Pham, Truong An; Wang, Junjue; Iyengar, Roger; Xiao, Yu; Pillai, Padmanabhan; Klatzky, Roberta; Satyanarayanan, Mahadev

Ajalon

Published in:
Software - Practice and Experience

DOI:
10.1002/spe.2987

Published: 01/08/2021

Document Version
Publisher's PDF, also known as Version of record

Received: 10 October 2020 | Revised: 25 March 2021 | Accepted: 23 April 2021

DOI: 10.1002/spe.2987

RESEARCH ARTICLE

WILEY

Ajalon: Simplifying the authoring of wearable cognitive assistants

Truong An Pham[1] | Junjue Wang[2] | Roger Iyengar[2] | Yu Xiao[1] | Padmanabhan Pillai[3] | Roberta Klatzky[2] | Mahadev Satyanarayanan[2]

[1]School of Electrical Engineering, Aalto University, Espoo, Finland

[2]Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

[3]Intel Labs, Pittsburgh, Pennsylvania, USA

Correspondence
Truong An Pham, Department of Communications and Networking, School of Electrical Engineering, Aalto University, Konemiehentie 2, 02150 Espoo.
Email: truong.pham@aalto.fi

Funding information
Business Finland, Grant/Award Number: 1660/31/2018; Division of Computer and Network Systems, Grant/Award Numbers: CNS-1518865, DGE1252522, DGE1745016

Summary

Wearable Cognitive Assistance (WCA) amplifies human cognition in real time through a wearable device and low-latency wireless access to edge computing infrastructure. It is inspired by, and broadens, the metaphor of GPS navigation tools that provide real-time step-by-step guidance, with prompt error detection and correction. WCA applications are likely to be transformative in education, health care, industrial troubleshooting, manufacturing, assisted driving, and sports training. Today, WCA application development is difficult and slow, requiring skills in areas such as machine learning and computer vision that are not widespread among software developers. This paper describes *Ajalon,* an authoring toolchain for WCA applications that reduces the skill and effort needed at each step of the development pipeline. Our evaluation shows that Ajalon significantly reduces the effort needed to create new WCA applications.

KEYWORDS

artificial intelligence, augmented reality, cloudlets, computer vision, edge computing, Gabriel, machine learning, mobile computing, software productivity, wearables

1 | INTRODUCTION

1.1 | Motivation

Since its introduction in 2014,[1] *Wearable Cognitive Assistance* (WCA) has attracted considerable attention. A WCA application provides just-in-time guidance and error detection for a user who is performing an unfamiliar task. Prompt error detection is also valuable for a user who is performing familiar tasks, since human errors cannot be completely avoided, especially when the user is tired or stressed. Informally, WCA is like having "an angel on your shoulder."[2] It is inspired by, and broadens, the metaphor of GPS navigation tools that provide real-time step-by-step guidance, with prompt error detection and correction. Table 1 summarizes the attributes of a small sample of the many WCA applications that we have built since 2014. A YouTube video of a WCA application to assemble an IKEA kit can be viewed at https://shorturl.at/iEOZ0.

Abbreviation: WCA, wearable cognitive assistance

TABLE 1 Example wearable cognitive assistance applications (Source: Adapted from Satyanarayanan[4])

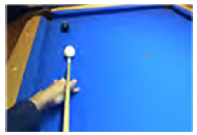
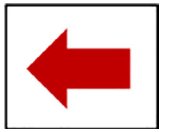
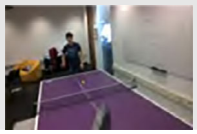
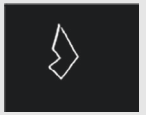
App name	Example input video frame	Description	Symbolic Representation	Example Guidance
Pool		Helps a novice pool player aim correctly. Gives continuous visual feedback (left arrow, right arrow, or thumbs up) as the user turns his cue stick. The symbolic representation describes the positions of the balls, target pocket, and the top and bottom of cue stick.	<Pocket, object ball, cue ball, cue top, cue bottom>	
Ping-pong		Tells novice to hit ball to the left or right, depending on which is more likely to beat opponent. Uses color, line, and optical-flow-based motion detection to detect ball, table, and opponent. Video URL: https://youtu.be/_lp32sowyUA	<InRally, ball position, opponent position>	Whispers "Left!"
Work-out		Counts out repetitions in physical exercises. Classification is done using Volumetric Template Matching on a 10-15 frame video segment. A poorly performed repetition is classified as a distinct type of exercise (e.g., "good pushup" versus "bad pushup").	<Action, count>	Says "8 "
Face		Jogs your memory on a familiar face whose name you cannot recall. Detects and extracts a tightly-cropped image of each face, and then applies a state-of-art face recognizer. Whispers the name of the person recognized.	ASCII text of name	Whispers "Barack Obama"
Lego		Guides a user in assembling two-dimensional Lego models. The symbolic representation is a matrix representing color for each brick. Video URL: https://youtu.be/7L9U-n29abg	[[0, 2, 1, 1], [0, 2, 1, 6], [2, 2, 2, 2]]	 Says "Put a 1x3 green piece on top"
Draw		Helps a user to sketch better. Builds on third-party app for desktops. Our implementation preserves the back-end logic. A Glass-based front-end allows a user to use any drawing surface and instrument. Displays the error alignment in sketch on Glass. Video URL: https://youtu.be/nuQpPtVJC6o		
Sandwich		Helps a cooking novice prepare sandwiches according to a recipe. Since real food is perishable, we use a food toy with plastic ingredients. Object detection uses Faster-RCNN deep neural net approach.[3] Video URL: https://youtu.be/USakPP45WvM	Object: "E.g. Lettuce on top of ham and bread"	 Says "Put a piece of bread on the lettuce"

TABLE 2 CPU load, latency bounds, and the required bandwidth of example wearable cognitive assistance (WCA) applications. The implementations of the application servers[7] were tested on a laptop (with an Intel® Core™ i7-8500Y processor and 8GB RAM), running the frontend on an Android phone, using 480p, 720p, and 1080p video resolutions. (*) End-to-end latency includes both the round trip time (RTT) from the Android phone to the cloudlet and compute time in the cloudlet. Tight and loose bounds are adopted from Chen et al.,[7] where they are defined: "The tight bound represents an ideal target, below which the user is insensitive to improvements. Above the loose bound, the user becomes aware of slowness, and user experience and performance is significantly impacted."

	Pool	Ping-pong	Face	Lego	Sandwich
CPU load(%)	72.10	45.40	75.60	52.20	85.10
End-to-end latency bounds (tight–loose, ms)*	95–105	150–230	370–1000	600–2700	
Video streaming bandwidth requirement	480p: 3.6 / 7.0				
(Average / Peak, Mbps)	720p: 6.8 / 9.9				
	1080p: 8.1 / 12.7				

A WCA application runs on a wearable device such as Google Glass or Microsoft Hololens, leaving the user's hands free for task performance. It provides visual and verbal guidance through the video and audio channels of the wearable device. It provides a user experience that is similar to the "look and feel" of augmented reality (AR) applications, while being deeply dependent on artificial intelligence (AI) algorithms such as object recognition via computer vision using deep neural networks (DNNs). Because of the computational limitations of lightweight wearable devices that have acceptable battery life, a WCA application uses a wireless network to offload its compute-intensive operations to a nearby cloudlet.[5] Table 2 lists the resource consumption and end-to-end latency bounds of five offloading-based WCA applications. It shows that WCA applications are simultaneously compute-intensive, bandwidth-hungry, and latency-sensitive. They are hence perceived as a class of "killer apps" for *edge computing*.[4,6]

Based on our experience from authoring many WCA applications, this paper describes a toolchain that we have built to simplify their creation. WCA applications are inherently task-specific because they embody deep knowledge of the task being performed. The task-specific focus reduces the complexity of the WCA application, and prevents it from becoming "AI-Complete."[8] There is a narrow context within which sensor data is interpreted, progress to completion is defined, and ambiguities are resolved. For example, in the task of assembling a kit of parts, the computer vision module only needs to be able to accurately recognize the parts in the kit. Everything else can be ignored as irrelevant information. Because of the narrowing of context, the WCA application is able to resolve errors and provide detailed-enough instructions for a user to complete the task.

1.2 | Contributions

This paper focuses on a toolchain called *Ajalon* that simplifies the task-specific aspects of authoring a WCA application. Ajalon layers task-specific components on top of a task-independent runtime platform called *Gabriel* that we have released open source.[1,9] For ease of exposition, we will refer to WCA applications as "Gabriel applications" in the rest of this paper. Developing a Gabriel application without Ajalon often requires multiple person-months. This is clearly not scalable. Ajalon's goal is to simplify the process so that a small team (1–2 people) of a task expert and a developer without computer vision expertise can create an initial version of a Gabriel application within a few days. This is a productivity improvement of at least one order of magnitude. Refinement and tuning may take longer, but can be guided by early use of the application. We focus on vision-based Gabriel applications in this paper, but plan to extend Ajalon to multisensor applications.

