Proceedings of the Ninth Goal Reasoning Workshop Submitted 11/2021; published 11/2021

Plan-design Goal Recognition with Autoencoding Transformer for the CERBREC Modeling Platform

David R. Winer DRW@CERBREC.COM
Garrett Z. Wang GARRETT@CERBREC.COM
415 Monroe Street, Unit 302, Hoboken, NJ 07030 USA

Abstract

We describe a goal recognition problem for tasks where agents design plans or plan-like programs in an interactive environment. For these problems, a goal is an entire design rather than the state that results from a plan's execution. We introduce a specific application as a case study, CERBREC, which is a platform for designing machine learning models as plan-like artifacts. It is used in this paper as both a case study for our approach as well as the tool used to implement our solution. Our solution is an autoencoding transformer architecture adapted from language modeling that supplies a heuristic function for a planner to generate suggested user actions that bring the user's current model in CERBREC to a goal state. Though the problem is widely applicable, we have not seen any attempts to formalize the sub-type of problem or learn from the trajectory of plans created by the user during the design process.

1. Introduction

Planning agents create plans to achieve a goal state, and the typical goal recognition problem is to infer the goal state of an agent from observed or partially observed plan steps (Kautz et al., 1986; Ramírez & Geffner, 2009; Shvo & McIlraith, 2020). Of practical significance is a sub-type of problem that we identify in this work, where planning agents take actions to *design* a plan, and the agent's goal is the final plan design. In other words, we model goal recognition for a plan-space planning agent where the steps the agent takes are actions like add steps and bindings and remove steps and bindings on a path towards creating the final plan. The corresponding sub-type of goal recognition problem is to determine the plan that the agent intends to design. Though similar to the plan recognition problem because the form of a solution is a plan, it is distinct because the trajectory of observations are plans, not observed plan steps that are part of a single plan. For non-rational agents, these problems are not equivalent.

Rather than acting directly in an environment, imagine an agent is designing a plan on a white-board, incrementally adding steps and bindings, accidentally creating plans that are not executable, and sometimes undoing their previous actions (Figure 1). An observer that wishes to be helpful will predict the intended plan design in order to provide the most helpful recommendation. For example, if the planning agent makes an error i.e. some plan constraint is violated, then perhaps the observing agent can correctly infer the desired plan and have its suggestion accepted by the planning agent. This has relevance to the class of intelligent user interface recommender systems (Bohnenberger &

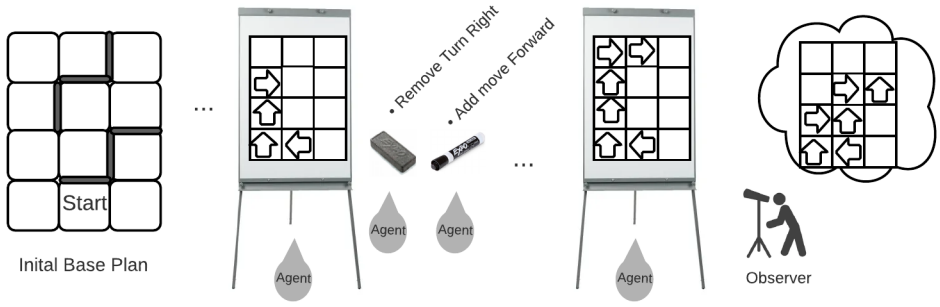

Figure 1. A depiction of the plan-design goal recognition problem. The agent designs a plan, depicted with a whiteboard to illustrate that each action is an edit to a plan rather than an action in the base environment. During the design process, the agent may create invalid plans. The observer uses the sequence of design steps (edits) and infers a valid goal plan that best explains the design trajectory.

Jameson, 2001; Du et al., 2019) where specifically, users are designing a plan-like artifact such as for data flow visual programming tools (Hils, 1992; Dinmore & Boylls, 2010), intelligent coding tutors (Price et al., 2016, 2019), robot behavior design tools (Datta et al., 2012), narrative authoring assistants (Thomas & Young, 2006; Stefnisson & Thue, 2018), game design assistants (Nelson & Mateas, 2008; Smith et al., 2010), experiment design assistants (Percie du Sert et al., 2017), dialogue workflow design assistants (Yi & Jung, 2017; Allen et al., 2018), and deep learning model building assistants (Adhikari, 2021).

CERBREC is a visual modeling platform that lets users create a plan-like model to operate over input data such as through text and numerical operations. The resulting CERBREC model is reusable for new data and so it is the entire plan, not just the result, that is the goal artifact. Though the plan-design environment is fully observable and deterministic, users may violate constraints that allow data to be operated on, e.g., they forgot to transpose an array for a matrix multiplication operation, or they did not tile a scalar value to the correct array size. Our objective is to maximize the productivity of CERBREC users by providing them with suggestions. Figure 2 shows a screenshot of the CERBREC platform.

In this paper, we provide an initial formulation of plan-design goal recognition, ground the formalization for the CERBREC platform, outline a learning approach (which is implemented with CERBREC), and walk through an example prediction.

2. Related Work

The objective of goal recognition is to infer a goal that best accounts for the observed behavior of an agent (Shvo & McIlraith, 2020) and is a sub-type of the plan recognition problem where the goal state and some set of intermediate plan states are also inferred (Kautz et al., 1986; Carberry, 2001; Pattison & Long, 2011). Sometimes plan and goal recognition problems are lumped together as plan and goal recognition (PGR) (Höller et al., 2018). Agents are frequently modeled as planners, sometimes called "plan recognition as planning" (PRP) (Ramírez & Geffner, 2009; Pereira et al.,

