010–14026. doi:10.18653/v1/2024.findings-emnlp.819
[14] Nicholson Collier and Michael North. 2013. Parallel agent-based simulation with repast for high performance computing. *Simulation* 89, 10 (2013), 1215–1235.
[15] Miles Cranmer. 2023. Interpretable machine learning for science with PySR and SymbolicRegression. jl. *arXiv preprint arXiv:2305.01582* (2023).
[16] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. 2020. Discovering symbolic models from deep learning with inductive biases. *Advances in neural information processing systems* 33 (2020), 17429–17442.
[17] Pei Dang, Jun Zhu, Weilian Li, Yakun Xie, and Heng Zhang. 2025. Large-language-model-driven agents for fire evacuation simulation in a cellular automata environment. *Safety Science* 191 (2025), 106935. doi:10.1016/j.ssci.2025.106935
[18] Enric Junque De Fortuny and David Martens. 2015. Active learning-based pedagogical rule extraction. *IEEE transactions on neural networks and learning systems* 26, 11 (2015), 2664–2677.
[19] Flavio Di Palo, Prateek Singhi, and Bilal Fadlallah. 2024. Performance-guided llm knowledge distillation for efficient text classification at scale. *arXiv preprint arXiv:2411.05045* (2024).
[20] Junlan Dong and Jinghui Zhong. 2025. Recent Advances in Symbolic Regression. *Comput. Surveys* 57, 11 (2025), 1–37.
[21] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv e-prints* (2024), arXiv–2407.
[22] Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Rana Omer, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, et al. 2023. Explainable AI (XAI): Core ideas, techniques, and solutions. *ACM computing surveys* 55, 9 (2023), 1–33.
[23] Jian Fang, Yican Sun, and Yingfei Xiong. 2025. Proof Strategy Extraction from LLMs for Enhancing Symbolic Provers. *arXiv preprint arXiv:2510.10131* (2025).
[24] Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems* 37 (2024), 119919–119948.
[25] Chang Gao, Haiyun Jiang, Deng Cai, Shuming Shi, and Wai Lam. 2024. Strategyllm: Large language models as strategy generators, executors, optimizers, and evaluators for problem solving. *Advances in Neural Information Processing Systems* 37 (2024), 96797–96846.
[26] Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. 2024. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications* 11, 1 (2024), 1–24.
[27] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*. PMLR, 10764–10799.
[28] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. 1998. The belief-desire-intention model of agency. In *International workshop on agent theories, architectures, and languages*. Springer, 1–10.
[29] Nigel Gilbert and Pietro Terna. 2000. How to build and use agent-based models in social science. *Mind & Society* 1, 1 (2000), 57–72.
[30] Arya Grayeli, Atharva Sehgal, Omar Costilla Reyes, Miles Cranmer, and Swarat Chaudhuri. 2024. Symbolic Regression with a Learned Concept Library. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=B7S4jJGlvl
[31] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107. doi:10.1109/TSSC.1968.300136
[32] Dirk Helbing. 2012. Agent-based modeling. In *Social self-organization: Agent-based simulations and experiments to study emergent social behavior*. Springer, 25–70.
[33] Dirk Helbing and Peter Molnar. 1995. Social force model for pedestrian dynamics. *Physical review E* 51, 5 (1995), 4282.
[34] L Jeff Hong and Xiaowei Zhang. 2021. Surrogate-based simulation optimization. In *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*. INFORMS, 287–311.
[35] Sikai Huang, Yixin Berry Wen, Tara Adusumilli, Kusum Choudhary, and Haizhao Yang. 2025. Parsing the Language of Expression: Enhancing Symbolic Regression with Domain-Aware Symbolic Priors. *arXiv preprint arXiv:2503.09592* (2025).
[36] Cameron Robert Jones, Ishika Rathi, Sydney Taylor, and Benjamin K Bergen. 2025. People cannot distinguish GPT-4 from a human in a Turing test. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency*. 1615–1639.
[37] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. 2022. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems* 35 (2022), 10269–10281.

[38] Nurulaqilla Khamis, Hazlina Selamat, Fatimah Sham Ismail, Omar Farouq Lutfy, Mohamad Fadzli Haniff, and Ili Najaa Aimi Mohd Nordin. 2020. Optimized exit door locations for a safer emergency evacuation using crowd evacuation model and artificial bee colony optimization. *Chaos, Solitons & Fractals* 131 (2020), 109505.

[39] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin Soljačić. 2020. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE transactions on neural networks and learning systems* 32, 9 (2020), 4166–4177.

[40] G.A. Klein. 1993. A recognition-primed decision (RPD) model of rapid decision making. In *Decision Making in Action: Models and Methods*, G.A. Klein, Judith Orasanu, R. Calderwood, and Caroline E. Zsambok (Eds.). Norwood: Ablex Publishing Corporation, 138–147.

[41] Ricardo Knauer, Mario Koddenbrock, Raphael Wallsberger, Nicholas M Brisson, Georg N Duda, Deborah Falla, David W Evans, and Erik Rodner. 2025. 'Oh LLM, I'm Asking Thee, Please Give Me a Decision Tree': Zero-Shot Decision Tree Induction and Embedding with Large Language Models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 1196–1206.

[42] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[43] Gabriel Kronberger, Bogdan Burlacu, Michael Kommenda, Stephan M Winkler, and Michael Affenzeller. 2024. *Symbolic regression*. Chapman and Hall/CRC.