As depicted in Figure 1, Ajalon has four stages:

- The first stage, *Preprocessing Tool (PT),* automatically extracts a workflow from videos that were created by task experts. Some of these videos may demonstrate common errors a novice typically makes. PT automatically segments an input video into working steps. At each step, a list of associated objects is discovered. The segmentation into working steps and the list of associated objects are the output of PT.
- In the second stage, the application developer refines the output of PT using *PTEditor*. This refinement is needed because workflow extraction by PT is imperfect even with state-of-the-art computer vision techniques. PTEditor

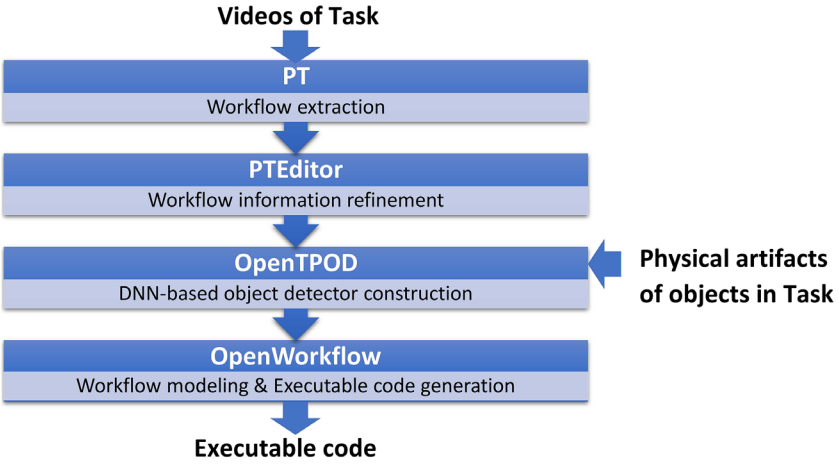


FIGURE 1 Overview of Ajalon's four stages [Colour figure can be viewed at wileyonlinelibrary.com]

provides merge and split functions for the task expert to modify the steps of the workflow. The task expert can also edit the list of associated objects discovered by PT.

- The third stage addresses the computer vision aspect of the toolchain. It consists of associating task-specific *object detectors* based on DNNs for each object in the task. In some cases, the necessary object detectors may already have been created and available through a library. This is likely to happen over time, as the use of WCA grows and object detectors become available for every manufactured component in a standard parts catalog. For nonstandard components and in the interim while WCA is still in its infancy, the creation of custom object detectors will be essential. For this purpose, our toolchain includes *OpenTPOD*. This tool enables even a developer who has no skill in computer vision or machine learning to easily create custom object detectors without writing a single line of code. At the end of this Ajalon stage, object detectors are available for all task-relevant objects.
- In the fourth and final stage of Ajalon, the developer uses a finite state machine (FSM) editor called *OpenWorkflow* to bring together all the pieces from the earlier stages. OpenWorkflow models user actions as transitions in a task-specific FSM. Each state represents partial completion of the task, or an error. Object detectors are used to detect the current state of the task, and the transitions are inferred from final and prior state. To trigger the next desired transition, the developer can add visual and verbal guidance to a state. At runtime, when the FSM enters that state, the associated visual and verbal guidance will be presented to the user. The output of OpenWorkflow is executable code for the new Gabriel task.

We describe Ajalon in detail in the rest of the paper. Section 2 provides background on Gabriel. The four stages of Ajalon are presented in depth in Section 3. Section 4 presents our evaluation of Ajalon through microbenchmarks of individual components, as well as an end-to-end user study of the entire toolchain.

2 | BACKGROUND AND RELATED WORK

In this section, we introduce technical background and prior work on the Gabriel platform, applications, and authoring process. For brevity, we often shorten "the Gabriel platform" to just "Gabriel."

2.1 | Gabriel platform

Gabriel enables its applications to preserve crisp quality of experience (QoE) while overcoming the resource limitations of small, lightweight, and energy-efficient mobile devices. A key metric of QoE is the perceived end-to-end latency, which ideally falls within into the range defined by the latency bounds listed in Figure 2. Gabriel achieves this by enabling a mobile device to offload compute-intensive operations to a nearby *cloudlet* rather than to the distant cloud.[4-6] As shown in Figure 2, Gabriel is an extensible PaaS (Platform as a Service) layer that we have created for WCA applications. Sensor data collected from a wearable device is preprocessed (e.g., compression and encoding) before being streamed to a cloudlet via

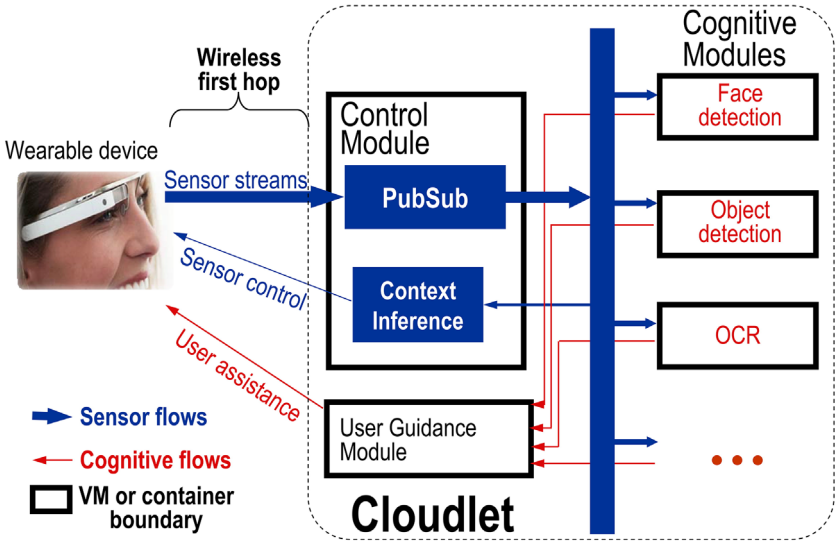


FIGURE 2 Gabriel Architecture (Source: Satyanarayanan et al.[6]) [Colour figure can be viewed at wileyonlinelibrary.com]

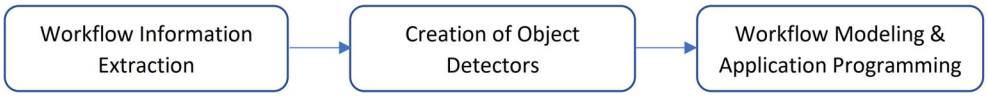


FIGURE 3 Current Gabriel applications development process [Colour figure can be viewed at wileyonlinelibrary.com]

a wireless connection. The sensor streams are further processed on the cloudlet by a collection of *cognitive modules* that employ compute-intensive algorithms such as object detection and speech recognition. Depending on the level of trust in a particular setting, each module can be encapsulated within a virtual machine (VM), container, or process boundary. The *control module* in Gabriel is the focal point for all interactions with the wearable device. A publish-subscribe (PubSub) mechanism decodes and distributes the incoming sensor streams. Cognitive module outputs are integrated by a task-specific *user guidance module* that performs higher-level cognitive processing. The application-specific code in the user guidance module typically consists of a task state extractor and a guidance generator. The task state extractor maps inputs from cognitive modules into the user's current point of progress on the task. Detection of this state then triggers task-appropriate visual, verbal, or tactile guidance that is transmitted back to the wearable device. Since the Gabriel back-end embodies all task-specific components, applications can be easily ported to different wearable devices. We have successfully used a diversity of devices such as Google Glass, Microsoft HoloLens, Vuzix Glass, and ODG R-7.[7]

2.2 | Authoring Gabriel applications

Even using the Gabriel platform, WCA application development remains a nontrivial endeavor. The current development process involves three main steps that are illustrated in Figure 3: (a) workflow extraction, (b) object detector creation, and (c) workflow modeling. These steps require very different types of expertise. First, for workflow extraction, a task expert is needed who understands how best to perform the task at hand, as well as the potential mistakes someone may make when trying to complete the task for the first time. Second, for creating task-specific object detectors using a DNN, a computer vision expert is needed. Object detection is necessary for determining when specific steps of the task have been completed. Note that the vision and task experts may need to iterate on the task workflow to make it amenable to easily-trained and accurate vision models. Third, a software developer combines the individual components into a front-end Android app that can be published in the Google Play Store, and back-end software that can be deployed on cloudlets.

Ajalon's goal is to simplify the process of creating a Gabriel application so that a single developer, working closely with a task expert, can create the application in a short time. This requires creating higher-level abstractions, providing common reusable modules, and reducing the amount of required computer vision and software engineering knowledge. This goal is different from recent efforts in speeding up software development[10] by leveraging parallelism across several developers. While such efforts can shorten the elapsed time (i.e., wall clock time) for the development process, it does

not reduce the total number of person-hours invested. In contrast, Ajalon's goals are twofold. The first goal is reducing the total number of person-hours required for development. The second goal is enabling developers who do not have expertise in computer vision and machine learning.