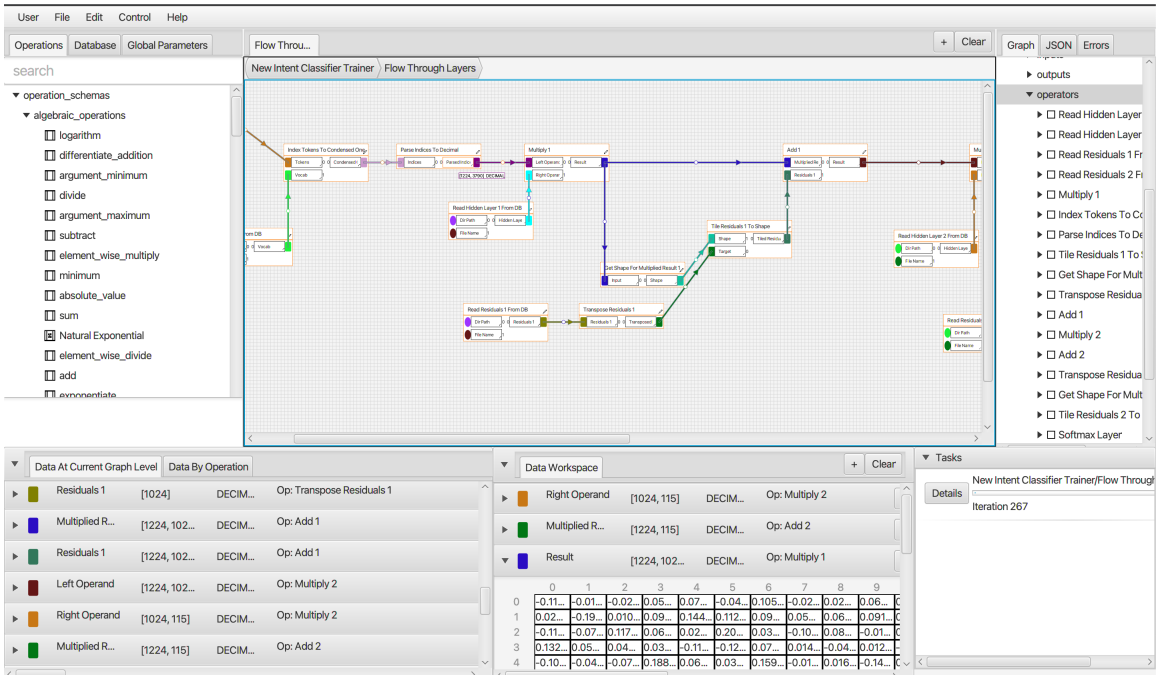

Figure 2. A screenshot of the CERBREC platform gives users a no-code interface for developing models such as for text cleaning and machine learning classifiers. In the central grid area, the boxes are operations that can be dropped in from the left panel. Each operation's inputs are on its left side, and its outputs are on its right side. The inputs are supplied either by another operation in a binding link, or supplied through a bootstrapped datum (oval). At the bottom, the ground data components are displayed along with their data shape, type, and operation that produced them. CERBREC is one application of the PDGR: users create a plan-like base model via plan-design interaction like add and remove, and the platform observes the user and predicts the goal base plan.

2017) either with classical planners or probabilistic planners (Ramírez & Geffner, 2010; Pattison & Long, 2011) with different levels observability and noise in the environment for the observer (Geib & Goldman, 2005; Ramirez & Geffner, 2011; Sohrabi et al., 2016; Keren et al., 2020). One typically distinguishes probabilistic PRP from statistical models like simple n-grams (Blaylock & Allen, 2003; Mott et al., 2006), variable-order Markov models (Armentano & Amandi, 2009), Markov logic networks (Ha et al., 2011; Min et al., 2016), and more recent deep learning approaches like CNN and LSTM networks (Maynard et al., 2019; Borrajo et al., 2020) via the degree a planning domain theory is used (Borrajo et al., 2020).

Goal recognition also can be distinguished by whether the task is performed actively or off-line. The merit of active goal recognition is that an observing agent can intervene and elicit or expedite learning the goals of another agent (Shvo & McIlraith, 2020; Amos-Binks & Cardona-Rivera, 2020). If successful, then agents could take anticipatory steps that leverage considering the goals of other agents (Amos-Binks et al., 2020) such as to help assist the user (Armentano & Amandi, 2012) or select dramatically appropriate actions in an interactive narrative (Cardona-Rivera & Young, 2015).

Off-line goal recognition can be appropriate for designing environments (Keren et al., 2014) by shaping the environment such that observations are goal-revealing.

Goal recognition in path planning domains may be relevant to our learning approach which encodes plans as continuous vectors. Vered & Kaminka (2017) utilizes active goal recognition for motion planning, where the task begins with an initial pose of an agent, and each observation is a specific point or trajectory in a continuous space. A planner generates optimal plans for each goal that includes all observations. To improve efficiency, they use a heuristic to decide if the planner should recompute (reducing the number of calls necessary) and they prune goals that are very unlikely from subsequent calls to the planner. Masters & Sardina (2018) address goal recognition for path planning by creating a cost difference heuristic that addresses the fact that agents may be unlikely to both be on an optimal path and to have passed through the observed locations.

Our task involves goal recognition with two planning domains, the underlying base planning domain and the domain modeling how the user interacts with the base plan. In bipartite partial-order causal-link planning (BiPOCL) (Winer & Young, 2016b,a, 2017) one planning domain contains schemata whose effects create constraints for another planning domain problem. The purpose in that work is to provide a way for a storytelling agent to design stories that fit some intended discourse-level narrative goals such as what kind of story to tell and what kinds of elements are needed to tell it. Some elements of our formalism are similar in some ways because we distinguish the base planning domain from a higher-level one containing predicates that characterize base plan using a specialized plan description language.

This work introduces the CERBREC platform for interactive model development which addresses many issues that model developers face with existing modeling frameworks such as being error-prone for data shape and type inconsistency (Islam et al., 2019; Zhang et al., 2019, 2020), having a lack of profiling and debugging tools (Zhang et al., 2019), running out of compute resources (Zhang et al., 2020; Gao et al., 2020), serving models in different use cases (Chen et al., 2020), and overall confusing model building API methods (Humbatova et al., 2020; Zhang et al., 2020; Chen et al., 2020). To further help users, the motivation for our work is to make CERBREC an adaptive interface that learns from interaction traces to make better predictions about users (Langley, 1999) and improve the experience for model development. There are other systems with similar visual programming functionality for model development such as DARVIZ (Sankaran et al., 2017) and DL-IDE (Tamilselvam et al., 2019) (also see their review of other systems), but many of these systems are non-hierarchical, require writing code (they don't decompose to primitive level operations like add and multiply), still require setting up servers and libraries, or don't let you see a sample of your data flow through the model. Ultimately, they don't follow a principled means-ends design that enables them to be leveraged by a planner to perform goal recognition as planning. At the time of writing, CERBREC is in the beta testing phase[1] and will be available by the end of the year as a no-subscription platform.

1. www.cerbrec.com

3. Problem Formulation

In goal-recognition, the most likely goal of an agent is identified or inferred from a sequence of observations (Kautz et al., 1986; Pattison & Long, 2011). In **plan-design goal recognition** (PDGR), agents are observed taking steps in a **plan-design domain** where each step is an edit to a **base plan** in a **base planning domain**, and the task is to infer the most likely valid base plan that the agent intends to design.

More formally, the **Plan-design Goal Recognition Problem** is defined as:

$$PDGR = \langle B, D, I_d, Obs \rangle$$

where

- B represents a **base** planning domain with operators and constraint predicates over typed variables and a plan is valid just if no constraint is violated,
- D is the **plan-design** domain whose states are plans in B
- I_d is the **initial design state**: a (partial) base plan with an initial state containing ground objects, a set of steps and bindings, no (known) goal state, and
- Obs is an alternating sequence of base plans and plan-design actions

$$Obs = I_d, d_0, b_0, ..., d_f, b_f$$

The solution to the problem is the (valid) base plan that best explains the plan design steps.

$$g_\pi = \text{argmax}_{b \in B} Pr(b|Obs)$$

Next, we introduce the CERBREC base domain (i.e., a domain for operating over data) and the plan-design domain (a domain for modifying base plans).

4. CERBREC

The CERBREC platform is a visual programming tool for users to create data-flow models on a remote server without coding. The user draws on a graph representing a plan-like network of data where nodes in the plan are operations. The user imports or bootstraps data that is then sourced into the model and then operated upon by the operations in the model's plan. The purpose of CERBREC is to provide users with a no-code tool to develop data-based models such as for learning parameters and serving model results. Each action by the user is an editing of the plan, and the CERBREC system executes as much as the plan as is valid and can execute from the initial state.