[44] So Kuroki, Yingtao Tian, Kou Misaki, Takashi Ikegami, Takuya Akiba, and Yujin Tang. 2025. Reimagining ABM with LLM Agents via Shachi. In *ICML 2025 Workshop on Computer Use Agents*. https://openreview.net/forum?id=VtobeHcQWs

[45] Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K Petersen. 2022. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems* 35 (2022), 33985–33998.

[46] Maik Larooij and Petter Törnberg. 2025. Do large language models solve the problems of agent-based modeling? a critical review of generative social simulations. *arXiv preprint arXiv:2504.03274* (2025).

[47] Ping Liu, CI Siettos, C William Gear, and IG Kevrekidis. 2015. Equation-free model reduction in agent-based computations: Coarse-grained bifurcation and variable-free rare event analysis. *Mathematical Modelling of Natural Phenomena* 10, 3 (2015), 71–90.

[48] Wenliang Liu, Danyang Li, Erfan Aasi, Roberto Tron, and Calin Belta. 2025. Interpretable imitation learning via generative adversarial STL inference and control. In *Proc. Int. l Conf. Neuro-Symbolic Syst.* 472–489.

[49] Ziyi Liu, Abhishek Anand, Pei Zhou, Jen-tse Huang, and Jieyu Zhao. 2024. InterIntent: Investigating Social Intelligence of LLMs via Intention Understanding in an Interactive Game Context. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 6718–6746. doi:10.18653/v1/2024.emnlp-main.383

[50] Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. 2024. LLM and Simulation as Bilevel Optimizers: A New Paradigm to Advance Physical Scientific Discovery. In *International Conference on Machine Learning*. PMLR.

[51] Charles M Macal and Michael J North. 2009. Agent-based modeling and simulation. In *Proceedings of the 2009 winter simulation conference (WSC)*. IEEE, 86–98.

[52] Nour Makke and Sanjay Chawla. 2024. Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review* 57, 1 (2024), 2.

[53] Matteo Merler, Katsiaryna Haitsiukevich, Nicola Dainese, and Pekka Marttinen. 2024. In-Context Symbolic Regression: Leveraging Large Language Models for Function Discovery. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, Xiyan Fu and Eve Fleisig (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 427–444. doi:10.18653/v1/2024.acl-srw.49

[54] Luke Merrick and Ankur Taly. 2020. The explanation game: Explaining machine learning models using shapley values. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, 17–38.

[55] Christopher E Mower and Haitham Bou-Ammar. 2025. Al-khwarizmi: Discovering physical laws with foundation models. *arXiv preprint arXiv:2502.01702* (2025).

[56] T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. 2021. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053* (2021).

[57] Joseph S Niedbalski, Kun Deng, Prashant G Mehta, and Sean Meyn. 2008. Model reduction for reduced order estimation in traffic models. In *2008 American Control Conference*. IEEE, 914–919.

[58] Xiaoshan Pan, Charles S Han, Ken Dauber, and Kincho H Law. 2007. A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *Ai & Society* 22, 2 (2007), 113–132.

[59] Joon Sung Park, Lindsay Popowski, Carrie Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2022. Social simulacra: Creating populated prototypes for social computing systems. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–18.

[60] Alberto Sánchez Pérez, Alaa Boukhary, Paolo Papotti, Luis Castejón Lozano, and Adam Elwood. 2025. An LLM-Based Approach for Insight Generation in Data Analysis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Luis Chiruzzo, Alan Ritter, and Lu Wang (Eds.). Association for Computational Linguistics, Albuquerque, New Mexico, 562–582. doi:10.18653/v1/2025.naacl-long.24

[61] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. 2019. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871* (2019).

[62] Michael JD Powell. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal* 7, 2 (1964), 155–162.

[63] Stephen Powers. 2025. *Symbolic Model Extraction of Collective Behaviors in Homogeneous and Heterogeneous Swarms.* Ph. D. Dissertation. Vrije Universiteit Amsterdam.

[64] David M Rhodes, Mike Holcombe, and Eva E Qwarnstrom. 2016. Reducing complexity in an agent based reaction model—benefits and limitations of simplifications in relation to run time and system level output. *Biosystems* 147 (2016), 21–27.

[65] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[66] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (2024), 468–475.

[67] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2023. Mathematical discoveries from program search with large language models. *Nature* (2023). doi:10.1038/s41586-023-06924-6

[68] Joel Rorseth, Parke Godfrey, Lukasz Golab, Divesh Srivastava, and Jarek Szlichta. 2025. Rule-Based Explanations for Retrieval-Augmented LLM Systems. *arXiv preprint arXiv:2510.22689* (2025).

[69] Hiroaki Sakoe. 1971. Dynamic-programming approach to continuous speech recognition. In *1971 Proc. the International Congress of Acoustics, Budapest.*

[70] Caesar Saloma, Gay Jane Perez, Giovanni Tapang, May Lim, and Cynthia Palmes-Saloma. 2003. Self-organized queuing and scale-free behavior in real escape panic. *Proceedings of the National Academy of Sciences* 100, 21 (2003), 11947–11952.

[71] Iman Sharifi, Mustafa Yildirim, and Saber Fallah. 2025. Symbolic imitation learning: From black-box to explainable driving policies. *Applied Sciences* 15, 23 (2025), 12464.

[72] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 8634–8652.