2.3 | Workflow extraction

Ajalon's goal of simplifying the creation of Gabriel applications has some overlap with previous work on *workflow extraction*. In this context, the term "workflow" is defined as a machine-readable description of a sequence of activities or working steps, that a user must carry out in order to complete a specific task. Workflows, by definition, are task-specific and often crafted in an ad hoc fashion by task experts. Automatically extracting a workflow from a video depicting a task reduces application development time, especially for lengthy tasks with many steps.

Previous research on workflow extraction was concerned with creating tools for manual workflow extraction and computer vision algorithms for automated video analysis. The Computer Cooking Contest, which has been run since 2008, challenges researchers to extract workflows from cooking recipes available from sources such as Wikipedia. In an open-challenge system called CookingCakeWf, researchers described the steps required to make a recipe using a control flow, and they described the ingredients and their products using a data flow.[11] Mura et al.[12] proposed a tool named IBES that facilitates manual working step separation and annotation. Kim et al.[13] developed ToolScape, which helps verify workflow descriptions scripted by multiple people from videos depicting tasks. For automated video analysis, Nater et al.[14] developed an unsupervised machine learning method to extract the workflow information from the videos of an assembly line. In contrast to these earlier efforts, Ajalon leverages the fact that our headsets capture videos from a first-person perspective.

The current implementation of Ajalon assumes that the workflow of the target application has specific simplifying attributes. First, Ajalon assumes a linear workflow that can be readily decomposed into a sequence of steps. This is a common feature of tasks such as product assembly or equipment servicing. Second, Ajalon assumes that the task follows a canonical workflow; that is, there is a single preferred procedure (or if several are comparable, one procedure can be selected), and the steps in the task follow a prescribed order. Third, Ajalon assumes that the correct completion of a step (and transitively, all steps that precede it) can be visually determined using object detection algorithms. This excludes, for example, a step such as one that requires putting a small object inside of a large object but leaves the large object visually unchanged. In spite of these simplifying assumptions, Ajalon is able to cover a wide range of WCA applications. Future versions of Ajalon may relax some of these assumptions.

3 | AJALON'S DESIGN

Ajalon enables a development team consisting of a task expert and a software developer to jointly create a Gabriel application. We assume that the task expert understands the steps of the task well, but has no software skills. In contrast, the developer is skilled in software development, but is unfamiliar with the task. Neither person needs to be an expert in computer vision or machine learning, even though these are central to Gabriel applications. Ajalon codifies, simplifies and partially automates the end-to-end authoring steps to help the development team create a Gabriel application.

Figure 4 illustrates the four steps of the Ajalon toolchain that were mentioned in Section 1. The process starts by capturing an example video that shows the first-person viewpoint of a task performed by an expert. The video may also illustrate common mistakes and guidance on how the expert fixes them. Second, the development team uses Preprocessing Tool (PT) to automatically segment a video into time-stamped working steps, and to generate a list of task-relevant objects seen in the video. The extracted set of working steps is imported into PTEditor to manually correct mistakes made by PT. Third, the development team uses OpenTPOD to create a DNN-based object detector that can accurately distinguish between all the task-relevant objects discovered by PT. Finally, the outputs of PTEditor and OpenTPOD are used by OpenWorkflow to create an FSM whose state transitions are triggered by visual changes in the task state. The output of OpenWorkflow is executable code for the Gabriel application that embodies the workflow of the task. We next describe the steps after capturing a video in detail.

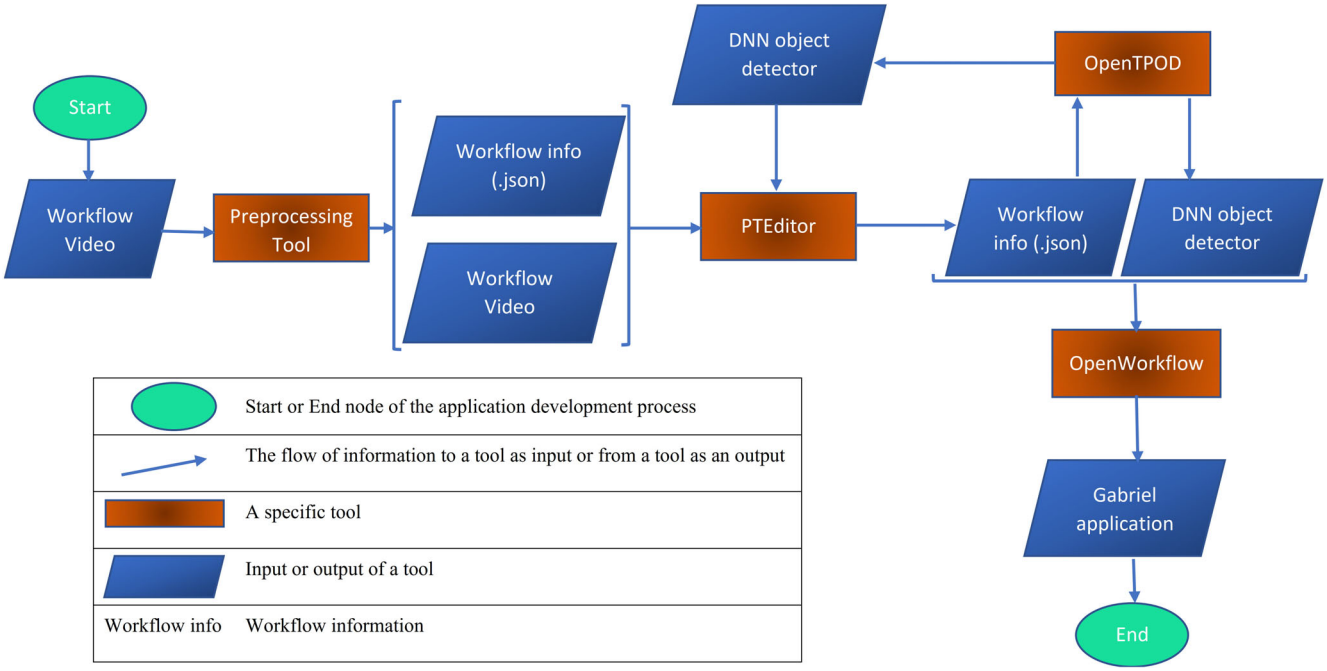


FIGURE 4 Ajalon pipeline overview [Colour figure can be viewed at wileyonlinelibrary.com]

3.1 | Preprocessing tool

PT targets a manual assembly process in which parts are combined by human hands. The input to PT is a first-person video of an assembly process in which parts are added in sequence. In such a video, assembly actions are viewed as interactions between hands and objects (i.e., parts or workpieces). PT splits this video into segments, each corresponding to a working step. PT assumes that only one part is added per step, and that each step changes the appearance of a partially-assembled workpiece. Step completion can therefore be confirmed from workpiece appearance. PT detects the boundary between working steps by finding a lull in hand-object interactions. This approach has proved to be statistically reliable,[15] in that similar results are achieved under consistent conditions. According to our previous work,[16] this boundary-detection approach achieves over 80% accuracy on videos of different assembly tasks. The detailed process of the approach is described below.

1. Run EgoNet,[17] a two-stream network that predicts per-pixel likelihood of action-objects from first-person videos, to obtain salient regions that represent the region of interest (ROI) of a video frame. Instead of going through each pixel within the salient regions, PT applies clustering and contour extraction to obtain the bounding boxes of ROIs, as illustrated in Figures 5 and 6. This narrows the search space for hand-object interaction to a few bounding boxes.
2. Run YOLOv3,[18] a real-time object detector, to detect human hands in each frame. Assuming that the first-person video only captures the operations of one worker, the object detector is trained to detect the regions where the left and/or right hand of the worker appear.
3. Recognize hand-object interaction by detecting the overlap between hand regions and ROIs. In Figures 5(D) and Figure 6(D), the detected interaction regions are highlighted by green boxes, while hand regions and other ROIs are marked with blue and red boxes, respectively. After going through all frames, PT generates a one-dimensional binary array that represents the occurrences of interaction in the video as an interaction signal.
4. Smooth and threshold the interaction signal to reduce false-positive and false-negative errors in the previous steps. For example, a false-positive error may occur when an unintentional hand movement that has no interaction with objects is detected in an ROI. A false-negative error may occur if PT misses hands or ROIs. PT overcomes such errors by assuming continuity of hand–object interaction, and taking into account detection in neighboring frames. A smoothing technique is used to estimate the interaction between the interaction signal of a video and the frame index. As illustrated in Figure 7, the interaction signal is represented as a probability of hand-object interaction along a video

