

# Chapter 1

## Hybrid Property, Event and Relation Learner

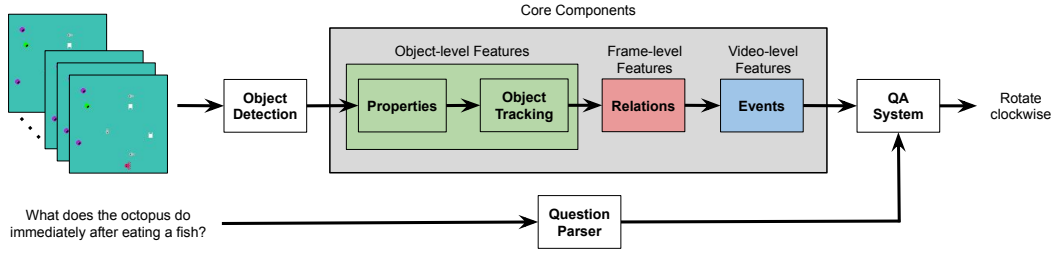
We outline a novel paradigm for solving the VideoQA problem. This structure merges deep learning and logic-based machine learning and inference methods in an attempt to learn the concepts required to answer the questions. We name this structure: Hybrid Property, Event and Relation Learner (H-PERL). Section 1.1 outlines the structure of an H-PERL model, while Section 1.2 discusses the implementation of a number of H-PERL components which are common to all models shown in this report.

### 1.1 H-PERL Model

H-PERL is a generic, pipelined structure for finding an answer to a set of questions, given a video. The pipeline is composed of a number of components which, when strung together, form a model (a model can be thought of as a specific ‘instance’ of the H-PERL pipeline). Chapters ?? and ?? each outline an H-PERL model for the OceanQA dataset.

#### 1.1.1 Architecture

An H-PERL pipeline assumes that all of the information in an environment which is required to answer the questions can be modeled using: objects; binary relations between objects; and events between two consecutive frames in the video. For many environments, simple environments (like our OceanQA dataset) in particular, this assumption holds. However for many other VideoQA environments, particularly those set in the real-world, this assumption may not be suitable. For example, we may not always be able to extract objects from a video (as in the case of abstract nouns), and yet information on these objects may still be required to answer the questions. Additionally, the H-PERL structure is not capable of modelling relations between objects with an arity larger than 2.



**Figure 1.1:** An H-PERL pipeline for VideoQA. Green, red and blue shading indicates components which work to extract features at different levels of abstraction. Grey shading indicates the ‘core’ components of the pipeline.

Figure 1.1 shows the components involved in a typical H-PERL pipeline. During evaluation, information from the video and the question flow, from left to right, through the pipeline. Each H-PERL model assumes that the object detection, question parsing and QA system components are “pre-made” (either pre-trained or manually engineered). We refer to these as ‘non-core’ components. H-PERL allows the remaining, ‘core’ components to be updated as the model is trained, although they don’t necessarily have to be. Each core component in the pipeline accumulates information. This means that each component guarantees to add more features to the data, rather than overwrite existing features (with the small exception of the event component when error correction is used, discussed further in Chapter ??). As shown in Figure 1.1 components in the pipeline work at different levels of abstraction; the object properties and tracking components work to extract object-level features, while the relations and events components work to extract frame and video-level features, respectively.

The following is a high-level description of the tasks each component is required to complete for the H-PERL pipeline to work with high accuracy:

1. **Question Parser.** The QA parsing component is used to extract relevant pieces of information from the questions (and answers when training). For example, given the question: “What does the octopus do immediately after eating a fish?” and that the question is a state-transition question, the parsing component would extract, firstly, that the object in question was an octopus, and secondly, that the event was ‘eat a fish’. The parsing component, therefore, bridges the gap between the symbolic data, which the model works with, and the natural language questions and answers. When an H-PERL model is being evaluated, only the question needs to be parsed. However, when the model is training this component acts as a question-and-answer parser, since the model requires that the feedback that comes from the answer is also in symbolic form.
2. **Object Detector.** The detection component produces bounding boxes and classes for each object in each frame of the video. Any object not detected

at this stage of the pipeline is assumed to part of the background and is therefore ignored by the rest of the model.