[73] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. 2025. LLM-SR: Scientific Equation Discovery via Programming with Large Language Models. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=m2nmp8P5in

[74] Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K. Reddy. 2025. LLM-SRBench: A New Benchmark for Scientific Equation Discovery with Large Language Models. In *Forty-second International Conference on Machine Learning*. https://openreview.net/forum?id=SyQPiZJVWY

[75] Patrick Taillandier, Jean Daniel Zucker, Arnaud Grignard, Benoit Gaudou, Nghi Quang Huynh, and Alexis Drogoul. 2025. Integrating llm in agent-based social simulation: Opportunities and challenges. *arXiv preprint arXiv:2507.19364* (2025).

[76] Reiko Tanese. 1989. *Distributed genetic algorithms for function optimization.* University of Michigan.

[77] Bilge Taskin, Wenxiong Xie, and Teddy Lazebnik. 2025. Knowledge Integration for Physics-informed Symbolic Regression Using Pre-trained Large Language Models. *arXiv preprint arXiv:2509.03036* (2025).

[78] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786* (2025).

[79] Anne Templeton, Hui Xie, Steve Gwynne, Aoife Hunt, Pete Thompson, and Gerta Köster. 2024. Agent-based models of social behaviour and communication in evacuations: A systematic review. *Safety Science* 176 (2024), 106520.

[80] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi. 2021. Symbolicgpt: A generative transformer model for symbolic regression. *arXiv preprint arXiv:2106.14131* (2021).

[81] Martin Vastl, Jonáš Kulhánek, Jiří Kubalík, Erik Derner, and Robert Babuška. 2024. Symformer: End-to-end symbolic regression using transformer-based architecture. *IEEE Access* 12 (2024), 37840–37849.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[83] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro,

Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. doi:10.1038/s41592-019-0686-2

[84] Jinhuan Wang, Lei Zhang, Qiongyu Shi, Peng Yang, and Xiaoming Hu. 2015. Modeling and simulating for congestion pedestrian evacuation with panic. *Physica A: Statistical Mechanics and its Applications* 428 (2015), 396–409.

[85] Runxiang Wang, Boxiao Wang, Kai Li, Yifan Zhang, and Jian Cheng. 2025. DrSR: LLM based Scientific Equation Discovery with Dual Reasoning from Data and Experience. *arXiv preprint arXiv:2506.04282* (2025).

[86] Michael Winikoff, Lin Padgham, and James Harland. 2001. Simplifying the development of intelligent agents. In *Australian joint conference on artificial intelligence*. Springer, 557–568.

[87] Zengqing Wu, Run Peng, Takayuki Ito, and Chuan Xiao. 2025. LLM-Based Social Simulations Require a Boundary. *arXiv preprint arXiv:2506.19806* (2025).

[88] Zengqing Wu, Run Peng, Shuyuan Zheng, Qianying Liu, Xu Han, Brian I. Kwon, Makoto Onizuka, Shaojie Tang, and Chuan Xiao. 2024. Shall We Team Up: Exploring Spontaneous Cooperation of Competing LLM Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 5163–5186. doi:10.18653/v1/2024.findings-emnlp.297

[89] Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2025. Enhancing Mathematical Reasoning in LLMs by Stepwise Correction. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, Vienna, Austria, 21602–21623. doi:10.18653/v1/2025.acl-long.1048

[90] Jiajian Xiao, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. 2019. A Survey on Agent-based Simulation Using Hardware Accelerators. *ACM Computing Surveys (CSUR)* 51, 6 (2019). https://dl.acm.org/citation.cfm?id=3291048

[91] Zonglin Yang, Xinya Du, Junxian Li, Jie Zheng, Soujanya Poria, and Erik Cambria. 2024. Large Language Models for Automated Open-domain Scientific Hypotheses Discovery. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 13545–13565. doi:10.18653/v1/2024.findings-acl.804

[92] Yongchao Zeng, Calum Brown, and Mark Rounsevell. 2025. Too Human to Model: The Uncanny Valley of LLMs in Social Simulation–When Generative Language Agents Misalign with Modelling Principles. *arXiv preprint arXiv:2507.06310* (2025).

[93] Fa Zhang, Shihui Wu, and Zhihua Song. 2020. Crowd evacuation during slashing terrorist attack: a multi-agent simulation approach. In *2020 Winter Simulation Conference (WSC)*. IEEE, 206–217.

[94] Wenxin Zhang, Vincent Terrier, Xiaoshu Fei, Alex Markov, Scott Duncan, Michael Balchanos, Woong Je Sung, Dimitri N Mavris, Margaret L Loper, Elizabeth Whitaker, et al. 2018. Agent-based modeling of a stadium evacuation in a smart city. In *2018 winter simulation conference (WSC)*. IEEE, 2803–2814.

[95] Xinnong Zhang, Jiayu Lin, Xinyi Mou, Shiyue Yang, Xiawei Liu, Libo Sun, Hanjia Lyu, Yihang Yang, Weihong Qi, Yue Chen, et al. 2025. Socioverse: A world model for social simulation powered by llm agents and a pool of 10 million real-world users. *arXiv preprint arXiv:2504.10157* (2025).

[96] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19632–19642.

[97] Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. *Advances in neural information processing systems* 36 (2023), 31967–31987.

[98] Jinghui Zhong, Wentong Cai, and Linbo Luo. 2015. Crowd evacuation planning using cartesian genetic programming and agent-based crowd modeling. In *2015 Winter Simulation Conference (WSC)*. IEEE, 127–138.

[99] Jinghui Zhong, Dongrui Li, Zhixing Huang, Chengyu Lu, and Wentong Cai. 2022. Data-driven crowd modeling techniques: A survey. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 32, 1 (2022), 1–33.

[100] Jinghui Zhong, Linbo Luo, Wentong Cai, and Michael Lees. 2014. Automatic rule identification for agent-based crowd models through gene expression programming. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. 1125–1132.