Figure 3 shows a typical natural language processing workflow for CERBREC where the user starts from data, splits each sentence into tokens, converts each token to a vocab index, and then runs through multiplication layers of a classifier. Each of these components is composite (illustrated through the expand icon) and can be selected to view its sub-plan which itself may contain composite operations.

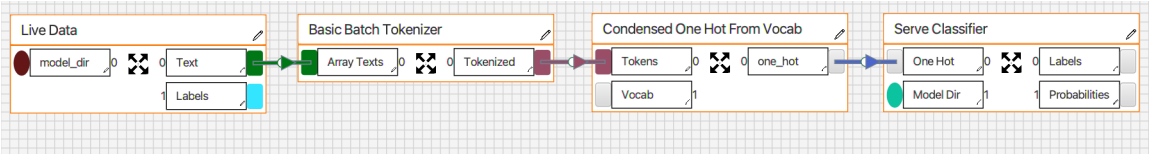

Figure 3. Typical natural language processing workflow represented as a CERBREC plan with composite actions. The plan is not complete because the Vocab input is not supplied with data.

4.1 Base Domain

The base domain characterizes the kinds of models that users can create in CERBREC from initial data. Each initial datum is typed

$$types = \{integer, decimal, boolean, text, datetime, enum\}$$

and has a shape $\mathcal{Z}_+^n$ (a list of positive integers) where if $n = 0$ then the datum is scalar, if $n = 1$ the shape is a 1D array and its first (0-th) dimension's value is the length of that array, $n = 2$ is 2D, e.g., shape $[2, 2]$ is a 2x2 array, and so on.

Each variable represents a datum and is either **ground** or **non-ground** (but may have constraints). The domain characterizes these constraints which provide partial criteria for a datum. For example, $\{d\}\_is\_integer\_or\_decimal$ indicates that datum substituting variable d is an integer or decimal, $\{d\}\_has\_min\_size\_\{2\}\_at\_dim\_\{0\}$ indicates that datum d's first dimension (0) has at least size 2, etc. More examples are shown in Table 1.

The CERBREC domain is the base domain, B, in the CERBREC instance of the $PDGR$. The CERBREC operation schemata have a similar purpose to classic STRIPS operator schemata (Fikes & Nilsson, 1971) and are defined slightly differently because of the nature of the task. A schema is a template for an operation step in a plan.

Definition 1 (CERBREC Primitive Operation Schema) *Represented as $\langle \alpha, I, O, C \rangle$ where α is a name, I is a set of input variables, O is a set of output variables, $I \cap O = \emptyset$, and C is a set of n-ary constraints over $I \cup O$. **Notation**: if s is an operation instance of a schema, then s_I and s_O refer to the inputs and outputs of s, respectively. s is **executable** if every input in I is supplied by ground data, and if s is executable then each datum in O is ground.*

Predicate	Meaning
{}_data_type_is_decimal	0th term is decimal
{}_is_less_than_{}	0th term scalar number is less than 1st term is scalar number
{}_num_dims_is_same_as_{}	0th and 1st terms have same number of dimensions in shape
{}_is_less_than_size_of_array_{}_at_dim_index_{}	1st term's shape at index of 2nd term is greater than scalar number provided in 0th term
{}_{}_dimension_is_equal_to_{}_{}	0th and 2nd terms are non-scalar arrays, and their data shapes at index provided by 1st and 3rd term, respectively, are equal

Table 1. Example constraint predicates for operations in the base CERBREC planning domain.

Each schema α represents a template method which accepts input variables I and generates outputs O if constraints C hold. For example, schema ADD has the following structure:

```
{   "name" : "add",
    "inputs" : [
        {
            "name": "left_operand",
            "data": null
        },
        {
            "name": "right_operand",
            "data": null
        }
    ],
    "outputs" : [
        {
            "name": "added_result",
            "data": null
        }
    ],
    "constraints" : [
        "{left_operand}_data_type_is_integer_or_decimal",
        "{left_operand}_shape_is_the_same_as_{right_operand}",
        "{right_operand}_data_type_is_the_same_as_{left_operand}",
        "{added_result}_data_type_is_the_same_as_{right_operand}",
        "{added_result}_shape_is_the_same_as_{right_operand}"
    ]}
```

A schema is the template from which instances are constructed, and we refer to a schema instance as an **operation**. The variables in operations can be supplied with data and linked together such that they have equal value.

Definition 2 (CERBREC **Binding Link)** *Represented as* $s \xrightarrow{s_o,t_i} t$ *where* s, t *are operations and* $s_o \in s_O, t_i \in t_I$ *are variables, indicating that for any grounding of variables,* $s_o = t_i$*. Terminology:* t *consumes* s_o *via* t_i*, and* t_i *is supplied by* s *via* s_o*.* s *is a causal ancestor of* t *as is any ancestor of* s*, transitively, and* t *is a causal descendant of* s *as is any descendant of* t*, transitively.*

Note that every operation creates new objects (i.e. new data), and the binding link refers to a binding between variables, not a binding between preconditions and effects. Thus, unlike classic planning, ground objects do not persist beyond the operations that consume them, *though they are still relevant because they affect the shape and type of data in the output in their causal descendants*. The constraints in each operation prescribe how the shape and type of the inputs affects the outputs, and vice versa, which combined with binding links form a constraint graph. Looking at an example in Figure 4, even though the transpose operation is not supplied with ground data, the binding link to left_operand of the add operation (with shape $[3, 2], decimal$) constrains what is a valid shape and type for the input of transpose ($[2, 3], decimal$).

Definition 3 (CERBREC **Plan)** *Represented as* $\langle D_{in}, S, L, D_{out} \rangle$ *where* D_{in} *is an input processing dummy operation whose outputs are the initial data,* S *is a set of* CERBREC *operations, and* L *is*

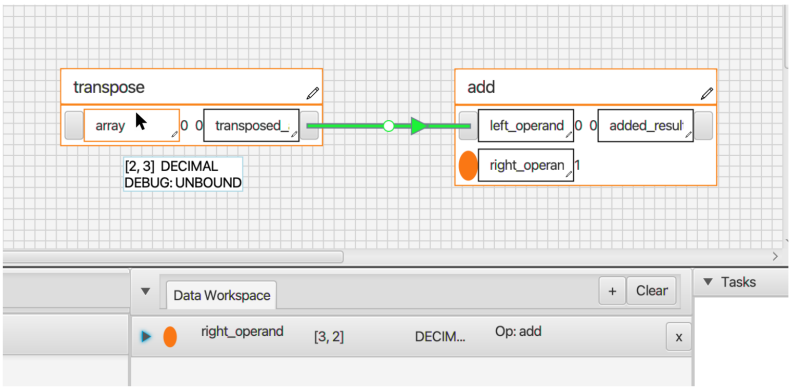

Figure 4. An example of constraints propagating through the plan. Even though the transpose operation is not supplied with ground data, the binding link to left_operand of the add operation (e.g., with shape $[3, 2], decimal$) constrains what is a valid shape and type for the input of transpose (e.g., to $[2, 3], decimal$).

a set of CERBREC *binding links, and* D_{out} *is an output processing dummy operation whose inputs are the final data. The plan is* ***valid*** *just if for every* $s \xrightarrow{s_o,t_i} t \in L$, s_o *and* t_i *are consistent given the constraints in* S*, every input of every step is supplied by exactly one link, and the graph formed by binding links and operations is acyclic. The plan is* ***complete*** *just if every input variable of every step in* S *is supplied by ground data and therefore every output variable of every step is ground.*

The CERBREC plan is reusable for different data input, so long as the data provided is valid given the constraints. Plans can be constructed hierarchically via composite operations which define a sub-plan for how its inputs are transformed into its outputs. A schema is considered composite if it is associated with a sub-plan.

