







Synthesizing Trajectory Queries from Examples

Stephen Mell¹ , Favyen Bastani² , Steve Zdancewic¹ ,
and Osbert Bastani¹ 



¹ University of Pennsylvania, Philadelphia, PA 19104, USA
{sm1,stevez,obastani}@cis.upenn.edu

² Allen Institute for AI, Seattle, WA 98104, USA
favyenb@allenai.org



Abstract. Data scientists often need to write programs to process predictions of machine learning models, such as object detections and trajectories in video data. However, writing such queries can be challenging due to the fuzzy nature of real-world data; in particular, they often include real-valued parameters that must be tuned by hand. We propose a novel framework called QUIVR that synthesizes trajectory queries matching a given set of examples. To efficiently synthesize parameters, we introduce a novel technique for pruning the parameter space and a novel quantitative semantics that makes this more efficient. We evaluate QUIVR on a benchmark of 17 tasks, including several from prior work, and show both that it can synthesize accurate queries for each task and that our optimizations substantially reduce synthesis time.

1 Introduction

Over the past decade, deep neural networks (DNNs) have successfully solved challenging artificial intelligence problems [47, 70]. Abstractly, these models can be thought of as providing interfaces to real-world data—e.g., they can provide object classes [30, 47], detections [59, 60], and trajectories [10, 11, 83]. Then, these predictions are processed by programs, e.g., to identify driving patterns [5], events in TV broadcasts [28], or animal behaviors [67].

However, writing such programs can be challenging since they must still account for the fuzziness of real data. To do so, these programs typically include real-valued parameters that need to be manually tuned by the user. For example, consider a query over car trajectories designed to identify instances where one car turns in front of another. This query must capture the shape of the trajectory of both the turning car and the car crossing the intersection. In addition, the user must select the appropriate maximum duration from the first car changing lanes to the second car crossing the intersection. Even an expert would require significant experimentation to determine good parameter values; in our experience, it can take up to an hour to tune the parameters for a single query.

Appendices are available in the technical report [51].

© The Author(s) 2023

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13964, pp. 459–484, 2023.

https://doi.org/10.1007/978-3-031-37706-8_23

We focus on programs that query databases of trajectories output by an object tracker [5, 7, 8, 28, 40–42, 54]. Given a video, the tracker predicts the positions of objects in each frame (e.g., cars, people, or mice), as well as associations between detections of the same object across successive frames. Applications often require subsequent analysis of these trajectories. For example, in autonomous driving, when a risky scenario is encountered, engineers typically search for additional examples of that driving pattern to improve their planner [63, 64, 66]—e.g., cars driving too close [82] or stopping in the middle of the road [6]. Object tracking has also been used to track robots [58, 81], animals for behavioral analysis [12, 67, 75], and basketball players for sports analytics [67, 85].

We propose an algorithm for synthesizing queries over object trajectories given just a handful of input-output examples. A query takes as input a representation of a trajectory as a sequence of states (e.g., position, velocity, and acceleration) in successive frames of the video, and outputs whether the trajectory matches its semantics. Our query language is based on regular expressions—in particular, a query is a composition of a user-extensible set of predicates using the sequencing, conjunction, and iteration operators. For instance, trajectories might correspond to cars in a video; Fig. 1 shows a query for identifying cars turning at an intersection. As we discuss in Sect. 6, the full query language semantics is rich enough to subsume (variants of) Kleene algebras with tests (KAT) [46] and signal temporal logic (STL) [50]; however, such generality is seldom needed, so we use a pared-down query language that works well in practice.

Our algorithm performs enumerative search over the space of possible queries to identify ones that are consistent with the given examples. A key challenge in our setting is that our predicates have real-valued parameters that must also be synthesized. Thus, our strategy enumerates *sketches*, which are partial programs that only contain holes corresponding to real-valued parameters. For each sketch, we search over the space of real-valued parameters, while using an efficient pruning strategy to reduce the search space. At a high level, we use a quantitative semantics to directly compute “boundary parameters” at which a given example switches from being labeled positive to negative. Then, depending on the target label, we can prune the entire region of the search space on one side of these boundary parameters. We prove that this synthesis strategy comes with soundness and (partial) completeness guarantees.

We implement our approach in a system called QUIVR.¹ Our implementation focuses on videos from fixed-position cameras. While our language and synthesis algorithm are general, the predicates we design are tailored to specific settings. We evaluate QUIVR on identifying driving patterns in traffic videos, including ones inspired by recent work on autonomous driving [63, 64, 66], on behavior detection in a dataset of mouse trajectories [72], and on a synthetic task from the temporal logic synthesis literature [44]. We demonstrate how both our parameter pruning strategies and our query evaluation optimizations lead to substantial reductions in the running time of our synthesizer.

¹ QUIVR stands for QUery Induction for Video tRajectories.

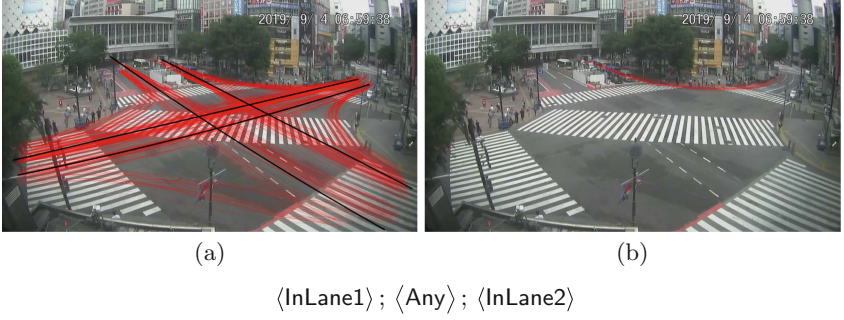


Fig. 1. (a) A video frame from a traffic camera, along with object trajectories (red) and manually annotated lanes (black). (b) The trajectories selected by the query (bottom), which selects cars turning at the intersection. (Color figure online)

In summary, our contributions are:

- A language for querying object trajectories (Sect. 3) and an algorithm for synthesizing such queries from examples (Sect. 4).
- An efficient parameter pruning approach based on a novel quantitative semantics (Sect. 4), yielding a $5.0\times$ speedup over the state-of-the-art quantitative pruning technique from the temporal logic synthesis literature.
- An implementation of our approach in QUIVR, and an evaluation of QUIVR on identifying driving behaviors in traffic camera video and mouse behaviors in a dataset of mouse trajectories (Sect. 5), demonstrating substantially better accuracy than neural network baselines.

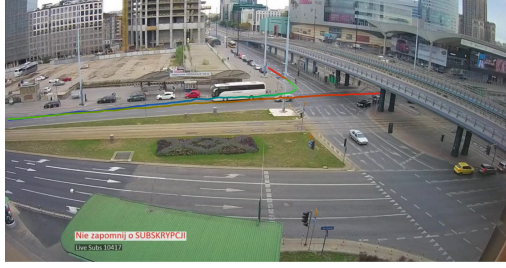
2 Overview

We consider a hypothetical scenario where an engineer is designing a control algorithm for an autonomous car and would like to identify certain driving patterns in video data. We show how they can use our framework to synthesize a query to identify car trajectories that exhibit a given behavior.

Video Data. Traffic cameras are a rich source of driving behaviors [5, 13, 61]; one dataset used in our evaluation is YTStreams [7], which includes video from several such cameras. Figure 1(a) shows a single frame from such a video; we have used an object tracker [83] to identify all car trajectories (in red).

Predicates. QUIVR assumes it is given a set of predicates that match portions of trajectories exhibiting behaviors of interest; during synthesis, it considers queries composed of these predicates. In Fig. 1(a), the engineer has manually annotated the lanes of interest in this video (black), to specify four $\text{InLane}K$ predicates that select trajectories of cars driving in each lane K visible in the video. Predicates may be configured by real-valued parameters. For example,

$$\langle \text{InLane1} \rangle \wedge \langle \text{DispLt}_\theta \rangle$$



$$\left(\langle \text{InLane1}(A) \rangle ; \langle \text{Any} \rangle ; \langle \text{InLane2}(A) \rangle \right) \wedge \langle \text{InLane2}(B) \rangle$$

Fig. 2. A single match (top) for the multi-object query (bottom) which captures one car, A , turning into a lane behind another car, B , that is in that lane. The trajectories change color from red to green as a function of time. As can be seen, the car making the right turn does so just after the car going straight passes through the intersection. (Color figure online)

searches for trajectories where the car stays in lane 1 for a period of time and the car has a displacement at most θ between the beginning and end of that period. Note that atomic predicates, like $\langle \text{Displ}_{\theta} \rangle$, can match multiple time-steps, whereas in formalisms like regular expressions and temporal logic, atomic predicates are over single time-steps. A key feature of our framework is that the set of available predicates is highly extensible, and the user can provide their own. See Sect. 5.1 for the predicates we use in our evaluation.

Synthesis. To specify a driving pattern, the engineer provides a small number of initial positive and negative examples of trajectories; then, QUIVR synthesizes a query that correctly labels these examples. In Fig. 1(b), we show the result of executing the query shown, which is synthesized to identify left turns in the data. Often, there are multiple queries consistent with the initial examples. While it may be hard for users to sift through the video for positive examples, it is usually easy for them to label a given trajectory. Thus, to disambiguate, QUIVR asks the user to label additional trajectories [19, 36, 62].

Multi-object Queries. So far, we have focused on queries that identify trajectories by processing each trajectory in isolation. A key feature of our framework is that users can express queries over multiple trajectories—for example,

$$(\langle \text{InLane1}(B) \rangle \wedge \langle \text{ChangeLane2To1}(A) \rangle) ; \langle \text{InFront}(A, B) \rangle.$$

This query says that car B is in lane 1 while car A changes from lane 2 to lane 1, and car A ends up in front of car B . Note that the predicates now include variables indicating which object they refer to, and the predicate $\text{InFront}(A, B)$ refers to multiple objects. An example of a pair of trajectories selected by a multi-object query is shown in Fig. 2.