[101] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Ed Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. 2024. Self-discover: Large language models self-compose reasoning structures. *Advances in Neural Information Processing Systems* 37 (2024), 126032–126058.

[102] Xuhui Zhou, Zhe Su, Tiwalayo Eisape, Hyunwoo Kim, and Maarten Sap. 2024. Is this the real life? is this just fantasy? the misleading success of simulating social interactions with llms. *arXiv preprint arXiv:2403.05020* (2024).

[103] Yu Zou, Vladimir A Fonoberov, Maria Fonoberova, Igor Mezic, and Ioannis G Kevrekidis. 2012. Model reduction for agent-based social simulation: coarse-graining a civil violence model. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 85, 6 (2012), 066106.

Prompt:

You are a helpful assistant tasked □ith disco□ering mathematical function for general problem described in natural language.

Here is the task description: Because of earthquake, □ou need to escape from the room □here □ou are as fast as possible. The room has a si□e of 33 * 33. There are 3 e□ts in the room. The e□ts are located at the left, bottom and right of the room.

Your task is to generate a general pattern that agents □ll follo□in the e□acuation process, please gi□e me 25 pieces of insights in bullet point:

Insights:

Here are 25 insights regarding a general e□acuation pattern for agents to follo□in a room of si□e 33□33 □ith three e□ts located on the left, bottom, and right:

1. **Identif□E□t Locations**: Clearl□mark the coordinates of the e□ts for all agents to see (e.g., left e□t at (0, □), bottom e□t at (□, 33), right e□t at (33, □)).

2. **Agent Positioning**: Place agents at different starting points throughout the room to minimi□e congestion at e□ts.

Fig. 11. Directly use LLM to generate insights.

A LLM Details

The ground truth generation was carried out using a locally deployed **Gemma3:27b** [78]. We set the parameters as follows: Temperature=0, top_k=64, top_p=1, num_predict=512, num_ctx=128 000. We used **GPT-4o-mini** and **GPT-4.1-mini** to generate high-quality general insights and code-based insights, respectively. To align with [74], **Llama-3.1-8B-Instruct** [21] was used for the function refinement.

B DFD Details and Hyperparameter Settings

B.1 Training and Evaluation

The shared evaluation function and seed function ϕ_0 for all islands at round 0 are shown in Figure 13. Figure 11 shows the prompt used for DFD-P, in which the LLM generates insights directly from the task description, without referring to agent trajectories. The resulting insights replace the general insights in LLM-SR-GI.

Figure 12, shows the the agents' previous exit choice supplied as part of the feature set in DFD-M and DFD-CM. The prompts used for decision function distillation are the same as those for DFD and DFD-C, the only difference being the seed function and the previous exit choices.

```python
def equation(x: np.ndarray, y: np.ndarray, capability: np.ndarray, mental: np.ndarray, crowdedness:
np.ndarray, dist0: np.ndarray, dist1: np.ndarray, dist2: np.ndarray, dist3: np.ndarray, dist4: np.ndarray,
dist5: np.ndarray, dist6: np.ndarray, left_exits_dens: np.ndarray, bottom_exits_dens: np.ndarray,
right_exits_dens: np.ndarray, decisions: np.ndarray, params: np.ndarray) -> np.ndarray:
    """ Mathematical function for exit score under multi-factor influences in emergency evacuation
scenarios.

    Args:
        x, y: Current agent coordinates (discrete 0-32).
        capability: 1 = strong, 0 = weak.
        mental: 1 = positive, 0 = negative.
        crowdedness: Current local crowdedness (discrete).
        dist0: A numpy array represents the distance to exit0.
        dist1: A numpy array represents the distance to exit1.
        dist2: A numpy array represents the distance to exit2.
        dist3: A numpy array represents the distance to exit3.
        dist4: A numpy array represents the distance to exit4.
        dist5: A numpy array represents the distance to exit5.
        dist6: A numpy array represents the distance to exit6.
        left_exits_dens: Number of people around left exits (0, 1, 2)
        bottom_exits_dens: Number of people around bottom exits (3, 4, 5)
        right_exits_dens: Number of people around right exits (6)
        decisions: Agent's memory of historical exit choice (1: bottom exits, 2: left exits, 3: right exits,
0: out of scope)
        params: Array of numeric constants or parameters to be optimized

    Noted that exit 0, 1, 2 belong to left exits, exit 3, 4, 5 belong to bottom exits, exit 6 belongs to
right exits.

    Return:
        A numpy array of score to different exits (7 exits in total) as the result of applying the
mathematical function to the inputs.
    """
    # leverage historical decision choices
    bottom_decision_sumup = np.count_nonzero(decisions==1)
    left_decision_sumup = np.count_nonzero(decisions==2)
    right_decision_sumup = np.count_nonzero(decisions==3)

    scores_to_exits =  x * params[0] + y * params[1] +  capability * params[2] + mental * params[3] +
crowdedness * params[4] + dist0 * params[5] + dist1 * params[6] + dist2 * params[7] + dist3*params[8] +
dist4*params[9] + dist5 * params[10] + dist6 * params[11] + left_exits_dens * params[12] + bottom_exits_dens
* params[13] + right_exits_dens * params[14] + bottom_decision_sumup * params[15] + left_decision_sumup *
params[16] + right_decision_sumup * params[17] + params[18]


    return scores_to_exits
```

Seed Function ϕ_0

Fig. 12. The seed function added with stateful information.