Definition 4 (CERBREC Composite Schema) *Represented as* $\langle \alpha, I, O, \pi_{sub} \rangle$ *where* α, I, O *are defined as with primitive schemata, and* π_{sub} *is a* CERBREC *plan whose dummy input operation's output is* I *whose dummy output operation's input is* O*. A composite operation is valid just if its sub-plan is valid and complete just if its sub-plan is complete.*

Users can drop in pre-built composite operations or create their own and can then click into them to view their sub-plan. Composites help users organize the level of abstraction they are working on, and every composite is accessible by an API making it worthwhile to organize a project around the inputs and outputs of a composite.

4.2 CERBREC Plan-Design Domain

Users interact with a CERBREC plan in the CERBREC platform's environment. The interaction is modeled as a planning domain with STRIPS operator schemata (Fikes & Nilsson, 1971) where each schema has preconditions indicating whether the operation can execute on the base plan and effects that indicate add or remove conditions on the base plan. It is similar to previous work on bipartite

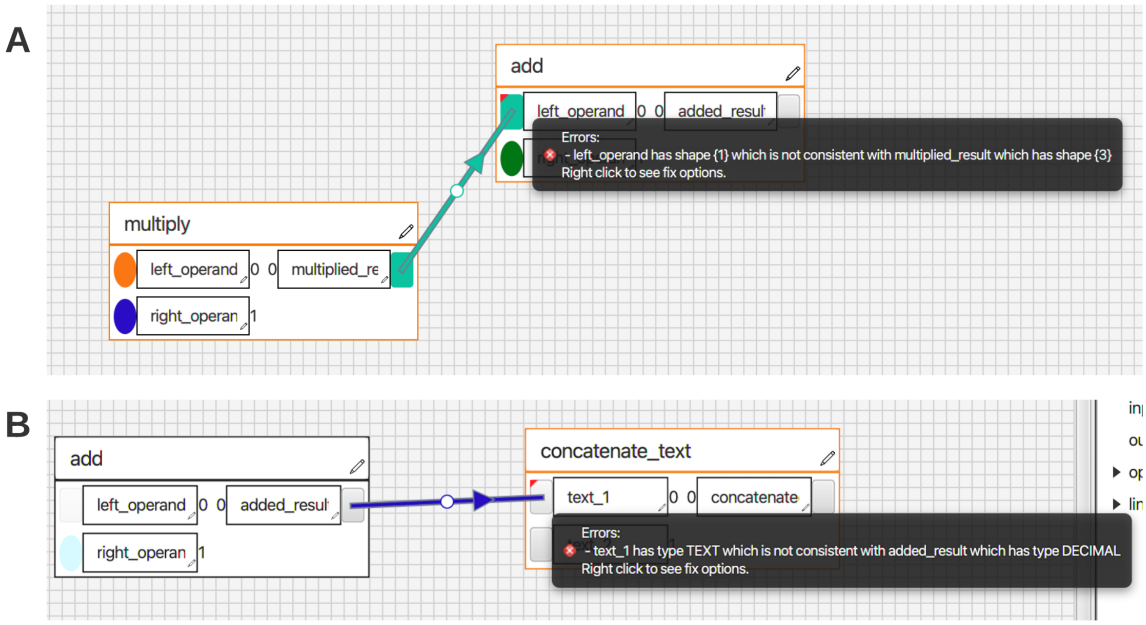

Figure 5. Examples of constraint violations in CERBREC. In (A), data from the multiplied result has a shape that is inconsistent with the expected shape of the left_operand, therefore the added_result is not calculated (no color means non-ground). (B) shows that the constraints can be violated even when data is non-ground: added_result is decimal type but concatenate_text expects text inputs.

planning for generating narrative discourse where the higher-level discourse planning operations use predicates that are descriptors on the base story plan (Winer & Young, 2016a, 2017). The interactive planning domain is called the **Plan-Design Domain**

During plan-designing (i.e., the act of authoring a plan), users can create invalid base plans which could not execute in the base environment. Except for cycles (not allowed by the system), users are *not* prevented from taking missteps; instead, they are informed through a warning message as shown in Figure 5. When a user does some plan-design action that creates an invalid base plan, it is as though the planning agent proposes to walk through an obstacle in Figure 1.

The set of actions that are available to the user which are relevant here include

- *Add {operation} from {schema} to {sub-plan}*,
- *Remove {operation} from {sub-plan}*,
- *Add {binding} from {variable} to {variable}*, and
- *Remove {binding} from {variable} to {variable}*
- *Bootstrap {datum} on {variable}*
- *Remove {datum} from {variable}*

where elements in {} are variables representing CERBREC domain and plan elements. Users can bootstrap datum to a variable, which is where the user can supply small data manually through a GUI such as a name, a scalar, or a short array, instead of reading data from a database or a streaming source (see CERBREC plans can execute with arbitrary data input, but bootstrapped data is hard-coded into the plan. Bootstrapping data is equivalent to creating a dummy operation with no inputs and whose only output is the bootstrapped data that supplies the original input through a binding link. Other actions that are less relevant to this work include changing position of operations, adding control points to binding links, changing the viewing order of variables, importing and exporting data, creating data from file, loading and saving plans from file, etc.

The CERBREC *design planner* is a planner that takes a base plan and performs a sequence of plan-design steps. The planner iteratively selects a flaw in the plan, and then selects a repair method, until there are no more flaws, similar to a classic plan-space POCL algorithm Weld (1994); Younes & Simmons (2003). There are 2 kinds of flaws:

1. **Violated Constraint Flaw** represented as a binding link and set of constraints that are violated. The flaw is repaired by removing some binding link in the causal ancestry to the binding link, or removing the binding link itself.

2. **Un-supplied Input Flaw** represented as an input of an operation which is not supplied by a binding link or by bootstrapped data. The flaw is repaired by adding some binding link with the output of another operation, either one that is newly added to the plan or is reused, and whose output is consistent with the un-supplied input via the constraints of the base plan.

The design planner is used to generate a suggested sequence of plan-design steps to make a plan valid and complete. A solution to the PDGR will provide a heuristic for guiding the design planner. The planner should select the action that approaches the predicted goal state i.e. the user's goal plan. A heuristic ranks neighboring nodes by estimating which plan is closer to the goal.