3 Query Language

We describe our query language for matching object trajectories in videos. Our system first preprocesses the video using an object tracker to obtain trajectories, which are sequences $z = (x_0, x_1, \dots, x_{n-1})$ of states $x_i \in \mathcal{X}$. Then, a query Q in our language maps each trajectory z to a value $\mathbb{B} = \{0, 1\}$ indicating whether it matches z . Our language is similar to both STL and KAT. One key difference is that predicates are over arbitrary subsequences of z rather than single states x . In the main paper, we consider a simpler language, but in Appendix A we show how it can be extended to subsume both STL and KAT.

Trajectories. We begin by describing the input to a query in our language, which is the representation of one or more concurrent object trajectories in a video.

Consider a space \mathcal{S} corresponding to a single object detection in a single video frame—e.g., $s \in \mathcal{S} \subseteq \mathbb{R}^6$ might encode the 2D position, velocity, and acceleration of s in image coordinates. When considering m concurrent objects, let the space of *states* $\mathcal{X} = \mathcal{S}^m$, and then a *trajectory* $z \in \mathcal{Z} = \mathcal{X}^*$ is a sequence $z = (x_0, x_1, \dots, x_{n-1})$ of states of length $|z| = n$. We use the notation $z_{i:j} = (z_i, z_{i+1}, \dots, z_{j-1})$ to denote a subtrajectory of z .

Predicates. We assume a set of predicates Φ is given, where each predicate $\varphi \in \Phi$ matches trajectories $z \in \mathcal{Z}$; we use $\text{sat}_\varphi(z) \in \mathbb{B} = \{0, 1\}$ to indicate that φ matches z . As discussed below, queries in our language compose these predicates to match more complex patterns.

Next, predicates in our language may have real-valued parameters that must be specified. We denote such a predicate φ with parameter $\theta \in \mathbb{R}$ by φ_θ . To enable our synthesis algorithm to efficiently synthesize these real-valued parameters, we leverage the monotonicity in all such predicates we have used in our queries. In particular, we assume that the semantics of these predicates have the form

$$\llbracket \varphi_\theta \rrbracket(z) := \mathbb{1}(\iota_\varphi(z) \geq \theta),$$

where $\iota_\varphi : \mathcal{Z} \rightarrow \mathbb{R}$ is a scoring function. We also assume that the range of ι_φ is bounded (which can be achieved with a sigmoid function, if necessary). For example, for the predicate DisPLt_θ , we have $\iota_{\text{DisPLt}}(z) = -\|z_0 - z_{n-1}\|$. Thus, $\iota_{\text{DisPLt}}(z) \geq \theta$ says the total displacement is at most $-\theta$. We describe the predicates we include in Sect. 5.1; they can easily be extended.

Syntax. The syntax of our language is

$$Q ::= \varphi \mid Q; Q \mid Q^k \mid Q \wedge Q,$$

where $Q^k = Q; Q; \dots; Q$ (k times). That is, the base case is a single predicate φ , and queries can be composed using sequencing ($Q; Q$) and conjunction ($Q \wedge Q$). Operators for disjunction, negation, Kleene star, and STL’s “until” are discussed in Appendix A.2. We describe constraints imposed on our language during synthesis in Sect. 4.7.

Semantics. The satisfaction semantics of queries have type $\llbracket \cdot \rrbracket : \mathcal{Q} \rightarrow \mathcal{Z} \rightarrow \mathbb{B}$, where \mathcal{Q} is the set of all queries in our language, \mathcal{Z} is the set of trajectories, and

$$\begin{aligned}
\llbracket \varphi \rrbracket(z) &:= \text{sat}_\varphi(z) \\
\llbracket Q_1 \wedge Q_2 \rrbracket(z) &:= \llbracket Q_1 \rrbracket(z) \wedge \llbracket Q_2 \rrbracket(z) \\
\llbracket Q_1 ; Q_2 \rrbracket(z) &:= \bigvee_{k=0}^n \llbracket Q_1 \rrbracket(z_{0:k}) \wedge \llbracket Q_2 \rrbracket(z_{k:n})
\end{aligned}$$

Fig. 3. Satisfaction semantics of our query language; $z \in \mathcal{Z}$ is a trajectory of length n and $\varphi \in \Phi$ are predicates. Iteration (Q^k) can be expressed as repeated sequencing.

$\mathbb{B} = \{0, 1\}$. In particular, $\llbracket Q \rrbracket(z) \in \mathbb{B}$ indicates whether the query Q matches trajectory z . The semantics are defined in Fig. 3. The base case of a single predicate φ checks whether φ matches z ; conjunction $Q_1 \wedge Q_2$ checks if both conjuncts match; and sequencing $Q_1 ; Q_2$ checks if z can be split into $z = z_{0:k} z_{k:n}$ in a way that Q_1 matches $z_{0:k}$ and Q_2 matches $z_{k:n}$. The semantics can be evaluated in time $O(|Q| \cdot n^2)$.

4 Synthesis Algorithm

We describe our algorithm for synthesizing queries consistent with a given set of examples. It performs a syntax-guided enumerative search over the space of possible queries [3]. In more detail, it enumerates *sketches*, which are partial programs where only parameter values are missing. For each sketch, it uses a quantitative pruning strategy to compute the subset of the input parameters for which the resulting query is consistent with the given examples. A key contribution is how our algorithm uses quantitative semantics for quantitative pruning.

4.1 Problem Formulation

Partial Queries. A *partial query* is in the grammar

$$Q ::= ?? \mid \varphi_{??} \mid \varphi \mid Q ; Q \mid Q^k \mid Q \wedge Q.$$

Note that there are two kinds of holes: (i) a *predicate hole* $h = ??$ that can be filled by a sub-query Q , and (ii) a *parameter hole* $h = \varphi_{??}$ that can be filled by a real value $\theta_h \in \mathbb{R}$. We denote the predicate holes of Q by $\mathcal{H}_\varphi(Q)$, the parameter holes by $\mathcal{H}_\theta(Q)$, and let $\mathcal{H}(Q) = \mathcal{H}_\varphi(Q) \cup \mathcal{H}_\theta(Q)$. A partial query Q is a *sketch* (denoted $Q \in \mathcal{Q}_{\text{sketch}}$) [71] if $\mathcal{H}_\varphi(Q) = \emptyset$, and is *complete* (denoted $Q \in \mathcal{Q}$) if $\mathcal{H}(Q) = \emptyset$. For example, for $Q = \langle \text{DispLt}_{??1} \rangle \wedge ??2$, we have $\mathcal{H}_\theta(Q) = \{??1\}$ and $\mathcal{H}_\varphi(Q) = \{??2\}$. (We label each hole $h = ??i$ with an identifier $i \in \mathbb{N}$ to distinguish them.)

Refinements and Completions. Given query $Q \in \mathcal{Q}$, predicate hole $h \in \mathcal{H}_\varphi(Q)$, and production $R = Q \rightarrow f(Q_1, \dots, Q_k)$ we can *fill* h with R (denoted $Q' = \text{fill}(Q, h, R)$) by replacing h with $f(??1, \dots, ??k)$, where each $??i$ is a fresh hole,

and similarly given a parameter hole $h \in \mathcal{H}_\theta(Q)$ and a value $\theta_h \in \mathbb{R}$. We call Q' a *child* of Q (denoted $Q \rightarrow Q'$). Next, we call Q'' a *refinement* of Q (denoted $Q \xrightarrow{*} Q''$) if there exists a sequence $Q \rightarrow \dots \rightarrow Q''$; if furthermore $Q'' \in \bar{\mathcal{Q}}$, we say it is a *completion* of Q . For example, we have

$$??1 \rightarrow ??2; ??3 \rightarrow \langle \text{InLane1} \rangle; ??3 \rightarrow \dots$$

Here, $\langle \text{InLane1} \rangle; ??3$ is a child (and refinement) of $??2; ??3$ obtained by filling $??2$ with $Q \rightarrow \langle \text{InLane1} \rangle$ —i.e.,

$$\langle \text{InLane1} \rangle; ??3 = \text{fill}(??2; ??3, ??2, Q \rightarrow \langle \text{InLane1} \rangle).$$

Parameters. We let $\theta \in \mathbb{R}^{|\mathcal{H}_\theta(Q)|}$ denote a choice of parameters for each $h \in \mathcal{H}_\theta(Q)$, let $\theta_h \in \Theta_h \subseteq \mathbb{R}$ denote the parameter for hole h , and let Q_θ denote the query obtained by filling each $h \in \mathcal{H}_\theta(Q)$ with θ_h . Note that if $Q \in \mathcal{Q}_{\text{sketch}}$, then $Q_\theta \in \bar{\mathcal{Q}}$ is complete. For example, consider the sketch

$$Q = \langle \text{DisPLt}_{??1} \rangle \wedge \langle \text{MinLength}_{??2} \rangle.$$

This query has two holes, so its parameters are $\theta \in \mathbb{R}^2$. If $\theta = (3.2, 5.0)$, then $\theta_{??1} = 3.2$ is used to fill hole $??1$ and $\theta_{??2} = 5.0$ is used to fill $??2$. In particular,

$$Q_\theta = \langle \text{DisPLt}_{3.2} \rangle \wedge \langle \text{MinLength}_{5.0} \rangle.$$

Query Synthesis Problem. Given examples $W \subseteq \mathcal{W} = \mathcal{Z} \times \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$, our goal is to find a query $Q \in \bar{\mathcal{Q}}$ that correctly labels these examples—i.e.,

$$\psi_W(Q) := \bigwedge_{(z,y) \in W} (\llbracket Q \rrbracket(z) = y).$$

Thus, $\psi_W(Q)$ indicates whether Q is consistent with the labeled examples W . Our goal is to devise a synthesis algorithm that is sound and complete—i.e., it finds a query that satisfies $\psi_W(Q) = 1$ if and only if one exists.