The function coefficients `params` are obtained via the implementation of Powell's gradient-free optimization method from the SciPy library [83]. During the decision function refinement process, the hyperparameters are set to $N_0 = 0.1$ and $N = 30000$.

Figure 14 lists the calibrated decision functions. A notable observation is that the number of parameters used, which we constrain to a maximum of 16, varies across the functions. We note that the code of DFD's final decision function is uncommented. Initially, DFD produced commented code, but gradually eliminated the comments over the course of the function refinement process.

The number of agents evacuated up to a given simulation time step on the training and generalization datasets is shown in Figures 15 and 16.

B.2 Prompt Templates

The prompt template used to generate general insights is shown in Figure 17. Figure 18 shows the prompt template used to generate the code-based insights from general insights.

Received 17 April 2026; revised 17 April 2026; accepted 22 May 2026

E□aluation Function

```
import numpy as np
from typing import Dict

#Initialize parameters
MAX_NPARAMS = 16
EXIT_NUM = 7
params =  np.ones(MAX_NPARAMS * EXIT_NUM)
def e□aluate(data: Dict) -> float:
    """ Evaluate the equation on data observations."""
    dict = □
            0: [15, 0],
            1: [16, 0],
            2: [17, 0],
            3: [32, 15],
            4: [32, 16],
            5: [32, 17],
            6: [17, 32]
        □
    # Load data observations
    inputs, outputs = data['inputs'], data['outputs']
    x, y, capability, mental, crowdedness, dist0, dist1, dist2, dist3, dist4, dist5,
dist6, left_exits_dens, bottom_exits_dens, right_exits_dens  = [inputs[:, i].reshape(-1, 1) for i in
range(inputs.shape[1])]

    # Optimize parameters based on data
    from scipy.optimize import minimize
    def loss(params):
        param_shaped = params.reshape((MAX_NPARAMS, 1, EXIT_NUM))
        y_pred = equation(x, y, capability, mental, crowdedness, dist0, dist1, dist2, dist3, dist4,
dist5, dist6, left_exits_dens, bottom_exits_dens, right_exits_dens, param_shaped)

        y_labels = np.argmax(y_pred, axis=1)
        y_pred = np.array([dict[int(value)] for value in y_labels])
        output = np.array([dict[value[0]]for value in outputs])
        return np.mean((y_pred - output) ** 2)

    loss_partial = lambda params: loss(params)
    result = minimize(loss_partial, np.ones(MAX_NPARAMS * EXIT_NUM), method='Powell')
    # Return evaluation score
    optimized_params = result.x

    loss = result.fun
    if np.isnan(loss) or np.isinf(loss):
        return None
    else:
        return -loss
```

Seed Function ϕ_0

```
def equation(x: np.ndarray, y: np.ndarray, capability: np.ndarray, mental: np.ndarray, crowdedness:
np.ndarray, dist0: np.ndarray, dist1: np.ndarray, dist2: np.ndarray, dist3: np.ndarray, dist4:
np.ndarray, dist5: np.ndarray, dist6: np.ndarray, left_exits_dens: np.ndarray, bottom_exits_dens:
np.ndarray, right_exits_dens: np.ndarray, params: np.ndarray) -> np.ndarray:
    """ Mathematical function for exit score under multi-factor influences in emergency evacuation
scenarios.

    Args:
        x, y: Current agent coordinates (discrete 0-32).
        capability: 1 = strong, 0 = weak.
        mental: 1 = positive, 0 = negative.
        crowdedness: Current local crowdedness (discrete).
        dist0: A numpy array represents the distance to exit0.
        dist1: A numpy array represents the distance to exit1.
        dist2: A numpy array represents the distance to exit2.
        dist3: A numpy array represents the distance to exit3.
        dist4: A numpy array represents the distance to exit4.
        dist5: A numpy array represents the distance to exit5.
        dist6: A numpy array represents the distance to exit6.
        left_exits_dens: Number of people around left exits (0, 1, 2)
        bottom_exits_dens: Number of people around bottom exits (3, 4, 5)
        right_exits_dens: Number of people around right exits (6)
        params: Array of numeric constants or parameters to be optimized

    Noted that exit 0, 1, 2 belong to left exits, exit 3, 4, 5 belong to bottom exits, exit 6
belongs to right exits.

    Return:
        A numpy array of score to different exits (7 exits in total) as the result of applying the
mathematical function to the inputs.
    """
    scores_to_exits =  x * params[0] + y * params[1] +  capability * params[2] + mental * params[3]
+ crowdedness * params[4] + dist0 * params[5] + dist1 * params[6] + dist2 * params[7] +
dist3*params[8] + dist4*params[9] + dist5 * params[10] + dist6 * params[11] + left_exits_dens *
params[12] + bottom_exits_dens * params[13] + right_exits_dens * params[14] + params[15]

    return scores_to_exits
```

Fig. 13. Evaluation function and seed function ϕ_0.