We have introduced the CERBREC platform and described what is considered the base planning domain and the plan-design domain for the PDGR problem. Next, we describe how we learn from observations in service of solving the PDGR problem.

5. Learning from Observations

In the PDGR problem, the observations are an alternating sequence of base plans and plan-design actions. In CERBREC, a complete sequence (i.e., fully observable) of plan-design actions (without the base plans) starting from Figure 3 might be

1. Add Operation READ-FROM-DATABASE

2. Bootstrap Input VOCAB-DIR on READ-FROM-DATABASE

3. Add Binding READ-FROM-DATABASE $\xrightarrow{output,Vocab}$ CONDENSED ONE HOT FROM VOCAB

In other words, the user has the vocab data already prepared in their directory and just needs to read it into the plan using READ-FROM-DATABASE operation, and link it to the $Vocab$ variable.

We collect observations in two different ways to train a model to perform PDGR: (1) artificially perturbing valid and complete plans, and (2) from session data. After our preliminary studies are finished, we could then also validate PDGR with live users receiving plan-design suggestions. At first, we plan to only offer those suggestions when users enter invalid base plan states (violate constraints).

We convert any CERBREC base plan into a vectorized form by establishing a vocabulary of bindings (i.e., the **binding vocabulary** (BV)). Given the full set of schemata, the BV is the complete set of output, input pairs between every schemata pair including itself, even if they are necessarily inconsistent via their constraints. Additionally, we include a singleton $(\emptyset, i)$ for each input variable that is supplied by in-place bootstrapped data. Some composite schemata are provided directly from the domain which are called **pre-built composites**, and the BV includes pre-built composite inputs and outputs, whereas others are defined as a sub-plan relative to a composite schema or are saved by the user and are not included. There are currently around 50k items in the vocab, and the size of the vocab could be reduced for output/input pairs that occur too infrequently. The **binding id** refers to the index of a binding link.

Given a binding vocabulary and a base plan, the vectorized base plan is a sequence of binding ids whose sequence respects the partial ordering of operations in the plan, such that for every operation in the plan, all binding ids for its inputs occur before every binding id with any output. This is sufficient to characterize plans minus the ground data. Every vectorized plan is padded so they all share the same dimensionality. Now, any observation sequence can be represented as a sequence of base plan vectors and each plan-design action is a transition through vectorized plan-space.

We adapt an **autoencoding transformer** for our task and train on vectorized base plans as though they were sentences whose binding ids are token ids. The merit of autoencoding transformers like BERT (Devlin et al., 2018) is that they can leverage the tokens from anywhere in the sentence, not just sequentially like with autoregressive encoders like GPT. The main objective of the learning procedure is to mask tokens in the sentence and try to recover the original token via multiple layers of neural attention (Vaswani et al., 2017). Given that our base plans are partially ordered and bindings can be inserted anywhere in the plan, not just at the end of the sequence, the autoencoder is a more natural choice than an autoregressive approach. In addition, the resulting encoding can then be used and mixed with other approaches like LSTM and further fine-tuned on 1 or more downstream tasks.

Our adaptation works by treating bindings as tokens and doing **Masked Binding Modeling** (MBM) where we mask bindings in the vectorized plans and train the model to recover the binding id. Rather than using positional encoding which is sensitive to the length of the training inputs, we adapt a recently evaluated technique by Press et al. (2021) where they simply perform a linear interpolation to each dot product based on the distance from the query word in the sentence. We experiment with different methods for this where notably instead of just using raw token distance, we can use our own plan-based definition of distance that is based on a distance between binding links, which we call **binding distance**. In essence, the binding distance is based on converting the plan to a network where variables are nodes, binding links are edges, and also for every operation, each input is adjacent to each output, as shown in Figure 6. The resulting distance matrix is $[\text{MAX-SEQ-LEN}, \text{MAX-SEQ-LEN}]$ (max sequence length), tiled to the number of attention heads

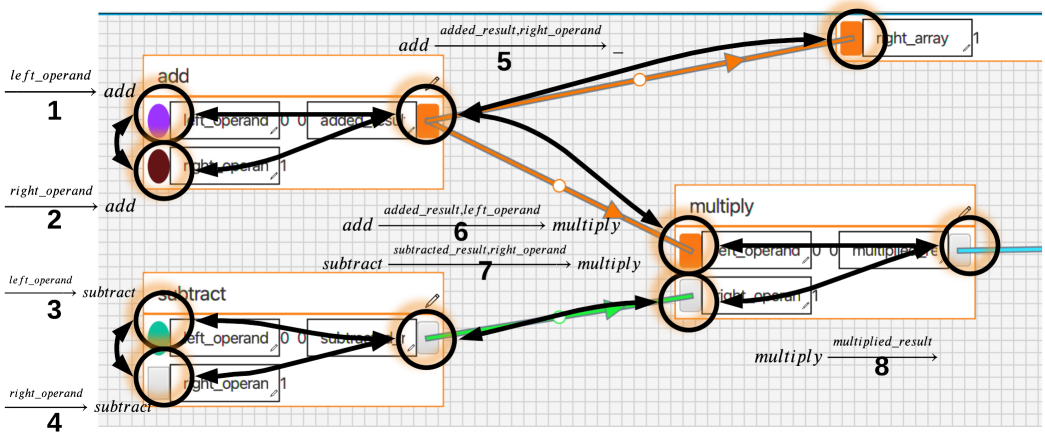

Figure 6. Illustration of a binding network overlaying a CERBREC plan. The binding network shows how binding distance is calculated between binding ids 1 through 8. By using the network form of the plan, distances reflect the path through the binding links and operations which may better capture the degree of relatedness between bindings.

and batch size to shape: [BATCH-SIZE, NUM-ATT-HEADS, MAX-SEQ-LEN, MAX-SEQ-LEN] so then it can simply be added to each layer. Figure 7 shows a snippet of the learning architecture and the shapes within a single layer of the network where the binding distances are added.

We make only two edits to BERT: (1) removing positional encoding and (2) adding distance interpolation weights. The rest of the architecture is exactly the same as BERT. The full architecture of our method is implemented within the CERBREC platform and is exportable to a JSON file which we will upload to a git repository, though the model is better displayed through the platform and can be viewed and edited for free by users. Because the binding vocabulary is comparable in size to a language vocabulary, our initial experiments start from the classic BERT-base model parameters which uses 12 encoder layers of size of 768 with 12 attention heads. Initially, we adopt a maximum sequence length of 128 bindings which currently is large enough for our pre-built models. Plans that are shorter are padded.

The learning procedure occurs in phases. First we perform MBM by taking complete and valid pre-built composite operations by masking 1 or more binding ids (in its sub-graph), running the vectorized base plan through the encoding layers, and learning to decode the masked binding(s). This results in pre-trained **binding embeddings**. When only one binding id is masked, the correct binding must include the only un-supplied input flaw and so the decoder's task is to recover the correct supplier. When multiple binding ids are masked, the task is not as straightforward.