4.2 Algorithm Overview

Our algorithm enumerates sketches $Q \in \mathcal{Q}_{\text{sketch}}$; for each one, it tries to compute parameter values θ such that the completed query Q_θ is consistent with W —i.e., $\psi_W(Q_\theta) = 1$. It can either stop once it has found a consistent query, or identify additional queries that are consistent with W . Algorithm 1 shows this high-level strategy—at each iteration, it selects a sketch Q , determines a region B of the parameter space containing consistent parameters $\theta \in B$, and adds (Q, B) to a list of consistent queries that solve the synthesis problem.

The key challenge is searching over the space of continuous parameters θ for a given sketch Q such that Q_θ is consistent with W . For efficiency, we rely heavily on pruning the search space. At a high level, consider evaluating a single candidate parameter θ on a single example $(z, y) \in W$ —i.e., check whether

Algorithm 1. Synthesizes consistent queries using the subroutine in Algorithm 2

```

1: procedure SYNTHESIZEQUERY( $W$ )
2:    $\mathcal{Q}_{\text{con}} \leftarrow \emptyset$ 
3:   for  $Q \in \mathcal{Q}_{\text{sketch}}$  do
4:      $B \leftarrow \text{SynthesizeParameters}(W, Q)$ 
5:      $\mathcal{Q}_{\text{con}} \leftarrow \{(Q, B)\}$ 
6:   return  $\mathcal{Q}_{\text{con}}$ 

```

$\llbracket Q_\theta \rrbracket(z) = y$. If this condition does not hold, then we can not only prune θ from the search space, but also a significant fraction of additional candidates. For instance, suppose $\llbracket Q_\theta \rrbracket(z) = 1$ but $y = 0$; if $\theta' \leq \theta$ (in all components), then by a monotonicity property we prove for our semantics, we also have $\llbracket Q_{\theta'} \rrbracket(z) = 1$. Thus, we can also prune θ' .

Previous work has leveraged this property to prune the search space [49, 53, 78]. Using a strategy based on binary search, for a given example $(z, y) \in W$, we can identify “boundary” parameters θ to accuracy ε in $O(\log(1/\varepsilon))$ steps—i.e., compute θ for which $\llbracket Q_{\theta-\varepsilon} \rrbracket(z) = 1$ and $\llbracket Q_{\theta+\varepsilon} \rrbracket(z) = 0$.

Our algorithm avoids this binary search process, which can lead to a significant speedup in practice. The key idea is to devise a quantitative semantics for queries that directly computes θ ; in fact, this quantitative semantics is closely related to robust temporal logic semantics, where the conjunction and disjunction of the satisfaction semantics are replaced with minimum and maximum, respectively.

4.3 Pruning with Boundary Parameters

We begin by describing how “boundary parameters” can be used to prune a portion of the search space over parameters. First, for *any* candidate parameters θ , we can prune parameters $\theta' \leq \theta$ (if $\llbracket Q_\theta \rrbracket(z) = 1$ and $y = 0$) or $\theta' \geq \theta$ (if $\llbracket Q_\theta \rrbracket(z) = 0$ and $y = 1$). Pruned regions of the parameter space take the form of hyper-rectangles, which we call *boxes*. For convenience, let $\vec{\infty} := (\infty, \dots, \infty)$.

Definition 1. Given $x, y \in \bar{\mathbb{R}}^d$, where $\bar{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$, a *box* is an axis-aligned half-open hyper-rectangle $[x, y] := \{v \mid x_i < v_i \leq y_i\} \subseteq \mathbb{R}^d$.

The key property ensuring that parameters prune boxes of the search space is that the semantics are monotonically decreasing in θ .

Lemma 1. *Given sketch Q , trajectory z , and two candidate parameters $\theta, \theta' \in \mathbb{R}^d$ such that $\theta \leq \theta'$ component-wise, we have $\llbracket Q_\theta \rrbracket(z) \geq \llbracket Q_{\theta'} \rrbracket(z)$.*

The proof follows by structural induction on the query semantics: the base case follows since the semantics $\mathbb{1}(\iota_\varphi(z) \geq \theta_k)$ for predicates is monotonically decreasing in θ_k , and the inductive case follows since conjunction and disjunction are monotonically increasing in their inputs (so they are also monotonically decreasing in θ_k). Below, we show how monotonicity ensures that we can prune whole regions of the search space if we find boundary parameters.

As an example, suppose we have two trajectories, z_0 of a car driving quickly and then slowly, and z_1 of a car driving slowly and then quickly, and that we are trying to synthesize a query for $W = \{(z_0, 0), (z_1, 1)\}$. For simplicity, we assume both $z_0 = (0.9, 0.6)$ and $z_1 = (0.5, 0.8)$ have just two time steps each, with just a single component representing velocity. Furthermore, we assume there is just a single predicate $\langle \text{VelGt}_\theta \rangle$ matching time steps where the velocity is at least θ , where θ is a real-valued parameter. Since $\langle \text{VelGt}_\theta \rangle$ matches single time steps, the satisfaction semantics is 0 except on trajectories of length 1, so:

$$\begin{array}{lll} \iota_{\text{VelGt}}((z_0)_{0:1}) = 0.9 & \iota_{\text{VelGt}}((z_0)_{1:2}) = 0.6 & \iota_{\text{VelGt}}((z)_{i:i}) = -\infty \\ \iota_{\text{VelGt}}((z_1)_{0:1}) = 0.5 & \iota_{\text{VelGt}}((z_1)_{1:2}) = 0.8 & \iota_{\text{VelGt}}((z)_{0:2}) = -\infty \end{array}$$

Consider the sketch $Q = \langle \text{VelGt}_{??1} \rangle; \langle \text{VelGt}_{??2} \rangle$. We can see that the candidate parameters $(0.5, 0.6)$ satisfy $\llbracket Q_{(0.5, 0.6)} \rrbracket(z_1) = 1$:

$$\begin{aligned} \llbracket Q_{(0.5, 0.6)} \rrbracket((z_1)_{0:n}) &= \bigvee_{k=0}^2 \llbracket \langle \text{VelGt}_{0.5} \rangle \rrbracket((z_1)_{0:k}) \wedge \llbracket \langle \text{VelGt}_{0.6} \rangle \rrbracket((z_1)_{k:n}) \\ &= \llbracket \langle \text{VelGt}_{0.5} \rangle \rrbracket((z_1)_{0:1}) \wedge \llbracket \langle \text{VelGt}_{0.6} \rangle \rrbracket((z_1)_{1:2}) \\ &= \mathbb{1}(0.5 \geq 0.5) \wedge \mathbb{1}(0.8 \geq 0.6) \\ &= 1, \end{aligned}$$

where the second equality holds because $\langle \text{VelGt}_\theta \rangle$ matches only length-1 trajectories, so the $k = 0$ and $k = 2$ cases evaluate to 0. Since the semantics are monotonically decreasing, we have $\llbracket Q_\theta \rrbracket(z_1) = 1$ for any $\theta \in [(-\infty, -\infty), (0.5, 0.6)]$.

Notice, however, that if we were to move any $\varepsilon' > 0$ upward, we would have $\llbracket Q_{(0.5+\varepsilon_1, 0.6+\varepsilon_2)} \rrbracket(z_1) = \mathbb{1}(0.5 \geq 0.5 + \varepsilon_1) \wedge \mathbb{1}(0.8 \geq 0.6 + \varepsilon_2) = 0$. So we know $\llbracket Q_\theta \rrbracket(z_1) = 0$ for any $\theta \in [(0.5, 0.6), (\infty, \infty)]$. This is because $(0.5, 0.6)$ lies on the boundary between $\{\theta' \mid \llbracket Q_{\theta'} \rrbracket(z) = 0\}$ and $\{\theta' \mid \llbracket Q_{\theta'} \rrbracket(z) = 1\}$. This boundary plays a key role in our algorithm.

Definition 2. Given a sketch Q with d parameter holes and a trajectory z , we say $\theta \in \mathbb{R}^d \cup \{\perp, \top\}$ is a *boundary parameter* if one of the following holds:

- $\theta \in \mathbb{R}^d$ and $\llbracket Q_\theta \rrbracket(z) = 1$, but $\llbracket Q_{\theta'} \rrbracket(z) = 0$ for all $\theta' \in [\theta, \infty]$
- $\theta = \perp$ and $\llbracket Q_{\theta'} \rrbracket(z) = 0$ for all $\theta' \in [-\infty, \theta]$
- $\theta = \top$ and $\llbracket Q_{\theta'} \rrbracket(z) = 1$ for all $\theta' \in [-\infty, \theta]$

In the first case, by monotonicity, we also have $\llbracket Q_{\theta'} \rrbracket(z) = 1$ for all $\theta' \in [-\infty, \theta]$; thus, θ lies on the boundary between parameters θ' where $Q_{\theta'}$ evaluates to 1 and those where it evaluates to 0. The second and third cases are where $Q_{\theta'}$ always evaluates to 0 and 1, respectively.

Given a boundary parameter θ for an example $(z, y) \in W$, we can prune $[\theta, \infty]$ if $y = 1$ or $[-\infty, \theta]$ if $y = 0$. Intuitively, boundary parameters provide optimal pruning along a fixed direction in the parameter space. Thus, our algorithm focuses on computing boundary parameters for pruning.