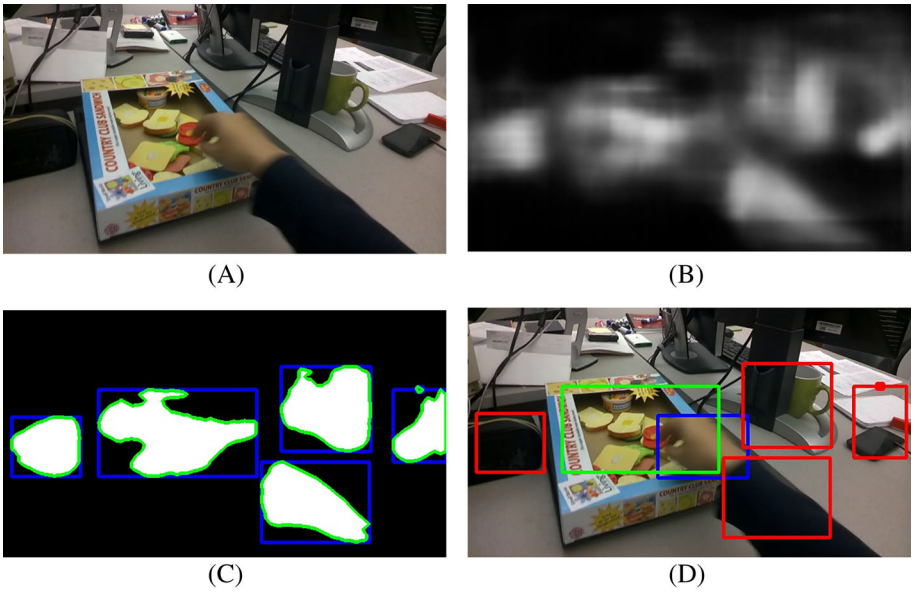


FIGURE 5 Working step separation in the *making sandwich* assembly example. (A) Original frame. (B) EgoNet saliency detection. (C) Region of interest (ROI) bounding box extraction. (D) Hand-object interaction detection [Colour figure can be viewed at wileyonlinelibrary.com]

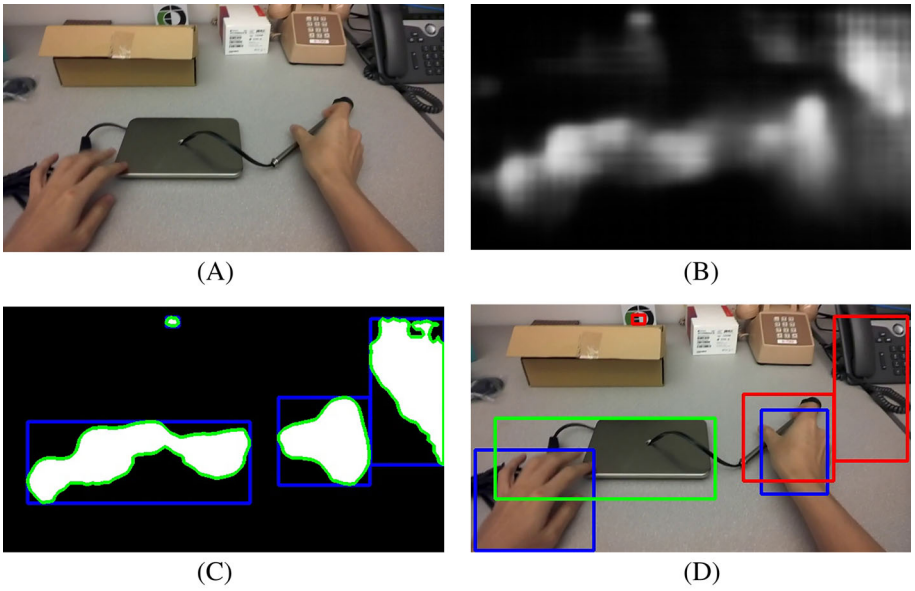


FIGURE 6 Working step separation in the *IKEA lamp* assembly example. (A) Original frame. (B) EgoNet saliency detection. (C) Region of interest (ROI) bounding box extraction. (D) Hand-object interaction detection. [Colour figure can be viewed at wileyonlinelibrary.com]

frame index. In Figure 7(A), the interaction signal from the detector described above is visualized as a binary 0/1 signal. Then, based on our prior experience in extracting task steps from video, we smooth the results by convolving with a Hanning window of size 19, selected empirically as in other approaches to activity recognition.[19] After smoothing the signal, we use a threshold of 0.5 to determine if an interaction happens at the considered frame. We set the threshold value to 0.5, assuming that no prior knowledge about the assembly process is available. As illustrated in Figure 8, with different threshold values, we obtain different working step segmentation results on an 8-s video example. For instance, with a lower threshold value (i.e., 0.4), more frames are classified as ones that contain hand–object interaction. As a result, more working-step segments are generated. In contrast, with a higher threshold value (i.e., 0.6), fewer hand–object interactions and thus fewer working-step segments are detected. By tuning the threshold value, PT can capture actual working steps while removing most erroneous hand–object interactions due to occasional misdetections of hands or object ROIs in some frames.

5. Cluster successive frames containing hand-object interactions to represent a continuous working step. With 25-fps videos, the system eliminates small clusters that are shorter than 12 frames, because we assume that meaningful actions or steps require time intervals of greater than half a second. The result of the entire process is illustrated in Figure 9. Here, PT has segmented the video into six steps (labeled A–F) of intense hand–object interactions, separated by periods of no interactions between hands and objects in the video.

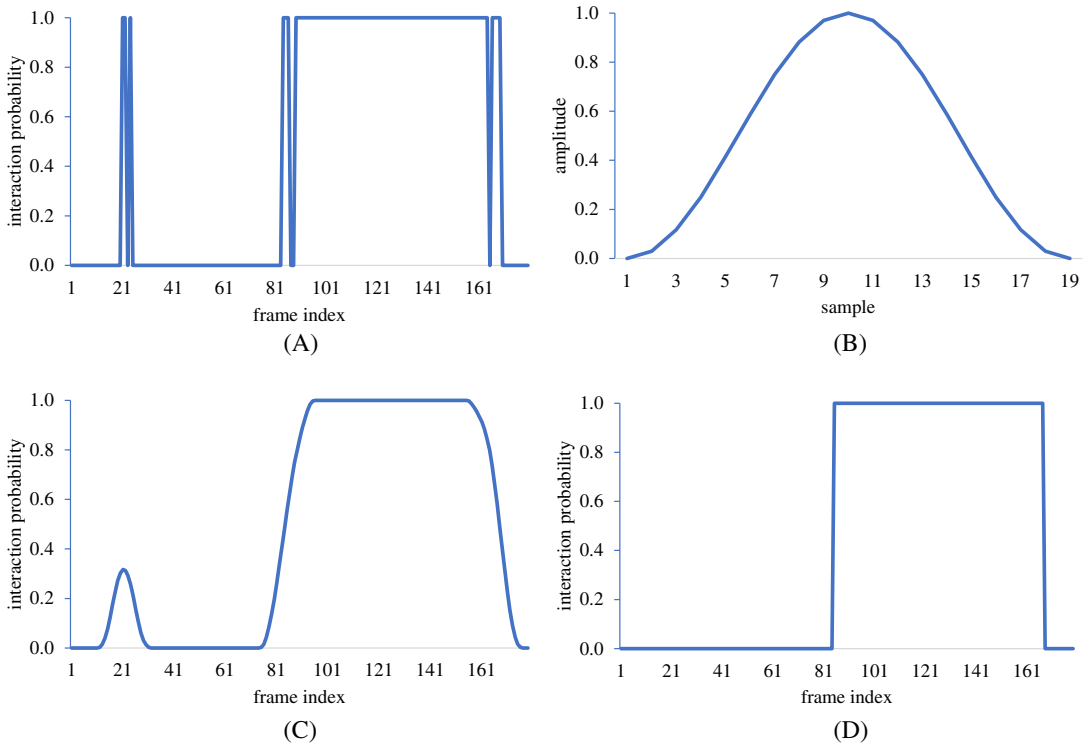


FIGURE 7 Noise reduction in working step detection. (A) Raw binary data. (B) Hanning window with size of 19. (C) Smoothing result. (D) Noise reduction after applying thresholding [Colour figure can be viewed at wileyonlinelibrary.com]