The next phase is a self-supervised **plan-design fine-tuning task**. We start with valid and complete base plans, and then randomly perturb them for several iterations until they are at least incomplete and possibly invalid. The original plan before perturbations is considered the goal plan. The aforementioned design planner is used to select a flaw and perform all possible repairs for the expanded node in the plan-space search, where some of the child plans are correct next actions

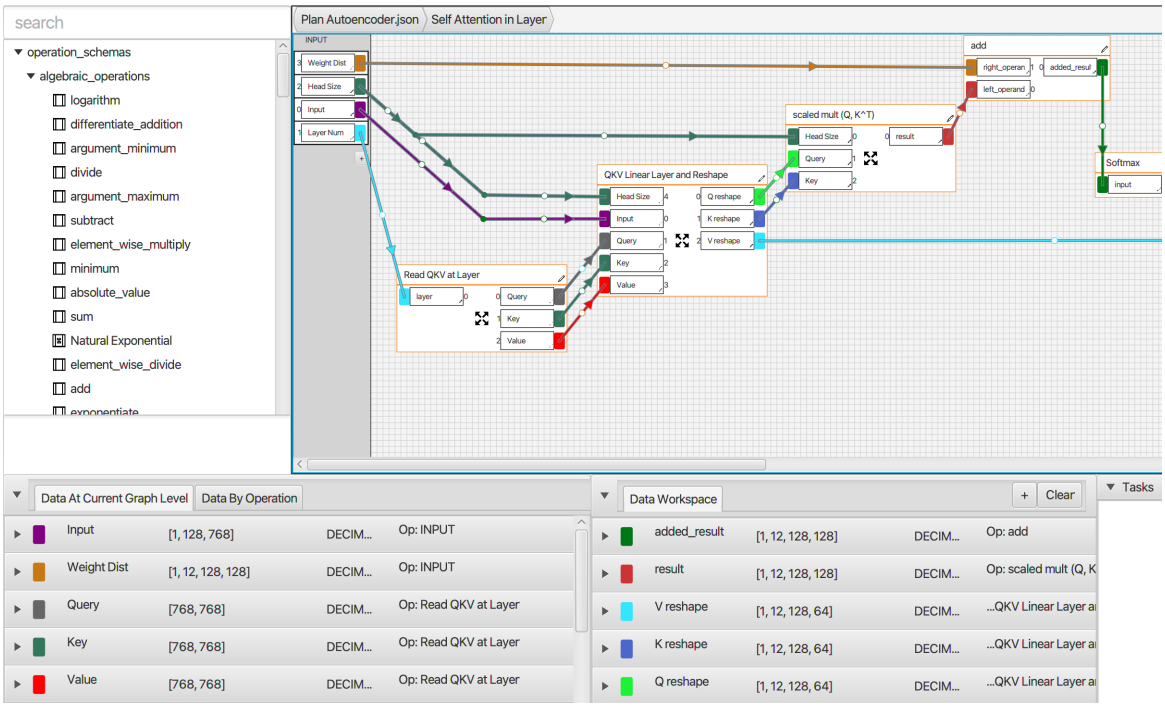

Figure 7. A snippet of the autoencoding transformer architecture at a single layer which is part of our model for solving PDGR for CERBREC; for clarification, CERBREC is shown here as an implementation of the solution, not itself the solution to a PDGR problem. The weighted distance interpolation is passed into each layer and added after the scaled dot product between the queries and the keys, and before applying softmax.

towards the goal (i.e., are on any optimal path to the goal). We learn task weights for a distance function between the child and the parent and fine-tune the binding embeddings. After every iteration on the design planner, we choose the correct child plan and repeat, irrespective of what was selected by the distance function. The resulting task weights are **plan-space distance weights** (PSDW). We are currently evaluating our approach on a held-out test set of perturbed base plans and will have preliminary results.

In a third phase, we augment our training process with session data. From the session data, we'll extract observation sub-sequences that start with some incomplete or invalid plan and end in a possible goal state. For example, we can start the observation sub-sequence when a user violates a constraint (invalid) and end either when the plan is valid and complete or a valid plan that occurs within n steps. We would then feed these sub-sequences into the fine tuning task. In future work, we hope to demonstrate that our learning architecture is sensitive to the context of users' goals and that requires session data.

The PDGR problem in the CERBREC platform is to learn a sequence of plan-design steps that produce a base plan, which is the inferred goal base plan of the user. Thus, our approach does not

explicitly use a goal library at run-time and is only trained on goal-like plans (and how to recover them). Next we step through the process of generating a prediction.

6. Prediction Generation

Now we step through a goal generation example with CERBREC to demonstrate how plan-design goal recognition is used to generate a recommendation. Using the plan in Figure 3 as an example, given a binding vocabulary, we would first convert the plan into a sequence of bindings:

1. $\xrightarrow{model\_dir}$ LIVE DATA
2. LIVE DATA $\xrightarrow{Text,ArrayTexts}$ BASIC BATCH TOKENIZER
3. BASIC BATCH TOKENIZER $\xrightarrow{Tokenized,Tokens}$ CONDENSED ONE HOT FROM VOCAB
4. CONDENSED ONE HOT FROM VOCAB $\xrightarrow{one\_hot,OneHot}$ SERVE CLASSIFIER
5. $\xrightarrow{ModelDir}$ SERVE CLASSIFIER
6. $[pad]$ ... $\times 123$

and map each to its index in the binding vocab. Each of the operations are composite schemata provided directly to the user (and thus their inputs and outputs are in the BV); if any composite was created by the user, then that composite would be replaced with its sub-plan and the binding sequence would include the bindings in the sub-plan and not include the parent composite.

The plan has an un-supplied binding link: $\xrightarrow{Vocab}$ CONDENSED ONE HOT FROM VOCAB. The constraints on this variable are that it is a 1D $text$ array. Since there are no other flaws in the plan, the design planner selects this flaw and expands the base plan for each way the flaw can be repaired without violating the constraint. This includes adding a binding link with existing variables that occur earlier in the plan such as "Text" and adding a binding link with an operation schema that is added to the graph. The existing base plan is the *parent* node, and each resulting plan to repair the selected flaw is a *child* node on the plan-space search space as with least-commitment planning (Weld, 1994). The set of valid action types depends on the type of flaw, so a Violated Constraint Flaw would include actions to remove operations and remove links.

The plan and all children plans are converted into the vectorized form and are compared through the following steps as depicted in Figure 8:

1 Vectorize the base plan and all children plans and shape into a single matrix that is $[n, 128]$ shape where n is the number of children plans $+1$.

2 Feed the input from [1] through the encoder layers to produce binding embeddings shaped $[n, 128, 768]$ and average them over the sequence to produce $[n, 768]$ plan embeddings.