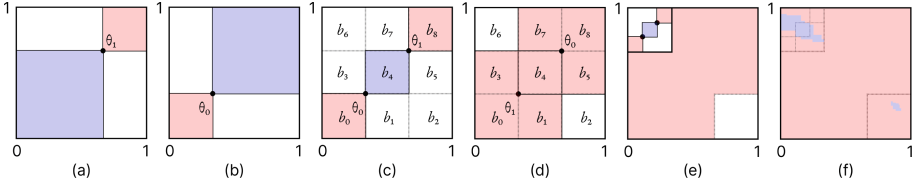


Fig. 4. (a) shows a boundary parameter, θ_1 , for z_1 , and a region that is inconsistent with z_1 and can be pruned (red), as well as a region that is consistent with it (blue). (b) similarly shows a boundary parameter θ_0 for z_0 . (c) shows the pruning pair composed of θ_0 and θ_1 , a region consistent with both (blue), and regions inconsistent with either (red). (d) is the same as (c), but if θ_0 and θ_1 swapped places. The labels b_0 through b_8 denote analogous boxes in (c) & (d). (e) shows how, if (d) were the result of the first step of search and b_6 were chosen next, search could proceed. (f) shows ground truth consistent (blue) and inconsistent (red) regions that the search process in (d) & (e) might converge toward. (Color figure online)

In Fig. 4(a), if θ_1 is a boundary parameter for z_1 , we know that the blue region satisfies z_1 , and thus is consistent with the label 1, while the red region dissatisfies z_1 , and thus is inconsistent with the label 1. Similarly, in Fig. 4(b), if θ_0 is a boundary parameter for z_0 , we know that the red satisfies z_1 , and thus is inconsistent with the label 0, while the blue dissatisfies z_0 , and thus is consistent with the label 0.

4.4 Pruning with Pairs of Boundary Parameters

To extend pruning to the entire dataset W , we could simply prune the union of the individual pruned regions for each $(z, y) \in W$. However, one important feature of our approach is that we can also establish regions of the parameter space where the parameters are guaranteed to be consistent with W . To formalize this idea, we introduce the concept of a “pruning pair”, which is a pair of boundary parameters which might allow us to find such a consistent region.

Definition 3. Given a sketch Q and a dataset W , a pair of boundary parameters $\theta^-, \theta^+ \in \mathbb{R}^d \cup \{\perp, \top\}$ is a *pruning pair* for Q and W if all of the following hold:

- θ^+ is a boundary parameter for some $z \in W^+$ and, for all $z' \in W^+$ such that $z' \neq z$, we have $\llbracket Q_{\theta^+} \rrbracket(z') = 1$.
- θ^- is a boundary parameter for some $z \in W^-$ and, for all $z' \in W^-$ such that $z' \neq z$, we have $\llbracket Q_{\theta^-} \rrbracket(z') = 0$.
- $\theta^- < \theta^+$ or $\theta^- \geq \theta^+$.

If $\theta^- < \theta^+$, the pruning pair (θ^-, θ^+) is *consistent*, and *inconsistent* otherwise.

Our algorithm searches for pruning pairs along a fixed direction—i.e., it considers a curve $L \subseteq \mathbb{R}^d$ and looks for the following pruning pair along L :

$$\theta^+ = \sup \left\{ \theta \in L \mid \bigwedge_{z \in W^+} \llbracket Q_\theta \rrbracket(z) = 1 \right\}, \quad \theta^- = \inf \left\{ \theta \in L \mid \bigwedge_{z \in W^-} \llbracket Q_\theta \rrbracket(z) = 0 \right\}.$$

Intuitively, θ^+ is the largest θ that correctly classifies all positive examples, and conversely for θ^- . We restrict to curves L that are monotonically increasing in all components, in which case the supremum and infimum are well defined since L comes with a total ordering (from its smallest point to its largest) that is consistent with the standard partial order on \mathbb{R}^d . Then, (θ^-, θ^+) form a pruning pair: since θ^+ is a boundary parameter for z , if we take θ^+ to be any larger, then we must have $\llbracket Q_\theta \rrbracket(z) = 0$ for some $z \in W^+$, and similarly for θ^- .

Given a curve L , we can compute an approximation to θ^+ and θ^- via binary search. However, our algorithm avoids the need to do so by directly computing θ^+ and θ^- using a quantitative semantics, which we describe in Sect. 4.6.

Figure 4(c) shows how the pair of boundary parameters θ_0 for z_0 and θ_1 for z_1 (where L is the diagonal line) prunes the parameter space. The blue region is guaranteed to be consistent with W , as it is the intersection of the region below θ^+ , which must satisfy $\llbracket Q_\theta \rrbracket(z_1) = 1$, and the region above θ^- , which must satisfy $\llbracket Q_\theta \rrbracket(z_0) = 0$. Conversely, the red regions are inconsistent with either z_0 or z_1 , and therefore with W . Thus, the red regions can be pruned, whereas the blue regions are solutions to our synthesis problem. Note that the red region is the union of the red regions in Fig. 4(a) and (b), whereas the blue region is the intersection of the blue regions in Fig. 4(a) and (b).

This pattern holds for any consistent pruning pair $(\theta^- < \theta^+)$; if instead the pair is inconsistent $(\theta^- \geq \theta^+)$, then the resulting pattern is illustrated in Fig. 4(d); in this case, we can prune the red regions as before, but there is no blue region of solutions. In general, for a d dimensional parameter space, a pruning pair divides the parameter space into 3^d boxes (i.e., for each dimension, the box can be below, in line with, or above the center box). The regions below θ^- and above θ^+ can be pruned, and the region between θ^- and θ^+ (if one exists) contains synthesis solutions. Precisely, it follows from the definitions and monotonicity that:

Lemma 2. *Every $\theta \in [-\infty, \theta^-]$ and $\theta \in [\theta^+, \infty]$ is inconsistent with W , and every $\theta \in [\theta^-, \theta^+]$ is consistent with W .*

The remaining boxes need to be further analyzed by our algorithm.

4.5 Pruning Parameter Search Algorithm

Next, we describe Algorithm 2, which searches over the space of parameters to fill a sketch Q for a given dataset W . The algorithm uses a subroutine that takes a box and returns a pruning pair in that box, which we describe in Sect. 4.6. Given this subroutine, our algorithm maintains a work-list of “unknown” boxes (i.e., unknown whether parameters in these boxes are consistent or inconsistent with W). At each iteration, it pops a box from the work-list (in first-in-first-out order), uses the given subroutine to find a pruning pair inside that box, applies the pruning procedure described in the previous section, and then adds each new unknown box to the work-list.

For the last step, the current box b is divided into 3^d smaller boxes. The box $b_{\text{center}} := [\min\{\theta^-, \theta^+\}, \max\{\theta^-, \theta^+\}]$ is pruned (added to B_{inc}) if the

Algorithm 2. Synthesizes consistent parameters for a given sketch

```

1: procedure SYNTHESIZEPARAMETERS( $W, Q$ )
2:    $B_{\text{con}} \leftarrow \emptyset; B_{\text{inc}} \leftarrow \emptyset, B_{\text{unk}} \leftarrow \{b_{\text{initial}}\}$ 
3:   for  $i \in \{1, \dots, N\}$  do
4:      $b \leftarrow \text{Pop}(B_{\text{unk}})$ 
5:      $\theta^-, \theta^+ \leftarrow \text{ComputePruningPair}(W, Q, b)$ 
6:      $b_{\text{center}} \leftarrow \lfloor \min\{\theta^-, \theta^+\}, \max\{\theta^-, \theta^+\} \rfloor$ 
7:      $b_{\text{lower}}, b_{\text{upper}}, B_{\text{incomp}}, B_{\text{extra}} \leftarrow \text{SplitBox}(b, b_{\text{center}})$ 
8:     if  $\theta^- < \theta^+$  then
9:        $B_{\text{con}} \leftarrow B_{\text{con}} \cup \{b_{\text{center}}\}$ 
10:       $B_{\text{inc}} \leftarrow B_{\text{inc}} \cup \{b_{\text{lower}}, b_{\text{upper}}\}$ 
11:       $B_{\text{unk}} \leftarrow B_{\text{unk}} \cup B_{\text{incomp}} \cup B_{\text{extra}}$ 
12:     else if  $\theta^- \geq \theta^+$  then
13:        $B_{\text{inc}} \leftarrow B_{\text{inc}} \cup \{b_{\text{center}}, b_{\text{lower}}, b_{\text{upper}}\} \cup B_{\text{extra}}$ 
14:        $B_{\text{unk}} \leftarrow B_{\text{unk}} \cup B_{\text{incomp}}$ 
15:   return  $B_{\text{con}}$ 

```

pair (θ^-, θ^+) is inconsistent, and contains solutions to the synthesis problem otherwise (added to B_{con}). The boxes $b_{\text{lower}} = \lfloor -\infty, \min\{\theta^-, \theta^+\} \rfloor$ and $b_{\text{upper}} = \lfloor \max\{\theta^-, \theta^+\}, \infty \rfloor$ are always pruned. The boxes $b \in B_{\text{incomp}}$ are the remaining corners of b , and always have indeterminate consistency (added to B_{unk}). The remaining boxes $b \in B_{\text{extra}}$ are indeterminate if (θ^-, θ^+) is consistent, and inconsistent otherwise. In our example, if the first step of the algorithm yielded Fig. 4(d), then the second step might pop b_6 and yield Fig. 4(e).

The following soundness result follows directly from Lemma 2.

Theorem 1. *In Algorithm 2, every $\theta \in B_{\text{con}}$ is consistent with W for Q , and every $\theta \in B_{\text{inc}}$ inconsistent.*

In addition, the algorithm is complete for almost all parameters:

Theorem 2. *The Lebesgue measure of $\{\theta \in b \mid b \in B_{\text{unk}}\} \rightarrow 0$ as $N \rightarrow \infty$.*

See Appendix D.1 for the proof. In other words, all parameters outside a subset of measure zero are eventually classified as consistent or inconsistent; intuitively, the parameters that may never be classified are the ones along the decision boundary. This result holds since at any search depth, the fraction of the parameter space pruned can be lower-bounded.

4.6 Computing Pruning Pairs via Quantitative Semantics

The pruning algorithm depends on the ability to compute, given a box b , a pruning pair (θ^-, θ^+) on the restriction of the parameter space to b . Recall that θ^+ must be a boundary parameter for some $z^+ \in W^+$ and must satisfy $\llbracket Q_{\theta^+} \rrbracket(z) = 1$ for all other $z \in W^+$, and θ^- must be a boundary parameter for some $z^- \in W^-$, and must satisfy $\llbracket Q_{\theta^-} \rrbracket(z) = 0$ for all other $z \in W^-$.