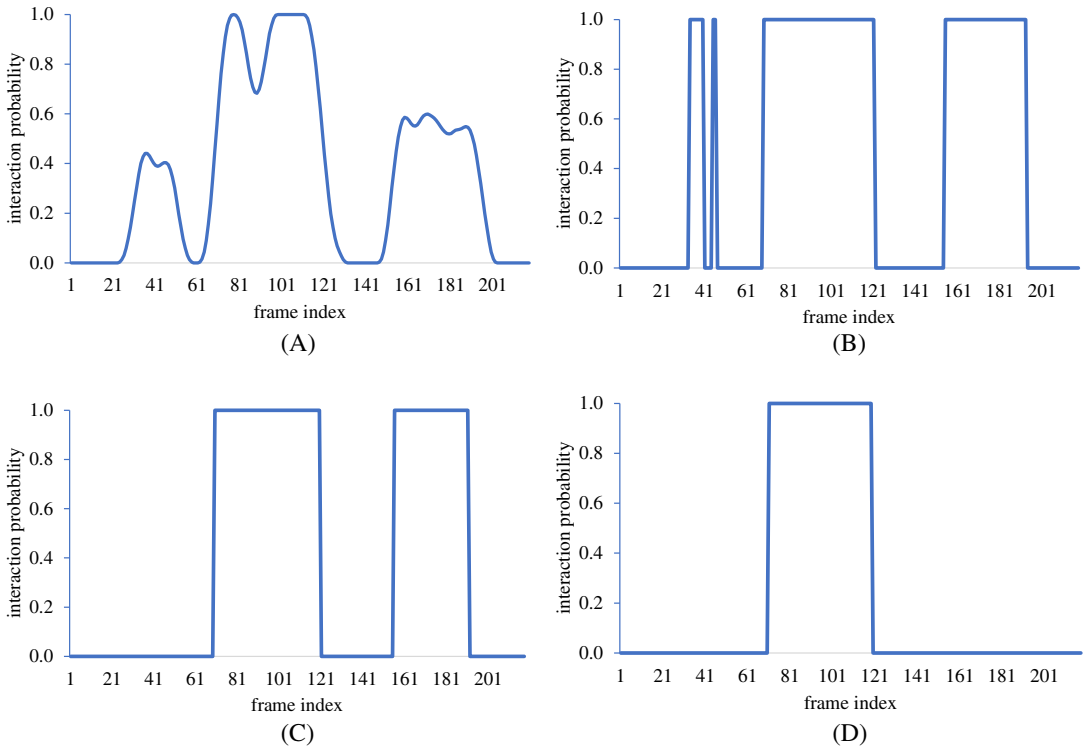


FIGURE 8 Results of hand-object interaction detection with different threshold values. (A) Interaction probability over time. (B) With a threshold value of 0.4, four clusters of hand-object interaction frames are detected. (C) With a threshold value of 0.5, two clusters of hand-object interaction frames are detected. (D) With a threshold value of 0.6, only one cluster of hand-object interaction frames is detected [Colour figure can be viewed at wileyonlinelibrary.com]

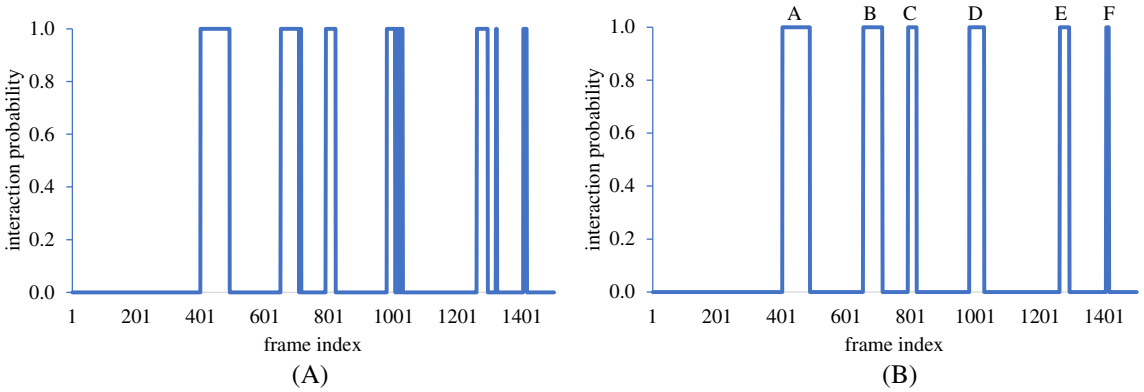


FIGURE 9 Refined working steps segmentation result on 1500 frames using smoothing, thresholding, and clustering. (A) Raw binary interaction result. (B) Refined sequence with six working steps including A, B, C, D, E, and F [Colour figure can be viewed at wileyonlinelibrary.com]

Algorithm 1. Extracting objects that the person interacted with

procedure PROCESS(video)
  Initialize `object_dictionary` with boxes of interest and their features in frame 0.
  Initialize multiple trackers with those boxes.
  for frame in [frame1, frameN] in video **do**
    Extract boxes of interest in frame.
    Filter boxes of interest to reduce noisy frames. The result is `filtered_boxes`.
    Extract features for those `filtered_boxes`.
    Match features of `filtered_boxes` with features of `object_dictionary`.
    Store retrieved boxes as `d_boxes` and retrieved labels as `d_labels`.
    Track boxes by using trackers. Store the boxes as `t_boxes` and their labels as `t_labels`.
    Pick boxes in `d_boxes` that do not overlap with `t_boxes` for creating new trackers.
    Update `object_dictionary` with `d_boxes` and `t_boxes`.
    Detect hands in the frame. Store bounding boxes for hands in `h_boxes`.
  end for
  return object boxes in `d_boxes` and `t_boxes` that overlap with the hand boxes `h_boxes` across all frames.
end procedure

Algorithm 1 shows the final part of preprocessing, which extracts the list of objects used in each step. The output of this algorithm is an object association list that maps each working step to objects involved in interactions in that step. We bootstrap the algorithm by using unsupervised object recognition on the frames, as described by Pham et al.[16] Only a subset of these objects may be used in any particular step. The lists `d_boxes`, `t_boxes`, and `h_boxes` are three crucial elements for recognizing which objects likely used in a particular segmented step. The first, `d_boxes`, is a set of bounding boxes of objects detected in the frames of the segment. We track these objects using a tracker, which helps fill in gaps where detections fails because the object is partially occluded, and store the tracked bounding boxes in `h_boxes`. We store the set of unique visible objects based on feature similarity in `object_dictionary`, which is updated on each frame. Finally, we return the set of objects that consistently associate closely with the hands, assuming these are the objects that are used in the step. The end results of PT are a temporal segmentation of the video into a sequence of steps, and a list of visual regions corresponding to objects associated with each working step. Note that the real-world identities of the objects are not actually determined here. In other words, the algorithm can detect the objects and can label/assign each a unique id, but cannot actually name the object. For instance, if tables and chairs appear in a video, the algorithm may label tables as Type 1 and chairs as Type 2. However, the algorithm cannot associate the names "table" and "chair" with them.

3.2 | PTEditor: correcting errors in preprocessing

Because of the limitations of computer vision, the workflow extraction process described in Section 3.1 is imperfect. For example, the object association list for a working step may miss an object, or include one in error. Another possible error is wrong lull detection. This happens when no hand–object interaction has been detected for a short while during a working step, due to inaccuracy of RoIs detected by EgoNet and/or hand detection errors from the YOLOv3 model. Such errors would result in an incorrect demarcation of a working step. These errors have to be manually corrected by the development team before the extracted workflow can be used. PTEditor, shown in Figure 10, is created for this purpose. This tool provides support for browsing and editing a workflow, as described below.

Browsing: A video can be loaded and viewed on the window in the left panel. The "«" and "»" buttons at the bottom of that panel enable quick navigation to the first frame of the previous, or next, working step, respectively. Frame-by-frame advance through the video is also possible. A horizontal scrollbar presented under the window shows segments representing working steps in different colors.

Editing: The "Split" and "Merge" buttons under the horizontal scrollbar can be used to correct erroneous demarcations of a working step. As their names imply, these buttons can divide a step, or merge two consecutive steps centered at the current frame. For each frame, the right panel shows the visual completion state and object association list for the working step of the current frame. Objects can be manually added to or deleted from an association list. Upon completion of edits, pressing the "SAVE" button at the bottom creates a JSON file that is the external representation of the extracted workflow.

3.3 | OpenTPOD: an open toolkit for painless object detection

The next component of the Ajalon toolchain is OpenTPOD (Open Toolkit for Painless Object Detection). The use of computer vision in a Gabriel application relies on accurate detection of objects in the association lists of working steps. OpenTPOD enables the creation of an accurate detector for these objects even if the development team lacks machine learning and computer vision skills. It does this by helping create a training set for a DNN, and then training the DNN. To reduce the size of the training set required, OpenTPOD uses a technique called *transfer learning* that starts with a pretrained model for a set of objects from a public dataset, such as Pascal VOC,[20] and ImageNet,[21,22] leveraging the pretrained representation to bootstrap training on custom data. Even with transfer learning, the accuracy of DNN-based object detectors is correlated with the amount of labeled training data.[23] However, capturing images and labeling them