3 Multiply all embeddings by the aforementioned PSDW weights which have shape $[768, h]$ where initially we chose $h = 1024$, resulting in $[n, 1024]$

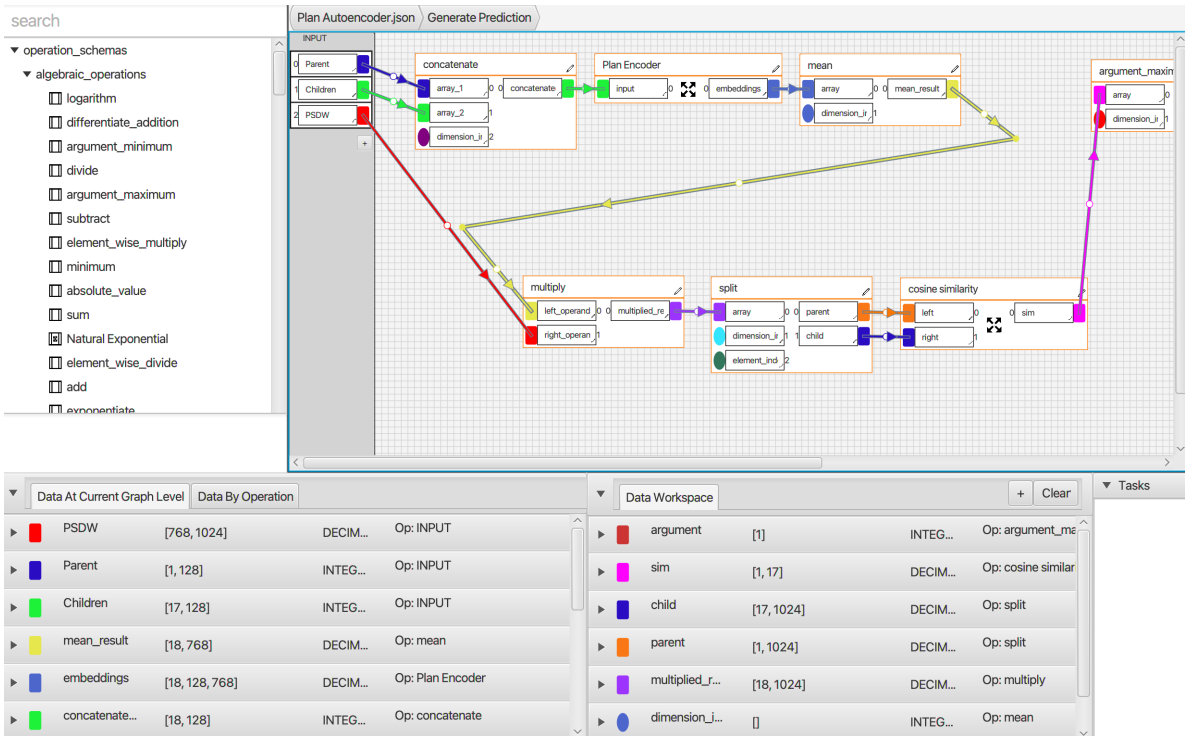

Figure 8. Steps to generate a prediction implemented in CERBREC. For clarification, this is work implemented with the CERBREC platform for solving a PDGR problem, but is not itself the result of the solution. The inputs are a vectorized parent base plan, 17 vectorized children, and the $[768, 1024]$ PSDW weight matrix. The result is the row index of the children plans and represents the plan to select in the plan-space search.

4 Separate into the parent plan embedding at index 0 and the $n - 1$ children.

5 Use the similarity score defined by $1 -$ COSINE DISTANCE between the parent embedding and child embeddings and ARG-MAX the index with the highest similarity. The child plan representing that embedding is the child to select.

7. Conclusion

We have presented our work-in-progress for plan-design goal recognition for (and with) the CERBREC platform. For clarification, we used our platform, CERBREC, as both the domain subject matter and as the implementation of our task solution. Users implement models with our platform, a process we call plan-design, where the resulting base plans are models whose steps are a sequence of operations over data. Our platform observes users as they plan-design and uses a neural encoder, also implemented with our platform, to make predictions that can be offered as suggestions

to the users. Thus, the platform is both introduced as the planning domain of interest and as an implementation of the solution.

The plan-design goal recognition problem, a sub-type of goal recognition problem where agent actions are plan-design steps, is relevant to a large class of applications where users are goal-directed and design plan-like artifacts. While we focused on data modeling with CERBREC, the class of applications includes other kinds of plan-like programs that need to be repeatable for new inputs such as for dialogue workflow design, coding, narrative design, game design, experiment design, and others. Though the problem is widely applicable, we have not seen any attempts to learn from a trace of plans created by the user during the design process.

The base planning domain for this work is the CERBREC domain which is defined to characterize the flow of data through a network of operations. One of our central contributions is that we introduce a novel way to encode these plans as vectors and learn plan embeddings. More work is needed to analyze if our approach extends to plans created with standard PDDL domains with STRIPS operators which are not provided with the capability of creating new ground objects as a result of execution without an extension like ground object properties (Pérez-Pinillos et al., 2011). Nevertheless, the CERBREC formalism for base planning may be desirable for other kinds of task-based data flow problems.

CERBREC is a tool that helps users develop models. We hope to make developing machine learning models less about software development and more about architecture design by addressing many of the challenges of model development (Zhang et al., 2019; Islam et al., 2019). CERBREC can help improve the model development by displaying constraint violations to the user, and future work will evaluate the time saving effect of the platform. Showing the constraint violation does not necessarily tell developers how they should solve it for their goal, and it can be challenging to solve problems with arrays that lead to efficiency gains with GPU hardware. Ultimately, the current work is our progress towards enabling CERBREC to give suggestions based on the user's model design trajectory.

References

Adhikari, B. (2021). *Intelligent simulink modeling assistance via model clones and machine learning*. Doctoral dissertation, Miami University.

Allen, J., Teng, C. M., & Galescu, L. (2018). Dialogue as collaborative problem solving: A case study. *Advances in Cognitive Systems*, *7*, 195–214.

Amos-Binks, A., & Cardona-Rivera, R. E. (2020). Goal elicitation planning: Acting to reveal the goals of others. *Proceedings of the 8th Annual Conference on Advances in Cognitive Systems*.

Amos-Binks, A., Cardona-Rivera, R. E., Dannenhauer, D., & Brewer, G. A. (2020). Anticipatory thinking: A new frontier for automated planning. *Proceedings of the 8th Annual Conference on Advances in Cognitive Systems*.

Armentano, M. G., & Amandi, A. A. (2009). Goal recognition with variable-order markov models. *Twenty-First International Joint Conference on Artificial Intelligence*.

Armentano, M. G., & Amandi, A. A. (2012). Modeling sequences of user actions for statistical goal recognition. *User Modeling and User-Adapted Interaction*, *22*, 281–311.

Blaylock, N., & Allen, J. (2003). Corpus-based, statistical goal recognition. *IJCAI* (pp. 1303–1308).

Bohnenberger, T., & Jameson, A. (2001). When policies are better than plans: Decision-theoretic planning of recommendation sequences. *Proceedings of the 6th international conference on Intelligent user interfaces* (pp. 21–24).

Borrajo, D., Gopalakrishnan, S., & Potluru, V. K. (2020). Goal recognition via model-based and model-free techniques. *arXiv preprint arXiv:2011.01832*.

Carberry, S. (2001). Techniques for plan recognition. *User modeling and user-adapted interaction*, *11*, 31–48.

Cardona-Rivera, R., & Young, R. (2015). Symbolic plan recognition in interactive narrative environments. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T., & Liu, X. (2020). A comprehensive study on challenges in deploying deep learning based software. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 750–762).