Given a box $b = [\theta_{\min}, \theta_{\max}]$, our algorithm takes $L \subseteq \mathbb{R}^d$ to be the diagonal from θ_{\min} to θ_{\max} and computes the pruning pair along L . We can naïvely

$$\begin{aligned}
 \llbracket \varphi_{??i} \rrbracket_{v,u}^q(z) &:= \frac{\iota_\varphi(z) - v_i}{u_i} \\
 \llbracket \varphi \rrbracket_{v,u}^q(z) &:= \begin{cases} \infty & \text{if } \text{sat}_\varphi(z) = 1 \\ -\infty & \text{if } \text{sat}_\varphi(z) = 0. \end{cases} \\
 \llbracket Q_1 \wedge Q_2 \rrbracket_{v,u}^q(z) &:= \min\{\llbracket Q_1 \rrbracket_{v,u}^q(z), \llbracket Q_2 \rrbracket_{v,u}^q(z)\} \\
 \llbracket Q_1 ; Q_2 \rrbracket_{v,u}^q(z) &:= \max_{0 \leq k \leq n} \min\{\llbracket Q_1 \rrbracket_{v,u}^q(z_{0:k}), \llbracket Q_2 \rrbracket_{v,u}^q(z_{k:n})\}
 \end{aligned}$$

Fig. 5. The quantitative semantics of our language, taking a sketch Q , trajectory z , parameter $v \in \mathbb{R}^d$, and positive vector $u \in \mathbb{R}_{>0}^d$. n is the length of z .

use binary search: for θ^+ , we search for the parameters where $\bigwedge_{z \in W^+} \llbracket Q_\theta \rrbracket(z)$ transitions from 0 to 1, and similarly for θ^- and $\bigwedge_{z \in W^-} \neg \llbracket Q_\theta \rrbracket(z)$.

Instead, by leveraging a quantitative semantics, we can directly compute θ^+ and θ^- , thereby reducing computation time substantially. Given a sketch Q , trajectory z , parameter $v \in \mathbb{R}^d$, and positive vector $u \in \mathbb{R}_{>0}^d$, we devise a quantitative semantics $\llbracket Q \rrbracket_{v,u}^q(z) \in \mathbb{R}$ such that the parameter $\llbracket Q \rrbracket_{v,u}^q(z) \cdot u + v$ is a boundary parameter. Intuitively, this semantics computes, starting at v , how many u -sized steps must be taken to reach the boundary. (For the uses in our algorithm, the number of steps is always in $[0, 1]$.) Then, for a box $b = [\theta_{\min}, \theta_{\max}]$, we can take $v = \theta_{\min}$ and $u = \theta_{\max} - \theta_{\min}$, and compute

$$\theta^+ = \left(\min_{z \in W^+} \llbracket Q \rrbracket_{v,u}^q(z) \right) \cdot u + v, \quad \theta^- = \left(\max_{z \in W^-} \llbracket Q \rrbracket_{v,u}^q(z) \right) \cdot u + v.$$

We define the quantitative semantics in Fig. 5. The base case of $\varphi_{??}$ adjusts and rescales ι_φ by v and u , and the other cases replace conjunction and disjunction in the satisfaction semantics with minimum and maximum. We have the following key result (where $\infty \cdot u := \top$, $-\infty \cdot u := \perp$, $\top + v := \top$, and $\perp + v := \perp$):

Theorem 3. *For a sketch Q , trajectory z , parameter $v \in \mathbb{R}^d$, and positive vector $u \in \mathbb{R}_{>0}^d$, we have that $\llbracket Q \rrbracket_{v,u}^q(z) \cdot u + v$ is a boundary parameter of z for Q .*

See Appendix D.2 for the full proof. For intuition, consider $\theta_{\min} = \vec{0}$ and $\theta_{\max} = \vec{1}$ (i.e., the current box $b \subseteq \mathbb{R}^d$ is the unit hypercube). Then, $v = \vec{0}$ and $u = \vec{1}$, so $\llbracket Q \rrbracket_{v,u}^q$ reduces to the standard max-min quantitative semantics for temporal logic [25].

Now, if we consider the satisfaction semantics of a base predicate $\llbracket \varphi_{\theta_i} \rrbracket = \mathbb{1}(\iota_\varphi(z) \geq \theta_i)$, then the value of θ_i where the semantics flips is just $\iota_\varphi(z)$. So any parameter with i -th component $\iota_\varphi(z)$ is a boundary parameter, and since L has the same slope in all dimensions, the boundary parameter along L is $\iota_\varphi(z) \cdot \vec{1} + \vec{0} = \llbracket \varphi_{??i} \rrbracket_{0,\vec{1}}^q(z) \cdot \vec{1} + \vec{0}$.

In the inductive cases, it suffices to show that we can replace conjunction and disjunction with minimum and maximum in the semantics. Since the satisfaction

semantics is monotonically decreasing, as we move upward along L , at some point we will transition from 1 to 0. A conjunction becomes 0 when either conjunct becomes 0, so the transition will occur when we hit the first of the conjuncts' transition points (their minimum). Dually, a disjunction becomes 0 when both disjuncts become 0, so we will transition at the last of the disjuncts' transition points (their maximum).

Finally, the intuition behind u and v is that they “preprocess” the parameters so that we evaluate along the diagonal of the current box instead of $\left[\vec{0}, \vec{1}\right]$.

4.7 Implementation

We implement our approach in a system called QUIVR. It begins by running Algorithm 1 on a small number of labeled examples.

Active Learning. With a small number of examples, there are typically many queries that are consistent with the labels, and yet which disagree on the labels of the remaining data. To disambiguate, we use an active learning strategy, asking the user to label specific trajectories that we choose, which are then added to our set of labeled examples. Queries that are not consistent with the new label are discarded. The labeling process continues until the set of consistent queries agrees on the labels of all unlabeled data.

When choosing the trajectory z^* to query the user for next, we select the one on which the set of consistent queries C disagrees most—i.e.,

$$z^* = \arg \min_{z \in Z} \left| J(z) - \frac{1}{2} \right|,$$

where

$$J(z) := |C|^{-1} \sum_{Q_\theta \in C} \mathbb{1}(\psi_{(z,y)}(Q_\theta))$$

is the fraction of consistent queries that predict a positive label for trajectory z .

Search Implementation. In some cases, searching for consistent parameters may take a very long time. To improve performance, we impose a timeout: for each sketch, we pause search if either: (i) we find some consistent box of parameters or (ii) we've exceeded 25 steps. In both cases, we save the sets of consistent, inconsistent, and unknown boxes. At each step of active learning, the newly labeled example may render previously consistent parameters inconsistent, so we mark all consistent boxes as unknown. We then resume search, again until (i) we find some consistent box (which may be the same one we had before), or (ii) we again exceed 25 steps.

Note that while this timeout may cause us to query the user more often than is strictly necessary, it does not affect either the soundness or completeness of our approach, as we continue search after querying the user.

Complete Query Selection. Active learning and evaluation of F_1 scores (in Sect. 5) both require complete queries with specific parameters, rather than sketches

Table 1. The predicates used for the YTStreams dataset.

Predicate	Description
$\text{InLaneK}(A)$	Whether, at every time-step in the interval, object A is sufficiently close to the annotated curve for lane K and A’s movement direction is sufficiently in line with the curve for K.
DurationNotShort	Whether the interval spans at least 5 seconds.
AvgAccelGt_θ	Whether the average acceleration over the interval is at least θ .
DistanceLt_θ	Whether, at every time-step in the interval, the distance between the two objects is less than θ .
$\text{SpeedRatioGt}_\theta(A, B)$	Whether, at every time-step in the interval, the speed of A divided by the speed of B is at least θ .
$\text{DisplT}_\theta(A)$	Whether the distance between the position in the first frame of the interval and the position in the last frame is less than θ .

with boxes of parameters. Since the set C of consistent queries is infinitely large, we instead we use one query for each sketch that is known to have consistent parameters (sketches where search timed-out are thus not included). For those sketches, we pick the middle of the box of known-consistent parameters.

5 Evaluation

We demonstrate how our approach can be used to synthesize queries to solve interesting tasks: in particular, we show that (i) given just a few initial examples, it can synthesize queries that achieve good performance on a held-out test set, and (ii) our optimizations significantly reduce the synthesis time.

5.1 Experimental Setup

Datasets. We evaluate on two datasets of object trajectories: YTStreams [7], consisting of video and extracted object trajectories from fixed-position traffic cameras, and MABe22 [72], consisting of trajectories of up to three mice interacting in a laboratory setting. We also evaluate on a synthetic maritime surveillance task from the STL synthesis literature [44]. On YTStreams, we use two traffic cameras, one in Tokyo and one in Warsaw, and we consider single cars or pairs of cars. On MABe22, we consider pairs of mice. For the predicates used, see Table 1 for YTStreams, Appendix Table 5 for MABe22, and Appendix Table 6 for maritime surveillance.

Tasks. On YTStreams, we manually wrote 5 ground truth queries. Several queries apply to multiple configurations (e.g., different pairs of lanes), resulting in 10 queries total (tasks H-Q in Table 2). The real-valued parameters were chosen manually, by visually examining whether they were selecting the desired trajectories. These queries cover a wide range of behaviors; for instance, they can

Table 2. Ground-truth queries for the YTStreams dataset. “IDs” indicates which tasks are instances of a given query. Multiple instantiations correspond to different lanes being used for “lane 1” and “lane 2”. The first is a one-object Shibuya query, the second is a one-object Warsaw query, and the rest are two-object Warsaw queries.