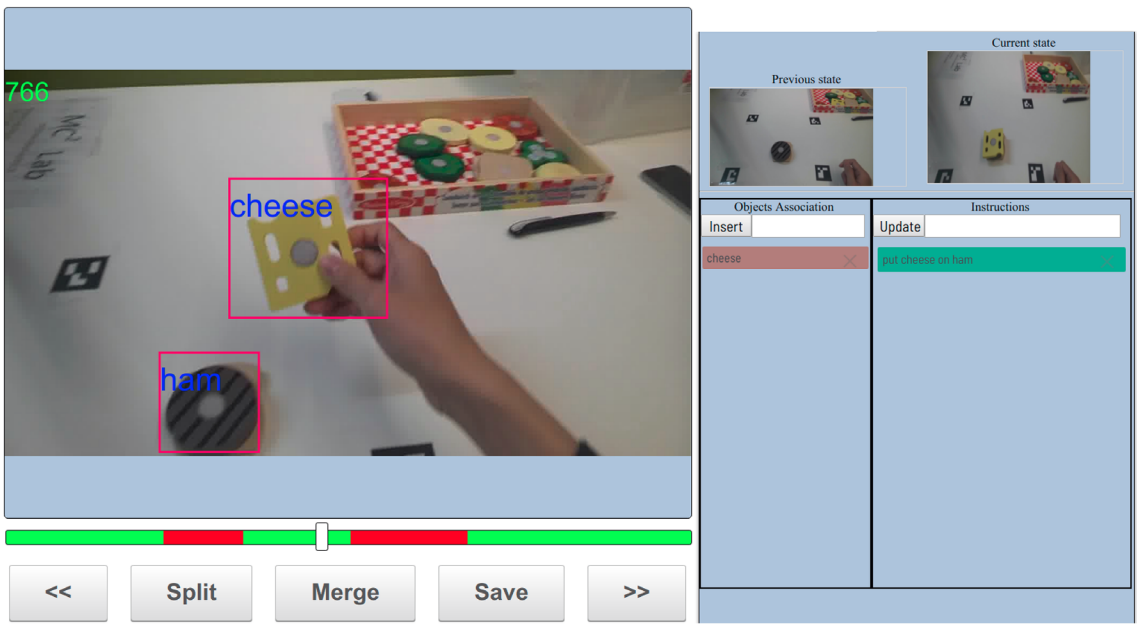


FIGURE 10 Graphical user interface of PTEditor [Colour figure can be viewed at wileyonlinelibrary.com]

with bounding boxes is laborious. OpenTPOD seeks to greatly reduce the human effort in capturing an labeling training data. OpenTPOD also automates and simplifies the process of training the DNN.

The developer captures a few short videos of each object used for the task. These are typically captured with a smartphone, from different viewing angles and against multiple backgrounds. The videos are uploaded to OpenTPOD using a web browser. A few minutes of video will contain several thousand frames with the object. Fortunately, as described below, only a handful of these frames need to be manually labeled. This is done by drawing a bounding box around the object in those frames.

Figure 11(A) illustrates the labeling process. First, the developer finds the initial occurrence of the object in the video and labels that frame. Then, using a *correlation tracking algorithm*,[24] OpenTPOD extrapolates the position of the object (and hence, the bounding box) in subsequent frames. Because of imperfect tracking, the bounding box will drift relative to the actual object in later frames. When the developer judges that the drift is excessive, he pauses OpenTPOD and explicitly labels that later frame. This reinitializes the position of the bounding box for tracking in subsequent frames. Then, OpenTPOD continues its extrapolation until the drift is again judged to be excessive by the developer. This process of extrapolation and explicit labeling is continued until the end of the video. In our experience, this approach of labeling followed by tracking can reduce the number of frames that need to be manually labeled by a factor of 10 or 20.

Next, OpenTPOD performs a data cleaning and augmentation pass. Because of interframe correlations, many of the object examples may be close to identical. OpenTPOD will eliminate the near duplicate examples, as these will not help in the training process. Optionally, data augmentation can be employed. This creates more images by applying image manipulation techniques on the original image such as rotation, flipping, changes to the color scheme, and distortion. Such augmentation has been shown to help produce more robust object detectors.[25]

Finally, OpenTPOD can automate the transfer learning of a DNN model using the collected dataset with the user interface shown in Figure 11(B). By default, OpenTPOD uses a state-of-the-art Faster R-CNN VGG network[3] pretrained on the Pascal VOC dataset,[20] though other network architectures can be used as well. The transfer learning process used by OpenTPOD is shown in Figure 12. Negative examples are mined from the video background; these are parts of the frames not included in the object bounding boxes. The training is started as a batch process that uses a standard, scripted learning schedule, and generates both the final Tensorflow model, and a Docker container with the code to run the model.

Overall, OpenTPOD can greatly reduce both the labeling effort and in-depth machine learning knowledge needed to effectively train and deploy a DNN-based detector. We note that OpenTPOD is largely a stand-alone tool that can be used separately from the rest of Ajalon.

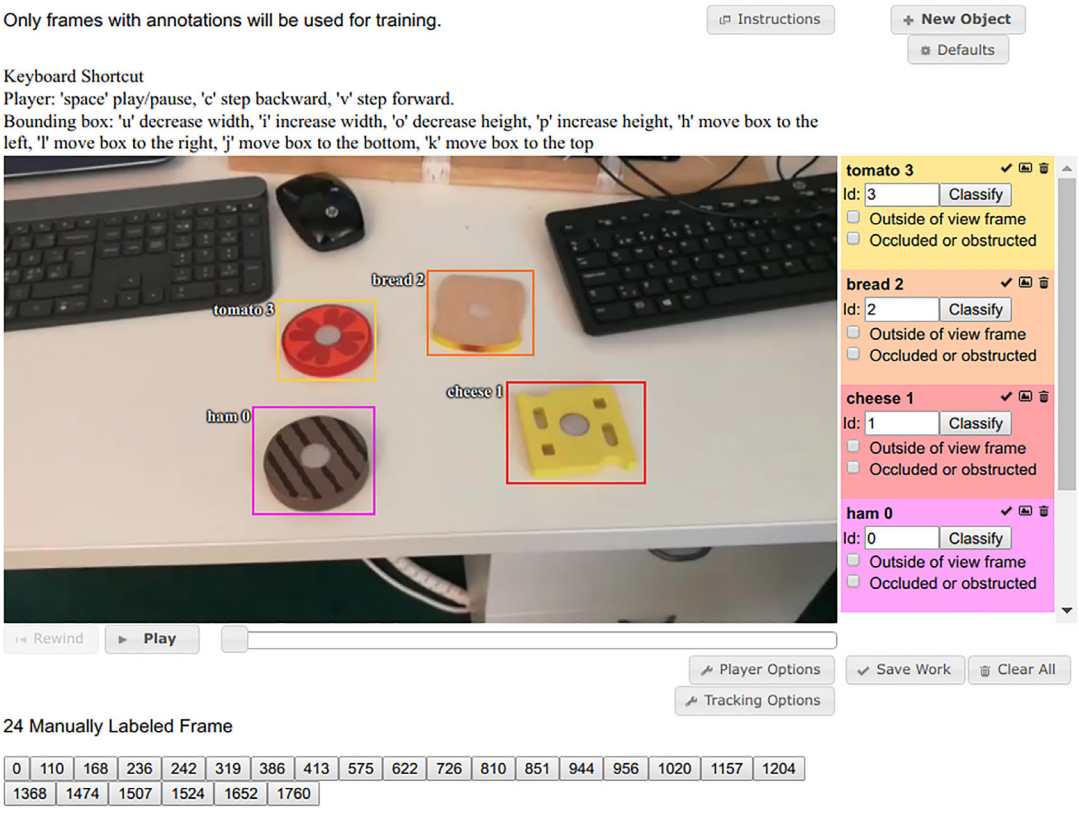
3.4 | OpenWorkflow: bringing the pieces together

The final stage of Ajalon is OpenWorkflow, a FSM editor that automatically generates executable code for an FSM. Each state can be viewed as a program counter value in the workflow of the task. At the start, the states of the FSM correspond to the output of PTEditor. The editing function of OpenWorkflow can be used to add new states that, for example, correspond to distinct error states resulting from frequently committed errors by users. The editing function lets one specify the visual cues that distinguish a state, using one or more object detectors that were created by OpenTPOD. As mentioned earlier, Ajalon assumes that progress on a task can be detected by the visual appearance of its parts. For example, the user might put a piece of ham on top of a piece of bread when making a sandwich. The detection of a "ham-on-top-of-bread" object in a video frame indicates transition into this state. Figure 13 shows a sample task workflow as an FSM that can be edited using this tool. The editing process can also indicate the visual and verbal guidance provided to a user as annotations on state transitions. When the editing process is complete, OpenWorkflow generates compiled code that implements the FSM as an executable application.