P□SR

```
score_exit_0 = (np.exp((x + distance0) / (0.016515918742839263 - distance3)) * distance6) * (0.005741653209937127 * people_around_bottom_exits)

score_exit_1 = (mental - (1 - distance5)) * (0.0020068668231833105 / 2.220198296024771)

score_exit_2 = np.exp(((1 - distance2) - distance3) / (1 - (1 - x))) / ((1 - (1 - crowdedness + distance2)) * 0.10278075995986115)

score_exit_3 = (4.289836091060936 / ((distance3 - 10.922678227509197) + y)) + -0.13286816955240077

score_exit_4 = 0.1714354087327947 / distance3

score_exit_5 = np.sqrt(y * (-0.049393654015761713 - (distance5 * -0.0019953099218021583)))

score_exit_6 = (((people_around_right_exits - people_around_left_exits) - distance3) * -0.0168302153187508 65) + ((distance6 * -0.014594868142424671) - (distance0 * -0.019156952334561))
```

LLM-SR

```
def equation(x: np.ndarray, y: np.ndarray, capability: np.ndarray, mental: np.ndarray, crowdedness: np.ndarray, distance0: np.ndarray, distance1: np.ndarray, distance2: np.ndarray, distance3: np.ndarray, distance4: np.ndarray, distance5: np.ndarray, distance6: np.ndarray, people_around_left_exits: np.ndarray, people_around_bottom_exits: np.ndarray, people_around_right_exits: np.ndarray, params: np.ndarray) -> np.ndarray:
    ''' comments '''
    # Calculate the exit scores considering the input factors and the desired weights
    # Introduce a term to penalize exits with high crowdedness
    crowdedness_factor = 1 - (crowdedness / 10)  # Normalized crowdedness
    # Use a polynomial function to model the relationship between capability and mental factors
    capability_mental_factor = (capability + 1) ** 2 * (mental + 1)
    # Use a weighted average to combine the factors affecting each exit
    scores_to_exits = (capability_mental_factor * (x / 32) * (params[0] + params[1] * (y / 32)) +
                       (distance0 + distance1 + distance2) * crowdedness_factor * 0.5 * (params[2] + params[3] * (y / 32)) +
                       (1 - (distance3 + distance4) / 32) * (params[4] + params[5] * (y / 32)) * people_around_left_exits +
                       (1 - (distance5 + distance6) / 32) * (params[6] + params[7] * (x / 32)) * people_around_bottom_exits +
                       (1 - (distance0 + distance1) / 32) * (params[8] + params[9] * (y / 32)) * people_around_right_exits +
                       params[10] ** params[11] * crowdedness_factor  # Introduce a higher-order term for crowdedness
                       )
    return scores_to_exits
```

LLM-SR-GI

```
def equation(x: np.ndarray, y: np.ndarray, capability: np.ndarray, mental: np.ndarray, crowdedness: np.ndarray, distance0: np.ndarray, distance1: np.ndarray, distance3: np.ndarray, distance4: np.ndarray, distance6: np.ndarray, people_around_left_exits: np.ndarray, people_around_bottom_exits: np.ndarray, people_around_right_exits: np.ndarray, params: np.ndarray) -> np.ndarray:
    ''' comments '''
    # Reciprocal of distances to exits with weighted average
    reciprocal_distance = ((1 / np.minimum(distance0, distance1)) * params[0] +
                           (1 / np.minimum(distance3, distance4)) * params[1] +
                           (1 / distance6) * params[2])
    # Weighted average of capability and mental state
    human_factor = ((capability * params[3] + mental * params[4]) +
                    (capability * params[5] * people_around_left_exits + mental * params[6] * people_around_bottom_exits))
    # Weighted sum of reciprocal of crowdedness and surrounding individuals
    surround_factor = ((1 / (people_around_left_exits + people_around_bottom_exits + people_around_right_exits)) +
                       (people_around_right_exits * params[7]) +
                       (1 / (crowdedness + 1)))
    # Balancing term
    balance_term = ((1 / (x + y + capability + mental + crowdedness)) +
                    (params[8] * (1 / (np.minimum(distance0, distance1) + 1 + people_around_left_exits))))
    # Weighted sum of factors
    score = ((reciprocal_distance * 0.3) + (human_factor * 0.2) +
             (surround_factor * 0.15) + (balance_term * 0.25))
    return score
```

DFD

```
def equation(y: np.ndarray, capability: np.ndarray, mental: np.ndarray, crowdedness: np.ndarray, distance0: np.ndarray, distance1: np.ndarray, distance2: np.ndarray, distance3: np.ndarray, distance4: np.ndarray, distance5: np.ndarray, distance6: np.ndarray, people_around_left_exits: np.ndarray, people_around_bottom_exits: np.ndarray, people_around_right_exits: np.ndarray, params: np.ndarray) -> np.ndarray:
    ''' comments '''
    exit_proximity = np.ones((x.shape[0], params.shape[-1]))
    crowd_size = np.ones((x.shape[0], params.shape[-1]))
    exit_proximity = np.concatenate((distance0, distance1, distance2, distance3, distance4, distance5, distance6), axis=1)
    crowd_size = np.concatenate((people_around_left_exits, people_around_left_exits, people_around_left_exits, people_around_bottom_exits, people_around_bottom_exits, people_around_bottom_exits, people_around_right_exits), axis=1)
    scores_to_exits = 1 / (exit_proximity + crowd_size)
    scores_to_exits = scores_to_exits * 1 / (params[15] + params[0] * (capability - 0.5) ** 2 + params[1] * (mental - 0.5) ** 2 + params[2] * crowdedness ** 2 + params[3] * people_around_left_exits ** 2 + params[4] * people_around_bottom_exits ** 2 + params[5] * people_around_right_exits ** 2 + params[6] * exit_proximity + params[7] * crowd_size)
    return scores_to_exits
```