IDs	Query
H, I, J, K	$\langle \text{InLane1}(A) \rangle; \langle \text{Any} \rangle; \langle \text{InLane2}(A) \rangle$ Matches cars that turn, starting in lane 1 and ending in lane 2.
L, M	$\langle \text{InLane1}(A) \rangle \wedge \langle \text{AvgAccelGt}(A) \rangle_{??} \wedge \langle \text{DurationNotShort} \rangle$ Matches cars that accelerate for a significant period of time while in lane 1.
N	$(\langle \text{InLane1}(A) \rangle; \langle \text{Any} \rangle; \langle \text{InLane2}(A) \rangle) \wedge \langle \text{InLane2}(B) \rangle$ Matches pairs of cars where car B is in lane 2 for the entire duration of A turning from lane 1 into lane 2.
O, P	$\langle \text{InLane1}(A) \rangle \wedge \langle \text{InLane2}(B) \rangle \wedge \langle \text{DurationNotShort} \rangle \wedge \langle \text{SpeedFactorGt}(A, B) \rangle_{??}$ Matches pairs of cars in parallel lanes, 1 and 2, where car A is going faster than car B for a significant period of time.
Q	$\langle \text{InLane1}(A) \rangle \wedge \langle \text{InLane2}(B) \rangle \wedge \langle \text{DurationNotShort} \rangle \wedge \langle \text{DistanceLt}(A, B) \rangle_{??}$ Matches pairs of cars in parallel lanes, 1 and 2, where the cars are close for a significant period of time



Fig. 6. Trajectories selected by multi-object queries. Each image shows two objects; the color of each one changes from red to green to denote the progression of time. Left: Unprotected right turn into lane with oncoming traffic. Middle: Bottom car drives faster than the top one and passes it. Right: One car driving closely behind the other. (Color figure online)

capture behaviors such as human drivers making unprotected turns, an important challenge for autonomous cars [64], as well as cars trying to pass [66]. We show examples of trajectories selected by three of our multi-object queries in Fig. 6. MABe22 describes 9 queries for scientifically interesting mouse behavior. We implemented the 6 most complex to use as ground truth queries (tasks A-F in Appendix Table 7). The maritime surveillance task has trajectory labels and so does not need a ground truth query (task G).

Synthesis. For each task, we divide the set Z of all trajectories into a train set Z_{train} and a test set Z_{test} , using trajectories in the first half of the video for training, and those in the second half for testing. We randomly sample a set of initial labeled examples W from Z_{train} , with 2 samples being positive and 10 being negative, and then actively label 25 additional examples from Z_{train} . For YTStreams and MABe22, labels are from the ground truth query.

Table 3. F_1 score after n steps of active learning, with our algorithm for selecting tracks to label (“ Q ”), an active learning ablation (“ R ”), an LSTM (“ L ”), and a transformer (“ T ”). For Q and R , there may be many queries consistent with the labeled data, so the median F_1 score is reported. Bold indicates best score at a given number of steps.

ID	0 Steps				5 Steps				10 Steps				25 Steps			
	Q	R	L	T	Q	R	L	T	Q	R	L	T	Q	R	L	T
A	0.69	0.69	1.00	0.74	1.00	1.00	1.00	0.74	1.00	1.00	1.00	0.74	1.00	1.00	1.00	0.74
B	0.99	0.99	0.47	0.20	0.99	0.99	0.47	0.25	0.99	0.99	0.47	0.05	0.99	0.98	0.47	0.06
C	0.96	0.96	0.38	0.09	0.99	0.96	0.38	0.08	0.99	0.96	0.38	0.02	0.99	0.98	0.38	0.01
D	0.77	0.77	0.52	0.27	0.99	0.96	0.52	0.28	0.99	0.99	0.52	0.32	0.99	1.00	0.52	0.08
E	1.00	1.00	0.44	0.29	1.00	1.00	0.44	0.14	1.00	1.00	0.44	0.13	1.00	1.00	0.44	0.07
F	0.88	0.88	0.78	0.38	0.99	0.96	0.78	0.39	1.00	0.96	0.78	0.18	1.00	0.96	0.78	0.27
G	0.68	0.68	0.65	0.78	1.00	1.00	0.65	0.77	1.00	1.00	0.65	0.77	1.00	1.00	0.65	0.77
H	0.30	0.30	0.12	0.22	0.34	0.34	0.13	0.23	0.92	0.92	0.13	0.22	0.92	0.92	0.13	0.37
I	0.37	0.37	0.13	0.00	1.00	0.37	0.13	0.00	1.00	1.00	0.13	0.00	1.00	1.00	0.13	0.31
J	0.07	0.07	0.01	1.00	0.41	0.07	0.01	0.86	0.80	0.09	0.04	0.75	0.80	0.09	0.04	0.86
K	0.28	0.28	0.15	0.00	0.99	0.27	0.15	0.00	0.99	0.99	0.15	0.00	0.99	0.99	0.15	0.00
L	0.67	0.67	0.07	0.37	0.96	0.88	0.07	0.42	0.96	0.88	0.07	0.08	0.96	0.88	0.07	0.31
M	0.92	0.92	0.10	0.37	0.99	0.92	0.10	0.46	0.99	0.92	0.10	0.00	0.99	0.92	0.10	0.18
N	0.60	0.60	0.02	0.00	0.20	0.09	0.02	0.00	0.11	0.21	0.02	0.00	0.18	0.78	0.02	0.31
O	0.11	0.11	0.01	0.04	0.50	0.17	0.01	0.21	0.70	0.17	0.01	0.21	1.00	0.21	0.01	0.00
P	0.16	0.16	0.04	0.04	0.23	0.21	0.03	0.04	0.82	0.21	0.03	0.14	1.00	0.29	0.03	0.14
Q	0.07	0.07	0.02	0.02	0.16	0.12	0.01	0.31	0.92	0.12	0.01	0.18	1.00	0.12	0.01	0.20

For tractability, we limit search to sketches with at most three predicates, at most two of which may have parameters. In most cases, this excludes the ground truth from the search space.

5.2 Accuracy of Synthesized Queries

We show that QUIVR synthesizes accurate queries from just a few labeled examples. We evaluate the F_1 score of the synthesized queries on Z_{test} . Recall that our algorithm returns a list C of consistent queries; we report the median F_1 score across $Q \in C$.

Baselines. We compare to (i) an ablation where we replace our active learning strategy with an approach that labels z uniformly at random from the remaining unlabeled training examples; (ii) an LSTM [16, 33] neural network; and (iii) a transformer neural network [26, 29, 77]. Because neural networks perform poorly on such small datasets, we pretrain the LSTM on an auxiliary task, namely, trajectory forecasting [43]. Then, we freeze the hidden representation of the learned LSTM, and use these as features to train a logistic regression model on our labeled examples. The neural network baselines do active learning by selecting among the unlabeled trajectories the one with the highest predicted probability of being positive.

Results. We show the F_1 score of each of the 17 queries in Table 3 after 0, 5, 10, and 25 steps of active learning. After just 10 steps, our approach provides F_1 score above 0.99 on 10 of 17 queries, and after 25 steps, it yields an F_1 score

Table 4. Running time (seconds) of synthesis (mean \pm standard error) using binary search (B) and quantitative semantics (Q) running on CPU and GPU, with 25 steps of active learning.

ID	CPU		GPU	
	B	Q	B	Q
A	8,460 \pm 1,517	3,343 \pm 202	737 \pm 36	174 \pm 14
B	3,511 \pm 549	2,291 \pm 237	428 \pm 37	110 \pm 9
C	3,319 \pm 505	2,007 \pm 359	376 \pm 6	113 \pm 9
D	2,728 \pm 476	2,714 \pm 334	370 \pm 8	119 \pm 2
E	1,264 \pm 176	599 \pm 54	225 \pm 3	50 \pm 1
F	1,689 \pm 360	748 \pm 81	285 \pm 7	60 \pm 1
G	661 \pm 141	133 \pm 23	219 \pm 77	30 \pm 1
H	399 \pm 70	147 \pm 9	185 \pm 94	32 \pm 17
I	400 \pm 74	84 \pm 13	163 \pm 85	23 \pm 12
J	544 \pm 120	173 \pm 5	227 \pm 121	36 \pm 19
K	493 \pm 77	125 \pm 25	163 \pm 83	30 \pm 16
L	732 \pm 47	286 \pm 73	252 \pm 133	57 \pm 29
M	697 \pm 40	253 \pm 49	245 \pm 128	56 \pm 30
N	5,691 \pm 272	977 \pm 176	1,393 \pm 590	264 \pm 136
O	8,306 \pm 521	2,314 \pm 476	811 \pm 12	127 \pm 2
P	11,326 \pm 673	4,198 \pm 1,333	970 \pm 60	167 \pm 8
Q	12,430 \pm 962	2,915 \pm 508	1,141 \pm 101	183 \pm 11

above 0.9 on all but 2 queries. Thus, QUIVR is able to synthesize accurate queries with relatively little user input. The neural networks achieve poor performance, particularly on the more difficult queries.

5.3 Synthesis Running Time

Next, we show that quantitative pruning and using a GPU each significantly reduce synthesis time, evaluating total running time for 25 steps of active learning.

Ablations. We compare to two ablations: (i) using the binary search approach of [53] to find pruning pairs, rather than using our quantitative semantics, and (ii) evaluating the matrix semantics (Appendix A.1) on a CPU rather than a GPU.

Results. In Fig. 4, we report the running time of our algorithms on a CPU (2 \times AMD EPYC 7402 24-Core) and a GPU (1 \times NVIDIA RTX A6000). For binary search, on average, the GPU is 7.6 \times faster than the CPU. On a GPU, using the quantitative semantics rather than binary search offers another 5.0 \times speed-up.

6 Related Work

Monotonicity for Parameter Pruning. We build on [49] for our parameter pruning algorithm. Their approach has been applied to synthesizing STL formulas for

sequence classification by first enumerating sketches and then using monotonicity to find parameters, similar to our binary search baseline [53]. We replace binary search with our novel strategy based on quantitative semantics, leading to $5.0\times$ speedup. There is also work building on [49] to create logically-relevant distance metrics between trajectories by taking the Hausdorff distance between parameter satisfaction regions (which they call “validity domains”), with applications to clustering [78]. For logics like STL, our quantitative semantics could provide a speedup to their approach.