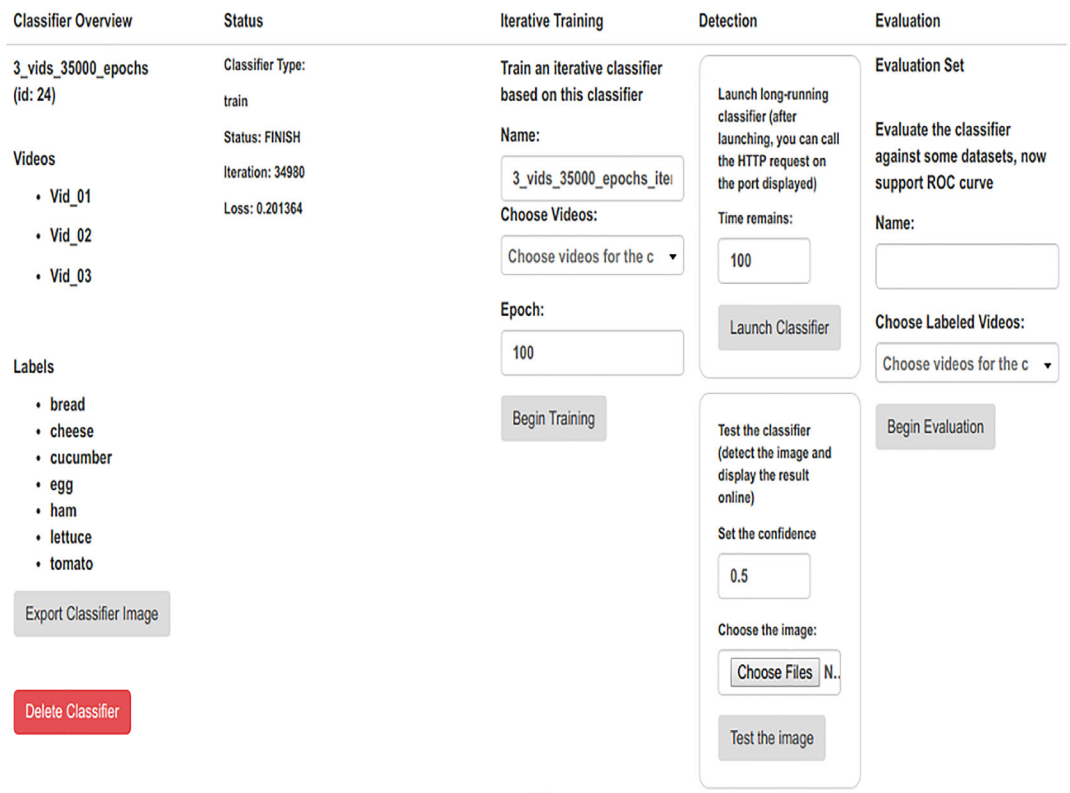
4 | EVALUATION

We explore the effectiveness and usability of Ajalon through two sets of questions. There are four questions in the first set, each pertaining to a different element of the Ajalon toolchain. These are answered via microbenchmarks in Section 4.1:

1. Is the automatic working-step segmentation of PT accurate?
2. Can a PT-extracted workflow be easily edited using PTEditor?



(A)



(B)

FIGURE 11 OpenTPOD GUI. (A) Data labeling user interface. (B) Deep neural network training and testing user interface. [Colour figure can be viewed at wileyonlinelibrary.com]

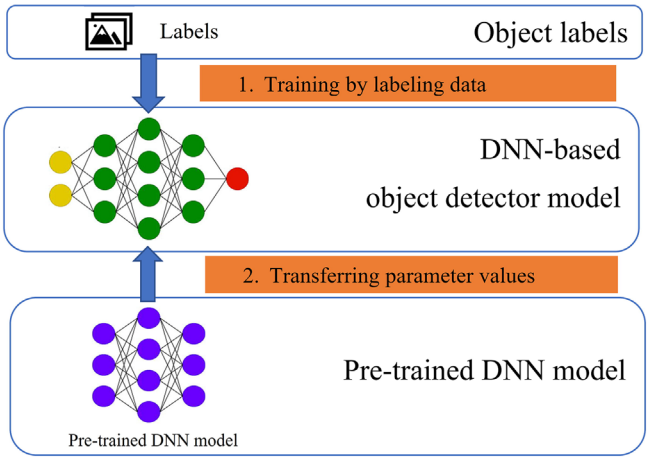


FIGURE 12 Transfer learning process used by Open Toolkit for Painless Object Detection (OpenTPOD). The initial model weights come from a pretrained deep neural network model. Then weights are tuned by training on labeled data of the new object we wish to detect [Colour figure can be viewed at wileyonlinelibrary.com]

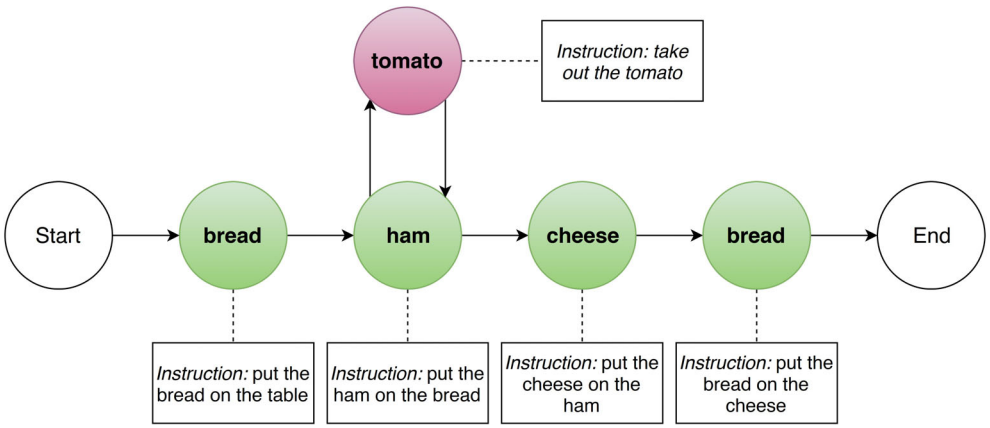


FIGURE 13 A finite state machine (FSM) for an application that assists in making a sandwich. White circles represent the start and end states of the process. Green circles represent correct states that help the user finish the task. The red circle represents an error state that requires the user to recover to the last correct state to finish the task [Colour figure can be viewed at wileyonlinelibrary.com]

3. Can facility with OpenTPOD easily be acquired, and how good are its object detectors?
4. Is OpenWorkflow easy and effective to use for adding error cases, modeling workflow, and generating executable code?

Questions in the second set pertain to the Ajalon toolchain as a whole, and are answered in Section 4.2 through a user study:

5. Do developers find Ajalon easy and effective to use?
6. Does Ajalon improve developer productivity?

4.1 | Microbenchmarks

Dataset. We used 40 videos, with a total length of 5.44 h, for evaluating the accuracy of PT. Half of these videos show the process of assembling HAPE toy model 1, as shown in Figure 14, while the other half show HAPE toy model 2, as depicted in Figure 15. In addition, we used one 58-s first-person video that captures the process of *making sandwich* (see Figure 16) to evaluate the whole pipeline of Ajalon and the performance of each module involved. The detailed experimental results are presented as follows.

4.1.1 | Accuracy of PT

Metric: First, precision, recall, and intersection over union (IoU) are used for evaluating the accuracy of EgoNet-based ROI detection. IoU represents how close a detected bounding box is to the ground truth. In practice, the groundtruth boxes

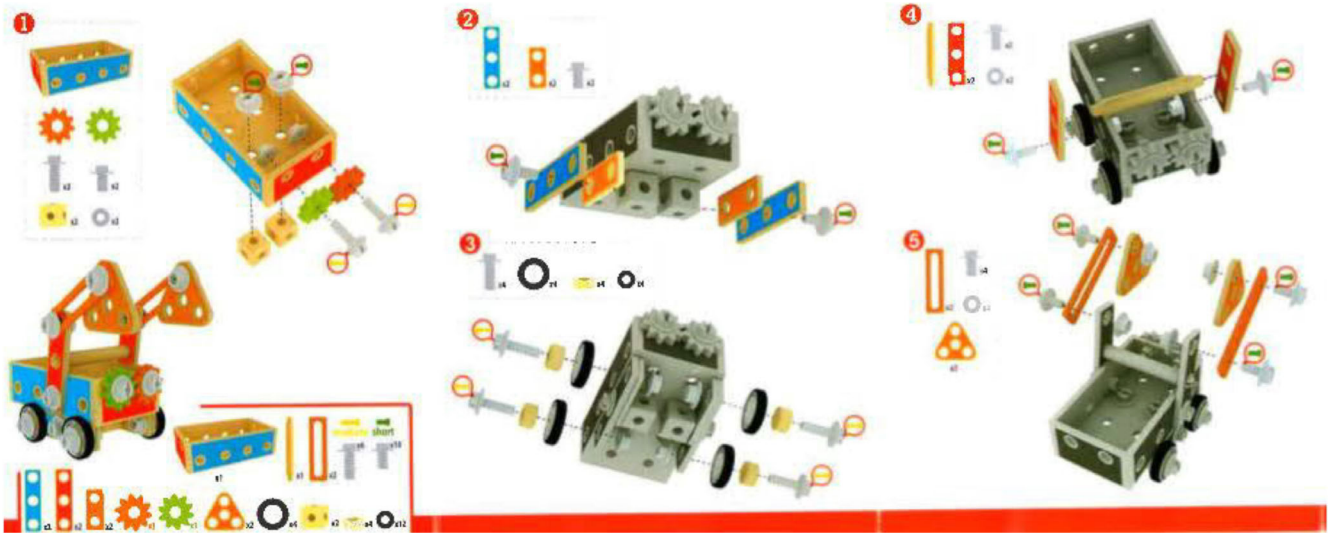


FIGURE 14 Assembly instructions for HAPE toy model 1[16] [Colour figure can be viewed at wileyonlinelibrary.com]