DFD-C

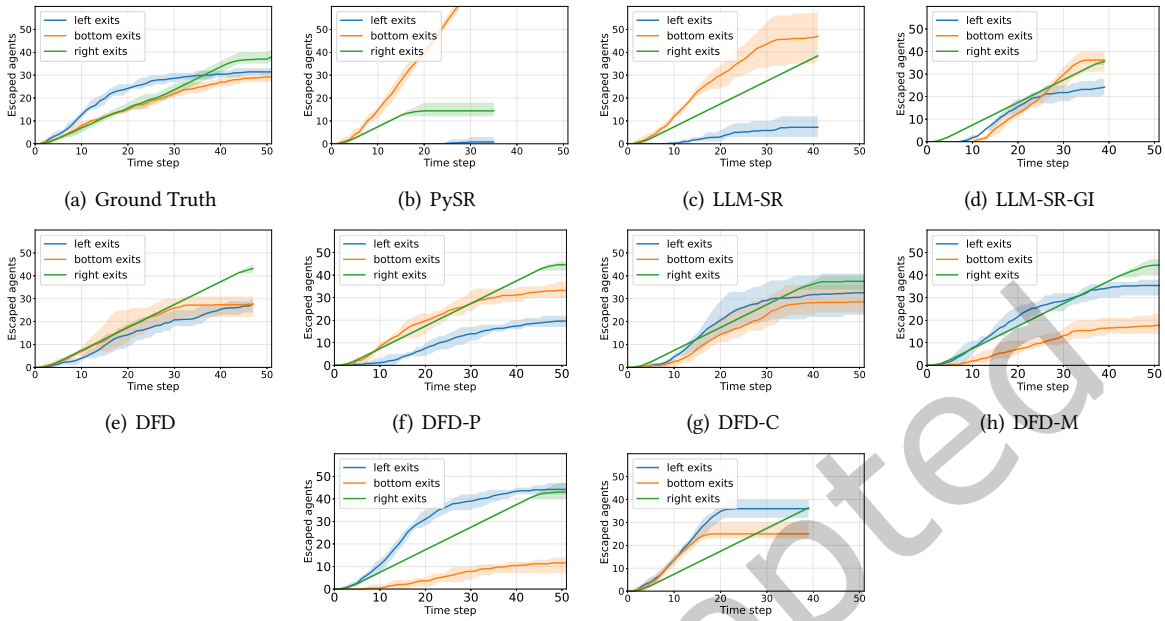
```
def equation(capability: np.ndarray, mental: np.ndarray, crowdedness: np.ndarray, distance0: np.ndarray, distance1: np.ndarray, distance2: np.ndarray, distance3: np.ndarray, distance4: np.ndarray, distance5: np.ndarray, distance6: np.ndarray, people_around_left_exits: np.ndarray, people_around_bottom_exits: np.ndarray, people_around_right_exits: np.ndarray, params: np.ndarray) -> np.ndarray:
    ''' comments '''
    # Initialize weights for distance, crowd, readiness, mental state and crowdedness
    weight_distance = params[0]
    weight_crowd = params[1]
    weight_readiness = params[2]
    weight_mental = params[3]
    weight_crowdedness = params[4]
    # Calculate distance
    distances = np.concatenate((distance0, distance1, distance2, distance3, distance4, distance5, distance6), axis=1)
    # Calculate crowd
    crowd = np.concatenate((people_around_left_exits, people_around_left_exits, people_around_left_exits, people_around_bottom_exits, people_around_bottom_exits, people_around_bottom_exits, people_around_right_exits), axis=1)
    # Calculate readiness based on capability
    readiness = weight_readiness * capability * (1 - crowdedness / 10)  # Adjust readiness based on crowdedness
    # Calculate mental state factor based on mental state
    mental_state_factor = weight_mental * mental * (1 - crowdedness / 10)  # Adjust mental state factor based on crowdedness

    # Calculate crowdedness factor
    crowdedness_factor = weight_crowdedness * crowdedness

    # Combine weighted factors to create composite score
    scores = (readiness * weight_readiness + mental_state_factor * weight_mental + crowdedness_factor) * 1 / (distances + crowd + 1e-6) + (1 - readiness - mental_state_factor - crowdedness_factor) * 1 / (distances + people_around_left_exits + 1e-6)

    return scores
```


Fig. 14. Optimized decision function.