Synthesis of Temporal Logic Formulas. More broadly, there has been work synthesizing parameters in a variant of STL by discretizing the parameter space and then walking the satisfaction boundary [24]; in one dimension, their approach becomes binary search, inheriting its shortcomings. There has been work on synthesizing STL formulas that are satisfied by a closed-loop control model [38], but they assume the ability to find counterexample traces for incorrect STL formulas, which is not applicable to our setting. Another approach is to synthesize parameters in STL formulas using gradient-based optimization [35] or stochastic optimization [45], but we found these methods to be ineffective in our setting, and they do not come with either soundness or completeness guarantees. There is work using decision trees to synthesize STL formulas [1, 14, 44, 48], but these operate on a restricted subset of STL, namely Boolean combinations of a fixed set of template formulas. This restriction prevents these approaches from synthesizing temporal structure, which is a key component of the queries in our domains. Finally, there has been work on active learning of STL formulae using decision trees [48], but it assumes the ability to query for equivalence between a particular STL formula and the ground truth, which is not possible in our setting.

Synthesizing Constants. There is work on synthesizing parameters of programs using counterexample-guided inductive synthesis and different theory solvers, including Fourier-Motzkin variable elimination and an SMT solver [2]. Though our synthesis objective can be encoded in the theory of linear arithmetic, it is extremely large, and we have found such solvers to be ineffective in practice.

Querying Video Data. There has been recent work on querying object detections and trajectories in video data [5, 7, 8, 28, 40–42, 54]. The main difference is our focus on synthesis; in addition, these approaches focus on SQL-like operators such as select, inner-join, group-by, etc., over predefined predicates, which cannot capture compositions such as the sequencing and iteration operators in our language, which are necessary for identifying more complex behaviors.

Neurosymbolic Models. There has been recent work on leveraging program synthesis in the context of machine learning. For instance, there has been work on using programs to represent high-level structure in images [21–23, 74, 84], for reinforcement learning [9, 34, 79, 80], and for querying websites [18]; in contrast, we use programs to classify trajectories. The most closely related work is on synthesizing functional programs operating over lists [67, 76]. Our language includes key constructs not included in their languages. Most importantly, we

include sequencing; in their functional language, such an operator would need to be represented as a nested series of if-then-else operators. In addition, their language does not support predicates that match subsequences; while such a predicate could be added, none of their operators can compose such predicates.

Quantitative Synthesis. There has been work on program synthesis with quantitative properties—e.g., on synthesis for producing optimized code [37, 57, 65], for approximate computing [15, 52], for probabilistic programming [56], and for embedded control [17]. These approaches largely focus on search-based synthesis, either using constraint optimization [52], continuous optimization [17], enumerative search [15, 57], or stochastic search [37, 56, 65]. While we leverage ideas from this literature, our quantitative semantics based pruning strategy is novel.

Quantitative Semantics. Our quantitative semantics is similar to the “robustness degree” [25] of a temporal logic formula. The difference is that, by adjusting the denotations of the base predicates, our quantitative semantics gives a parameter on the satisfaction boundary. More broadly, there has been work on quantitative semantics for temporal logic for robust constraint satisfaction [20, 25, 73], and to guide reinforcement learning [39]. There has been work on quantitative regular expressions (QREs) [4], though in general, QREs cannot be efficiently evaluated due to their nondeterminism, and our language is restricted to ensure efficient computation. There has been work on synthesizing QREs for network traffic classification [68], using binary search to compute decision thresholds. Similarly, there has been work using the Viterbi semiring to obtain quantitative semantics for Datalog programs [69], which they use in conjunction with gradient descent to learn the rules of the Datalog program. In contrast, we use our quantitative semantics to efficiently prune the parameter search space in a provably correct way. Finally, there has been work on using GPUs to evaluate regular expressions [55]; however, they focus on regular expressions over strings.

Query Languages. Our language is closely related to both signal temporal logic (STL) [50] and Kleene algebras with tests (KAT) [46]. In particular, it can straightforwardly be extended to subsume both (see Appendix A for details), and our pruning strategy applies to this extended language. The addition of Kleene star, required to subsume KAT, worsens the evaluation time. STL has been used to monitor safety requirements for autonomous vehicles [32]. Spatio-Temporal Perception Logic (SPTL) is an extension of STL to support spatial reasoning [31]. Many of its operators are monotone, and thus would benefit from our algorithm. Scenic [27] is a DSL for creating static and dynamic driving scenes, but its focus is on generating scenes rather than querying for behaviors.

7 Conclusion

We have proposed a novel framework called QUIVR for synthesizing queries over video trajectory data. Our language is similar to KAT and STL, but supports conjunction and sequencing over multi-step predicates. Given only a few examples, QUIVR efficiently synthesizes trajectory queries consistent with those examples. A key contribution of our approach is the use of a quantitative semantics to

prune the parameter search space, yielding a $5.0\times$ speedup over the state-of-the-art. In our evaluation, we demonstrate that QUIVR effectively synthesizes queries to identify interesting driving behaviors, and that our optimizations dramatically reduce synthesis time.

Acknowledgements. We thank the anonymous reviewers for their helpful feedback. This work was supported in part by NSF Award CCF-1910769, NSF Award CCF-1917852, and ARO Award W911NF-20-1-0080.

References

1. Aasi, E., Vasile, C.I., Bahreinian, M., Belta, C.: Classification of time-series data using boosted decision trees. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1263–1268 (2022). <https://doi.org/10.1109/IROS47612.2022.9982105>
2. Abate, A., David, C., Kesseli, P., Kroening, D., Polgreen, E.: Counterexample guided inductive synthesis modulo theories. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 270–288. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_15
3. Alur, R., et al.: Syntax-guided synthesis. IEEE (2013)
4. Alur, R., Mamouras, K., Ulus, D.: Derivatives of quantitative regular expressions. In: Aceto, L., Bacci, G., Bacci, G., Ingólfssdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools. LNCS, vol. 10460, pp. 75–95. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63121-9_4
5. Bastani, F., et al.: SkyQuery: optimizing video queries over UAVs (2019)
6. Bastani, F., et al.: SkyQuery: optimizing video queries over UAVs. In: Onward! (2021)
7. Bastani, F., et al.: MIRIS: fast object track queries in video. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 1907–1921 (2020)
8. Bastani, F., Moll, O., Madden, S.: Vaas: video analytics at scale. Proc. VLDB Endow. **13**(12), 2877–2880 (2020)
9. Bastani, O., Pu, Y., Solar-Lezama, A.: Verifiable reinforcement learning via policy extraction. In: Advances in Neural Information Processing Systems, pp. 2494–2504 (2018)
10. Bergmann, P., Meinhardt, T., Leal-Taixe, L.: Tracking without bells and whistles. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 941–951 (2019)
11. Bertinetto, L., Valmadre, J., Henriques, J.F., Vedaldi, A., Torr, P.H.S.: Fully-convolutional Siamese networks for object tracking. In: Hua, G., Jégou, H. (eds.) ECCV 2016. LNCS, vol. 9914, pp. 850–865. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48881-3_56
12. Betke, M., Hirsh, D.E., Bagchi, A., Hristov, N.I., Makris, N.C., Kunz, T.H.: Tracking large variable numbers of objects in clutter. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8. IEEE (2007)
13. Bock, J., Krajewski, R., Moers, T., Runde, S., Vater, L., Eckstein, L.: The inD dataset: a drone dataset of naturalistic road user trajectories at German intersections. arXiv preprint [arXiv:1911.07602](https://arxiv.org/abs/1911.07602) (2019)

14. Bombara, G., Belta, C.: Offline and online learning of signal temporal logic formulae using decision trees. *ACM Trans. Cyber-Phys. Syst.* **5**(3) (2021). <https://doi.org/10.1145/3433994>
15. Bornholt, J., Torlak, E., Grossman, D., Ceze, L.: Optimizing synthesis with metas-ketches. In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 775–788 (2016)
16. Chang, M.F., et al.: Argoverse: 3D tracking and forecasting with rich maps. In: *Conference on Computer Vision and Pattern Recognition (CVPR)* (2019)
17. Chaudhuri, S., Clochard, M., Solar-Lezama, A.: Bridging Boolean and quantitative synthesis using smoothed proof search. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 207–220 (2014)
18. Chen, Q., Lamoreaux, A., Wang, X., Durrett, G., Bastani, O., Dillig, I.: Web question answering with neurosymbolic program synthesis. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 328–343 (2021)
19. Dasgupta, S.: Analysis of a greedy active learning strategy. In: *Advances in Neural Information Processing Systems*, pp. 337–344 (2005)
20. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. *Formal Meth. Syst. Des.* **51**(1), 5–30 (2017). <https://doi.org/10.1007/s10703-017-0286-7>
21. Ellis, K., Nye, M., Pu, Y., Sosa, F., Tenenbaum, J., Solar-Lezama, A.: Write, execute, assess: program synthesis with a repl. In: *Advances in Neural Information Processing Systems*, pp. 9169–9178 (2019)
22. Ellis, K., Ritchie, D., Solar-Lezama, A., Tenenbaum, J.: Learning to infer graphics programs from hand-drawn images. In: *Advances in Neural Information Processing Systems*, pp. 6059–6068 (2018)
23. Ellis, K., Solar-Lezama, A., Tenenbaum, J.: Unsupervised learning by program synthesis. In: *Advances in Neural Information Processing Systems*, pp. 973–981 (2015)
24. Ergurtuna, M., Gol, E.A.: An efficient formula synthesis method with past signal temporal logic. *IFAC-PapersOnLine* **52**(11), 43–48 (2019). <https://doi.org/10.1016/j.ifacol.2019.09.116>. <https://www.sciencedirect.com/science/article/pii/S2405896319307451>
25. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. *Theoret. Comput. Sci.* **410**(42), 4262–4291 (2009)
26. Franco, L., Placidi, L., Giuliari, F., Hasan, I., Cristani, M., Galasso, F.: Under the hood of transformer networks for trajectory forecasting. *Pattern Recogn.* **138**, 109372 (2023). <https://doi.org/10.1016/j.patcog.2023.109372>. <https://www.sciencedirect.com/science/article/pii/S0031320323000730>
27. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 63–78 (2019)
28. Fu, D.Y., et al.: Rekall: specifying video events using compositions of spatiotemporal labels. *arXiv preprint arXiv:1910.02993* (2019)
29. Giuliari, F., Hasan, I., Cristani, M., Galasso, F.: Transformer networks for trajectory forecasting (2020)
30. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778 (2016)