Datta, C., Jayawardena, C., Kuo, I. H., & MacDonald, B. A. (2012). Robostudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2352–2357). IEEE.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dinmore, M., & Boylls, C. (2010). Empirically-observed end-user programming behaviors in yahoo! pipes. *PPIG* (p. 17). Citeseer.

Du, F., Plaisant, C., Spring, N., Crowley, K., & Shneiderman, B. (2019). Eventaction: A visual analytics approach to explainable recommendation for event sequences. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *9*, 1–31.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*, 189–208.

Gao, Y., Liu, Y., Zhang, H., Li, Z., Zhu, Y., Lin, H., & Yang, M. (2020). Estimating gpu memory consumption of deep learning models. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1342–1352).

Geib, C. W., & Goldman, R. P. (2005). Partial observability and probabilistic plan/goal recognition. *Proceedings of the International workshop on modeling other agents from observations (MOO-05)* (pp. 1–6).

Ha, E. Y., Rowe, J. P., Mott, B. W., & Lester, J. C. (2011). Goal recognition with markov logic networks for player-adaptive games. *Seventh Artificial Intelligence and Interactive Digital En-*

tertainment Conference.

Hils, D. D. (1992). Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, *3*, 69–101.

Höller, D., Behnke, G., Bercher, P., & Biundo, S. (2018). Plan and goal recognition as htn planning. *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 466–473). IEEE.

Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., & Tonella, P. (2020). Taxonomy of real faults in deep learning systems. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (pp. 1110–1121).

Islam, M. J., Nguyen, G., Pan, R., & Rajan, H. (2019). A comprehensive study on deep learning bug characteristics. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 510–520).

Kautz, H. A., Allen, J. F., et al. (1986). Generalized plan recognition. *AAAI* (p. 5).

Keren, S., Gal, A., & Karpas, E. (2014). Goal recognition design. *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Keren, S., Xu, H., Kwapong, K., Parkes, D., & Grosz, B. (2020). Information shaping for enhanced goal recognition of partially-informed agents. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 9908–9915).

Langley, P. (1999). User modeling in adaptive interface. *UM99 User Modeling: Proceedings of the Seventh International Conference on User Modeling* (pp. 357–370). Springer.

Masters, P., & Sardina, S. (2018). Cost-based goal recognition for the path-planning domain. *IJCAI* (pp. 5329–5333).

Maynard, M., Duhamel, T., & Kabanza, F. (2019). Cost-based goal recognition meets deep learning. *arXiv preprint arXiv:1911.10074*.

Min, W., Mott, B. W., Rowe, J. P., Liu, B., & Lester, J. C. (2016). Player goal recognition in open-world digital games with long short-term memory networks. *IJCAI* (pp. 2590–2596).

Mott, B., Lee, S., & Lester, J. (2006). Probabilistic goal recognition in interactive narrative environments. *AAAI* (pp. 187–192).

Nelson, M. J., & Mateas, M. (2008). An interactive game-design assistant. *Proceedings of the 13th international conference on Intelligent user interfaces* (pp. 90–98).

Pattison, D., & Long, D. (2011). Accurately determining intermediate and terminal plan states using bayesian goal recognition. ICAPS.

Pereira, R., Oren, N., & Meneguzzi, F. (2017). Landmark-based heuristics for goal recognition. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Pérez-Pinillos, D., Fernández, S., & Borrajo, D. (2011). Modeling motivations, personality traits and emotional states in deliberative agents based on automated planning. *International Conference on Agents and Artificial Intelligence* (pp. 146–160). Springer.

Press, O., Smith, N. A., & Lewis, M. (2021). Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Price, T. W., Dong, Y., & Barnes, T. (2016). Generating data-driven hints for open-ended programming. *International Educational Data Mining Society*.

Price, T. W., Dong, Y., Zhi, R., Paaßen, B., Lytle, N., Cateté, V., & Barnes, T. (2019). A comparison of the quality of data-driven programming hint generation algorithms. *International Journal of Artificial Intelligence in Education*, *29*, 368–395.

Ramírez, M., & Geffner, H. (2009). Plan recognition as planning. *Twenty-First International Joint Conference on Artificial Intelligence*.

Ramírez, M., & Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Ramirez, M., & Geffner, H. (2011). Goal recognition over pomdps: Inferring the intention of a pomdp agent. *Twenty-second international joint conference on artificial intelligence*.

Sankaran, A., Aralikatte, R., Mani, S., Khare, S., Panwar, N., & Gantayat, N. (2017). Darviz: deep abstract representation, visualization, and verification of deep learning models. *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)* (pp. 47–50). IEEE.

Percie du Sert, N., et al. (2017). The experimental design assistant. *PLoS biology*, *15*, e2003779.

Shvo, M., & McIlraith, S. A. (2020). Active goal recognition. *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 9957–9966).

Smith, G., Whitehead, J., & Mateas, M. (2010). Tanagra: An intelligent level design assistant for 2d platformers. *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Sohrabi, S., Riabov, A. V., & Udrea, O. (2016). Plan recognition as planning revisited. *IJCAI* (pp. 3258–3264).

Stefnisson, I., & Thue, D. (2018). Mimisbrunnur: Ai-assisted authoring for interactive storytelling. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Tamilselvam, S. G., Panwar, N., Khare, S., Aralikatte, R., Sankaran, A., & Mani, S. (2019). A visual programming paradigm for abstract deep learning model development. *Proceedings of the 10th Indian Conference on Human-Computer Interaction* (pp. 1–11).

Thomas, J. M., & Young, R. M. (2006). Author in the loop: Using mixed-initiative planning to improve interactive narrative. *Workshop on AI Planning for Computer Games and Synthetic Characters*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems* (pp. 5998–6008).

Vered, M., & Kaminka, G. A. (2017). Heuristic online goal recognition in continuous domains. *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (pp. 4447–4454).

Weld, D. S. (1994). An introduction to least commitment planning. *AI magazine*, *15*, 27–27.

Winer, D., & Young, R. M. (2016a). Discourse-driven narrative generation with bipartite planning. *Proceedings of the 9th International Natural Language Generation conference* (pp. 11–20).

Winer, D. R., & Young, R. M. (2016b). Discourse-driven tellability goals for narrative planning. *4th Workshop on Goal Reasoning at IJCAI-2016*.

Winer, D. R., & Young, R. M. (2017). Merits of hierarchical story and discourse planning with merged languages. *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Yi, S., & Jung, K. (2017). A chatbot by combining finite state machine, information retrieval, and bot-initiative strategy. *Proc. Alexa Prize*.

Younes, H. L., & Simmons, R. G. (2003). VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, *20*, 405–430.

Zhang, R., Xiao, W., Zhang, H., Liu, Y., Lin, H., & Yang, M. (2020). An empirical study on program failures of deep learning jobs. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)* (pp. 1159–1170). IEEE.

Zhang, T., Gao, C., Ma, L., Lyu, M., & Kim, M. (2019). An empirical study of common challenges in developing deep learning applications. *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 104–115). IEEE.