3. **Property Extractor.** Given a set of images of objects from the detection component, the property extraction component assigns a value to every property listed in the environment specification for every object in the set.
4. **Object Tracker.** The object tracker is required to assign an identifier to each object in a given video. The object identifiers assigned in the initial frame of the video can be arbitrary, but then an algorithm is usually applied inductively to each remaining frame of the video in an attempt to assign each object the same identifier as it was given in the previous frame.
5. **Relation Classifier.** The job of the binary relation classifier is, given a symbolic representation of a video, to list all of the instances of binary relations between objects in the video. The set of possible binary relation is defined in the dataset's environment specification.
6. **Event Detector.** The event detection component produces a set of events for each pair of consecutive frames in the video. Each set of events can contain both actions and effects, and each event consists of an event name and an object identifier which signifies the non-static object that took part in that event.
7. **QA System.** The job of the QA system is to take all of the features which have been accumulated by previous components in the pipeline, along with a parsed question, and produce an answer to the question. The QA system is also given the question type, this allows it to make sense of the parsed question and to apply different reasoning for each question type.

### 1.1.2 Information Representation

The first component in the H-PERL pipeline, the object detector, takes a raw video as input and produces a set of bounding boxes (object positions) and object classes for each frame of the video. After object detection has been applied data about the video is stored symbolically for each object in each frame of the video. Each of the following components in the pipeline can access this symbolic data along with the raw image of each object (the raw video frames are discarded). Each of these components then accumulates symbolic information about the objects, frames or video. This is what is meant by extracting object, frame and video-level features.

The pipeline needs an agreed upon way of representing the extracted information. This representation is as follows:

- For an object with identifier  $\langle id \rangle$  in frame  $\langle frame \rangle$ , the object's properties, rotation and class (all referred to as  $\langle property \rangle$ ), each with value  $\langle value \rangle$ , are represented as follows:

$$\text{obs}(\langle property \rangle(\langle value \rangle, \langle id \rangle), \langle frame \rangle) \quad (1.1)$$

As described in Chapter ??, the value of an object's rotation is given as  $(x1, y1, x2, y2)$ , where  $(x1, y1)$  is the top left corner of the object, and  $(x2, y2)$  is the bottom right.

- For two objects in frame  $\langle frame \rangle$ , with identifiers  $\langle id1 \rangle$  and  $\langle id2 \rangle$ , a binary relation,  $\langle relation \rangle$ , between the objects is represented as follows:

$$\text{obs}(\langle relation \rangle(\langle id1 \rangle, \langle id2 \rangle), \langle frame \rangle) \quad (1.2)$$

### 1.1.3 Requirements

To give a complete view of what is needed to construct an H-PERL model, we outline the minimum set of requirements and a number of assumptions. Each instance of an H-PERL model may require additional constraints to be applied on top of these. The set of requirements is as follows:

1. A set of pre-made, non-core components (question parser, object detector and QA system).
2. A VideoQA dataset, where each element is of the form:

$$\langle \text{video}, \{ \langle \text{question}, \text{answer}, \text{question type} \rangle \} \rangle$$

3. Environment specification for the given dataset.
4. (Optionally) Background knowledge of the environment written in ASP.

H-PERL also requires that the following assumptions be made about the data:

1. All relevant information in the video can be modelled by object properties; binary relations between objects; and events occurring between consecutive frames of the video.
2. Each of the properties, relations and events components can be trained individually and directly using a specific type of question. For example, for the OceanQA dataset, we can use question types 1, 2 and 3 to train the properties, relations and events components, respectively.
3. As mentioned in Chapter ??, properties, relations and events must be discrete. There is also currently no way of modelling continuous variables in the data, for example, we cannot say that the octopus rotated clockwise by  $32^\circ$ .

These requirements and assumptions clarify some of the limitations of H-PERL models. Firstly, for some environments pre-trained object detectors (or data to train them with) may be difficult to find. Secondly, it may also be difficult, if not impossible, to construct or train a question parser for free-form, natural language

question-answering datasets. Additionally, most QA datasets are not guaranteed to contain questions which can be used to directly train components of the model. Finally, many environments will simply be too complex to be accurately modelled by discrete properties, relations and events. These requirements and assumptions do, however, provide initial directions for future research in the area of Hybrid question-answering models. We discuss some potential extensions to H-PERL in Chapter ??.

As well as drawbacks, the H-PERL architecture does also provide a number of advantages. One of the most important advantages of hybrid models is the ability to encode common-sense knowledge or background knowledge of the environment directly into the model without having to learn it. Since the model accumulates video information symbolically, we can inject background knowledge at any stage of the pipeline. A number of further advantages will present themselves in the following chapters, and these are discussed in more detail in Chapter ??.

## 1.2 Common Components

Give an intro to each of the components common to all models: object detector, tracker, (QA system), QA parser.