31. Hekmatnejad, M., Hoxha, B., Deshmukh, J.V., Yang, Y., Fainekos, G.: Formalizing and evaluating requirements of perception systems for automated vehicles using spatio-temporal perception logic (2022)
32. Hekmatnejad, M., et al.: Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic. In: Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2019. Association for Computing Machinery, New York (2019). <https://doi.org/10.1145/3359986.3361203>
33. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
34. Inala, J.P., Bastani, O., Tavares, Z., Solar-Lezama, A.: Synthesizing programmatic policies that inductively generalize. In: International Conference on Learning Representations (2019)
35. Jha, S., Tiwari, A., Seshia, S.A., Sahai, T., Shankar, N.: TeLEx: learning signal temporal logic from positive examples using tightness metric. *Formal Meth. Syst. Des.* **54**(3), 364–387 (2019). <https://doi.org/10.1007/s10703-019-00332-1>
36. Ji, R., Liang, J., Xiong, Y., Zhang, L., Hu, Z.: Question selection for interactive program synthesis. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 1143–1158 (2020)
37. Jia, Z., Thomas, J., Warszawski, T., Gao, M., Zaharia, M., Aiken, A.: Optimizing DNN computation with relaxed graph substitutions. In: Proceedings of the 2nd Conference on Systems and Machine Learning, SysML 2019 (2019)
38. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* **34**(11), 1704–1717 (2015). <https://doi.org/10.1109/TCAD.2015.2421907>
39. Jothimurugan, K., Alur, R., Bastani, O.: A composable specification language for reinforcement learning tasks. In: Advances in Neural Information Processing Systems, pp. 13041–13051 (2019)
40. Kang, D., Bailis, P., Zaharia, M.: Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proc. VLDB Endow.* **13**(4), 533–546 (2019)
41. Kang, D., Mathur, A., Veeramacheneni, T., Bailis, P., Zaharia, M.: Jointly optimizing preprocessing and inference for DNN-based visual analytics. *arXiv preprint arXiv:2007.13005* (2020)
42. Kang, D., Raghavan, D., Bailis, P., Zaharia, M.: Model assertions for monitoring and improving ml model. *arXiv preprint arXiv:2003.01668* (2020)
43. Kitani, K.M., Ziebart, B.D., Bagnell, J.A., Hebert, M.: Activity forecasting. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012. LNCS, vol. 7575, pp. 201–214. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33765-9_15
44. Kong, Z., Jones, A., Belta, C.: Temporal logics for learning and detection of anomalous behavior. *IEEE Trans. Autom. Control* **62**(3), 1210–1222 (2017). <https://doi.org/10.1109/TAC.2016.2585083>
45. Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., Belta, C.: Temporal logic inference for classification and prediction from data. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, HSCC 2014, pp. 273–282. Association for Computing Machinery, New York (2014). <https://doi.org/10.1145/2562059.2562146>
46. Kozen, D.: Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.* (TOPLAS) **19**(3), 427–443 (1997)

47. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. *Commun. ACM* **60**(6), 84–90 (2017)
48. Linard, A., Tumova, J.: Active learning of signal temporal logic specifications. In: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), pp. 779–785 (2020). <https://doi.org/10.1109/CASE48305.2020.9216778>
49. Maler, O.: Learning monotone partitions of partially-ordered domains (Work in Progress), July 2017. Working paper or preprint. <https://hal.science/hal-01556243>
50. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT - 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
51. Mell, S., Bastani, F., Zdancewic, S., Bastani, O.: Synthesizing trajectory queries from examples. Technical report, MS-CIS-23-02, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, July 2023
52. Misailovic, S., Carbin, M., Achour, S., Qi, Z., Rinard, M.C.: Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. *ACM SIGPLAN Not.* **49**(10), 309–328 (2014)
53. Mohammadinejad, S., Deshmukh, J.V., Puranic, A.G., Vazquez-Chanlatte, M., Donzé, A.: Interpretable classification of time-series data using efficient enumerative techniques. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control, HSCC 2020, Association for Computing Machinery, New York (2020). <https://doi.org/10.1145/3365365.3382218>
54. Moll, O., Bastani, F., Madden, S., Stonebraker, M., Gadepally, V., Kraska, T.: ExSample: efficient searches on video repositories through adaptive sampling. *arXiv preprint arXiv:2005.09141* (2020)
55. Naghmouchi, J., Scarpazza, D.P., Berekovic, M.: Small-ruleset regular expression matching on GPGPUs: quantitative performance analysis and optimization. In: Proceedings of the 24th ACM International Conference on Supercomputing, pp. 337–348 (2010)
56. Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Efficient synthesis of probabilistic programs. *ACM SIGPLAN Not.* **50**(6), 208–217 (2015)
57. Phothilimthana, P.M., Thakur, A., Bodik, R., Dhurjati, D.: Scaling up superoptimization. In: Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 297–310 (2016)
58. Preiss, J.A., Honig, W., Sukhatme, G.S., Ayanian, N.: Crazyswarm: a large nanorobot swarm. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 3299–3304. IEEE (2017)
59. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. *arXiv preprint arXiv:1804.02767* (2018)
60. Ren, S., He, K., Girshick, R., Sun, J.: Fast R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99 (2015)
61. Robicquet, A., Sadeghian, A., Alahi, A., Savarese, S.: Learning social etiquette: human trajectory understanding in crowded scenes. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9912, pp. 549–565. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46484-8_33
62. Roy, N., McCallum, A.: Toward optimal active learning through sampling estimation of error reduction. In: International Conference on Machine Learning (2001)

63. Sadigh, D., Sastry, S.S., Seshia, S.A., Dragan, A.: Information gathering actions over human internal state. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 66–73. IEEE (2016)
64. Sadigh, D., Sastry, S., Seshia, S.A., Dragan, A.D.: Planning for autonomous cars that leverage effects on human actions. In: Robotics: Science and Systems, vol. 2. Ann Arbor, MI, USA (2016)
65. Schkufza, E., Sharma, R., Aiken, A.: Stochastic superoptimization. ACM SIGARCH Comput. Arch. News **41**(1), 305–316 (2013)
66. Schmerling, E., Leung, K., Vollprecht, W., Pavone, M.: Multimodal probabilistic model-based planning for human-robot interaction. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 1–9. IEEE (2018)
67. Shah, A., Zhan, E., Sun, J., Verma, A., Yue, Y., Chaudhuri, S.: Learning differentiable programs with admissible neural heuristics. In: Advances in Neural Information Processing Systems, vol. 33 (2020)
68. Shi, L., Li, Y., Loo, B.T., Alur, R.: Network traffic classification by program synthesis. In: TACAS 2021. LNCS, vol. 12651, pp. 430–448. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_23
69. Si, X., Raghothaman, M., Heo, K., Naik, M.: Synthesizing datalog programs using numerical relaxation. In: IJCAI (2019)
70. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. *nature* **529**(7587), 484–489 (2016)
71. Solar-Lezama, A.: Program Synthesis by Sketching. University of California, Berkeley (2008)
72. Sun, J.J., et al.: The MABe22 benchmarks for representation learning of multi-agent behavior. June 2022. <https://doi.org/10.22002/D1.20186>
73. Tabuada, P., Neider, D.: Robust linear temporal logic. In: Computer Science Logic 2016 (2016)
74. Tian, Y., et al.: Learning to infer and execute 3D shape programs. arXiv preprint [arXiv:1901.02875](https://arxiv.org/abs/1901.02875) (2019)
75. Tweed, D., Calway, A.: Tracking multiple animals in wildlife footage. In: Object Recognition Supported by User Interaction for Service Robots, vol. 2, pp. 24–27. IEEE (2002)
76. Valkov, L., Chaudhari, D., Srivastava, A., Sutton, C., Chaudhuri, S.: HOUDINI: lifelong learning as program synthesis. In: Advances in Neural Information Processing Systems, pp. 8687–8698 (2018)
77. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., Luxburg, U.V., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017). https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
78. Vazquez-Chanlatte, M., Ghosh, S., Deshmukh, J.V., Sangiovanni-Vincentelli, A., Seshia, S.A.: Time-series learning using monotonic logical properties. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 389–405. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_22
79. Verma, A., Le, H., Yue, Y., Chaudhuri, S.: Imitation-projected programmatic reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 15752–15763 (2019)
80. Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S.: Programmatically interpretable reinforcement learning. In: ICML (2018)
81. Weinstein, A., Cho, A., Loianno, G., Kumar, V.: Visual inertial odometry swarm: an autonomous swarm of vision-based quadrotors. *IEEE Robot. Autom. Lett.* **3**(3), 1801–1807 (2018)

82. Wishart, J., et al.: Driving safety performance assessment metrics for ads-equipped vehicles. SAE Technical Paper 2(2020-01-1206) (2020)
83. Wojke, N., Bewley, A., Paulus, D.: Simple online and realtime tracking with a deep association metric. In: 2017 IEEE International Conference on Image Processing (ICIP), pp. 3645–3649. IEEE (2017)
84. Young, H., Bastani, O., Naik, M.: Learning neurosymbolic generative models via program synthesis. In: ICML (2019)
85. Zhan, E., Tseng, A., Yue, Y., Swaminathan, A., Hausknecht, M.: Learning calibratable policies using programmatic style-consistency. In: ICML (2020)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