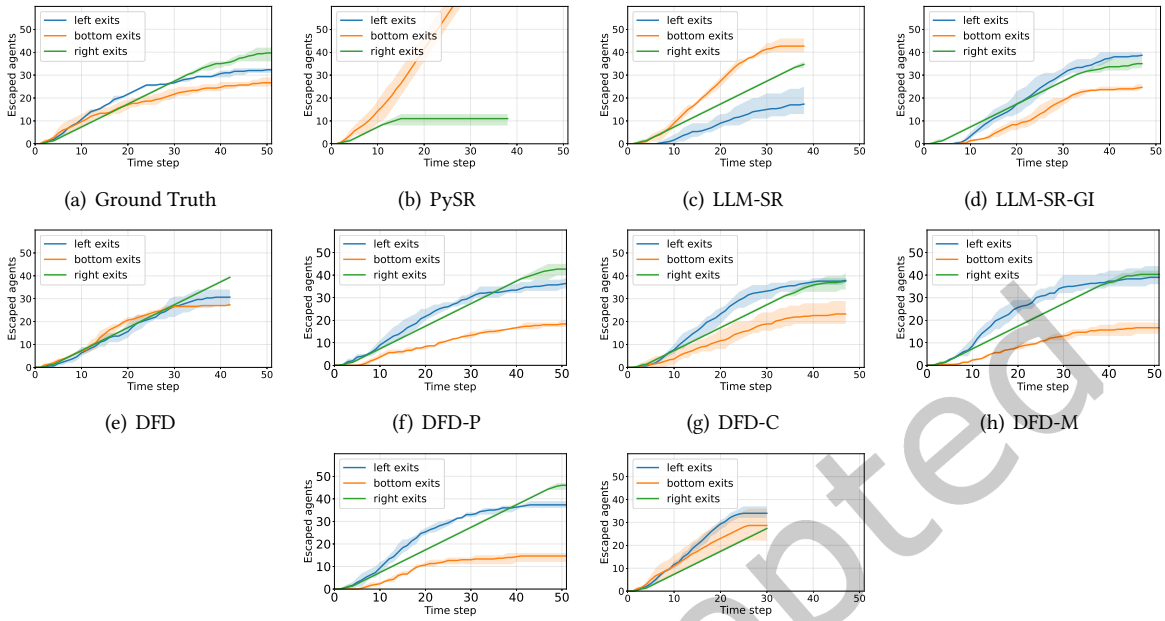


(a) Ground Truth (b) PySR (c) LLM-SR (d) LLM-SR-GI

(e) DFD (f) DFD-P (g) DFD-C (h) DFD-M

(i) DFD-CM (j) Wang et al.

Fig. 15. Escaped agents over time across the methods for the training dataset.



(a) Ground Truth (b) PySR (c) LLM-SR (d) LLM-SR-GI

(e) DFD (f) DFD-P (g) DFD-C (h) DFD-M

(i) DFD-CM (j) Wang et al.

Fig. 16. Escaped agents over time across the methods for the generalization testing dataset.

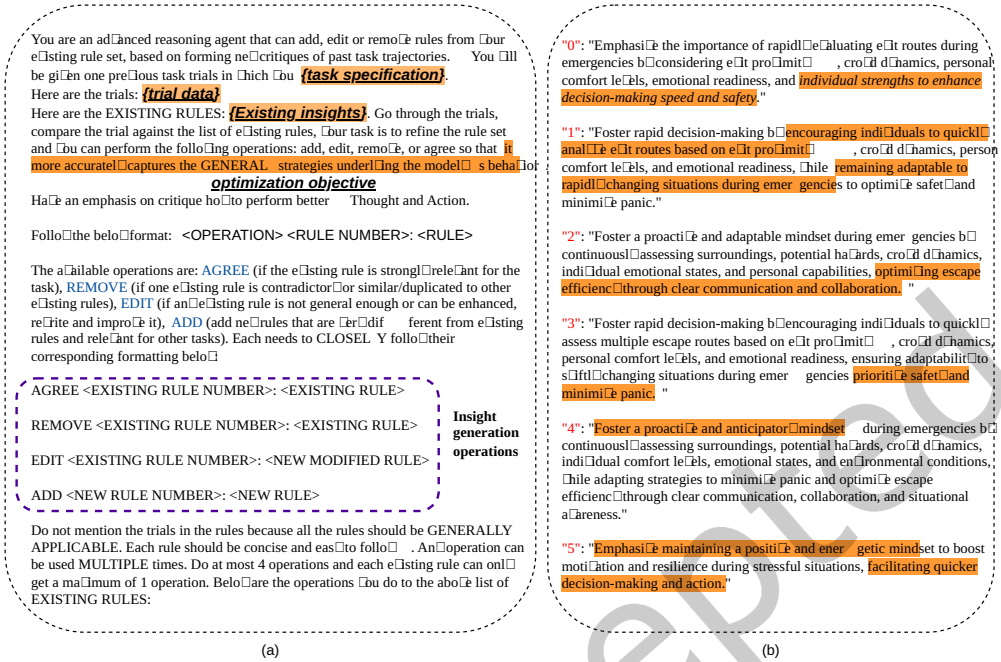


Fig. 17. (*a*) shows the **general insight generation template**, which includes the task specification, trajectories, existing insights, and the possible operations that can be applied to update the insights pool. Insights are sorted by their scores, which reflect the level of agreement. (*b*) lists the **generated insights** in a descending order. The highlighted parts indicate that these high-level instructions do not represent unambiguous and directly applicable rules.

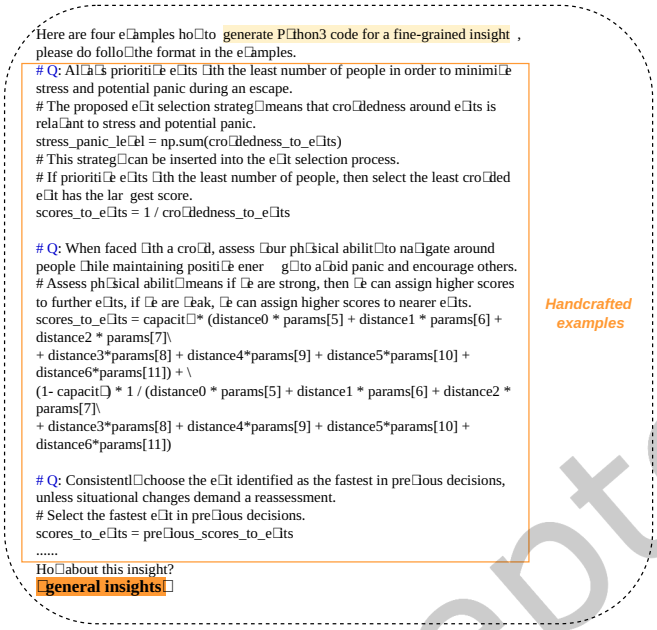


Fig. 18. Prompt template for code-based insights generation. Existing general insights are translated into code-based insights.