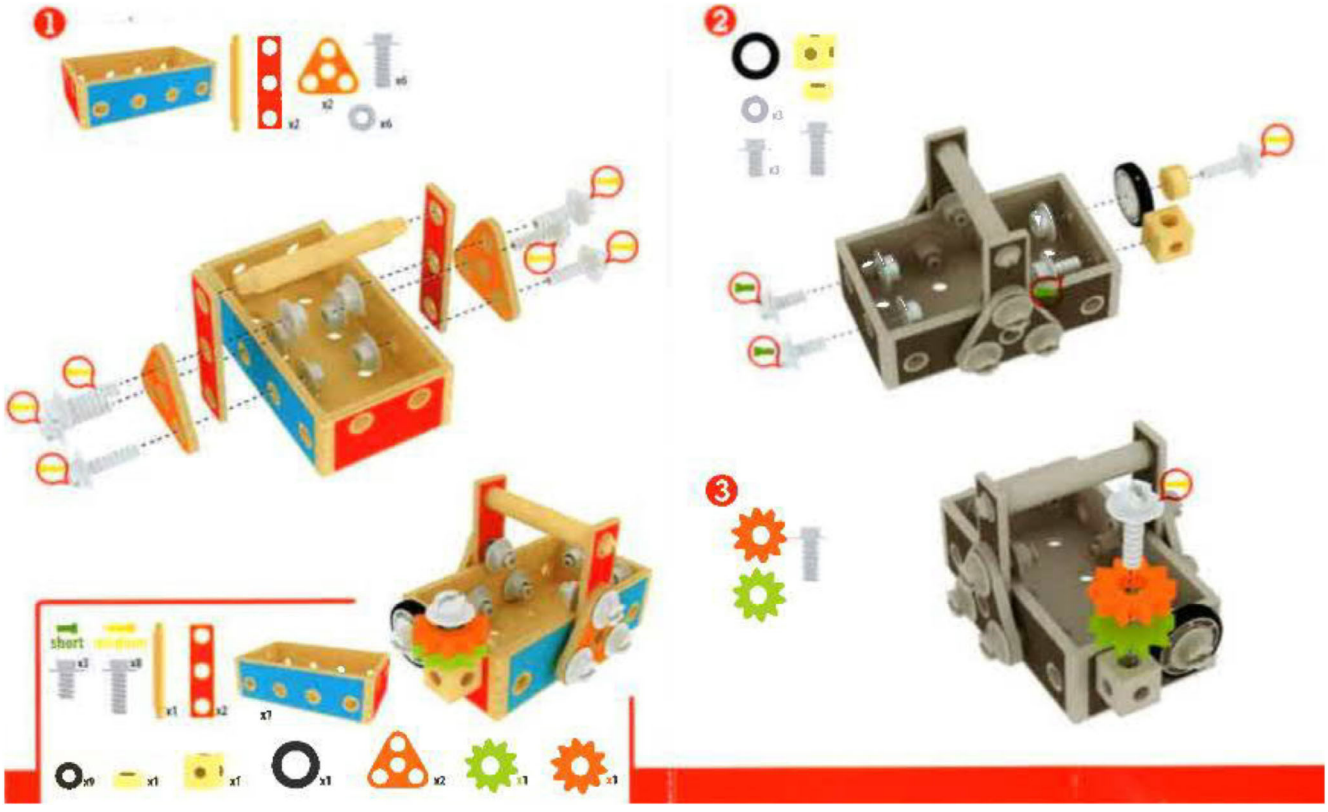


FIGURE 15 Assembly instructions for HAPE toy model 2[16] [Colour figure can be viewed at wileyonlinelibrary.com]

are manually labeled, as shown in Figure 17. A detection is classified as true positive if its IoU value exceeds a predefined threshold. We measured the effectiveness of automated working step segmentation using Boundary Detection Accuracy (BDA),[26] which compares automated and manual results using the following equation:

$$\mathrm{BDA} = \frac{\tau_{\mathrm{db}} \cap \tau_{\mathrm{mb}}}{\max(\tau_{\mathrm{db}}, \tau_{\mathrm{mb}})} \in [0, 1], \tag{1}$$

τ_{db} is the working step boundary detected automatically. τ_{mb} is the one labeled manually.

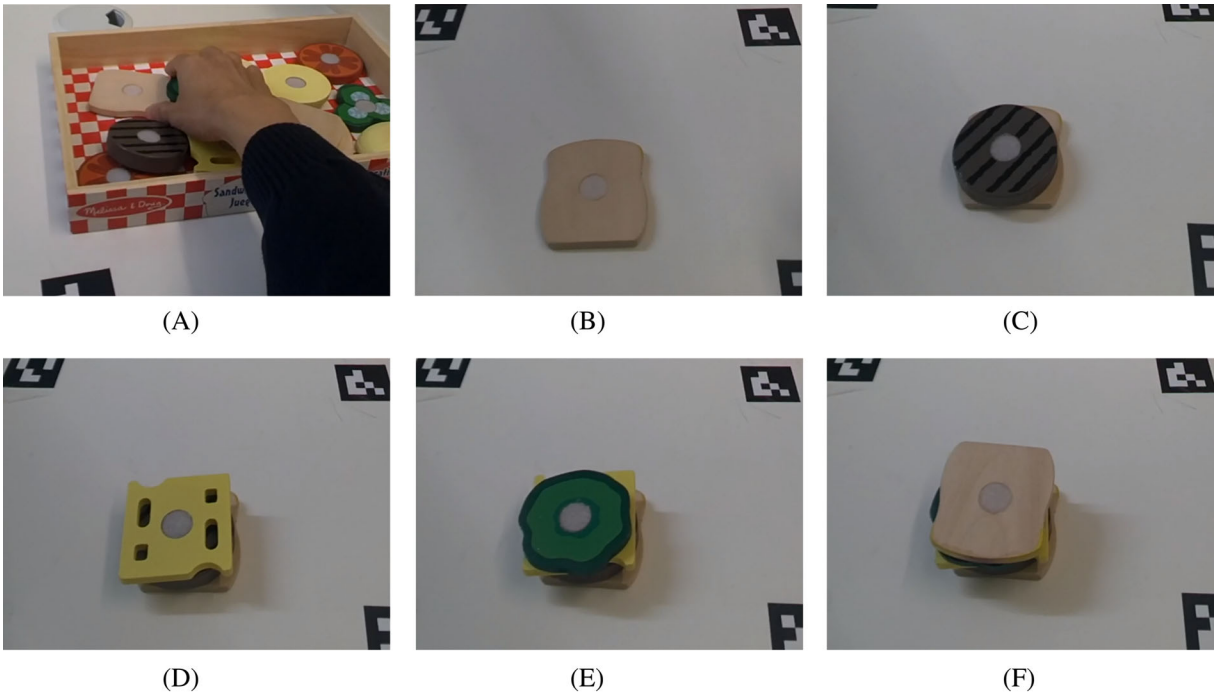


FIGURE 16 Workflow process of making a sandwich toy with the toy set. (A) Toy set box containing all components. (B–F) Assembly steps in order [Colour figure can be viewed at wileyonlinelibrary.com]

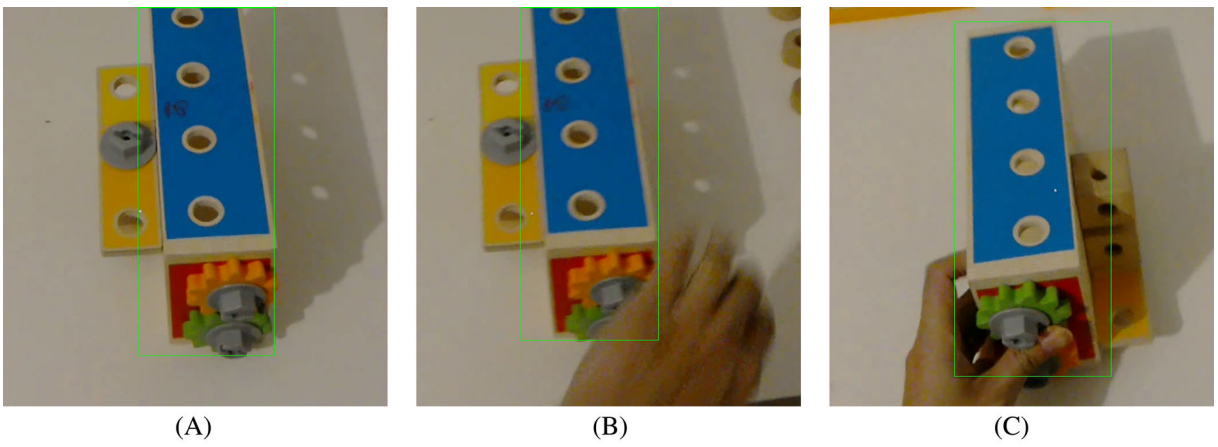


FIGURE 17 Examples of groundtruth region of interest (ROI). (A) A groundtruth bounding box in green over an ROI before the interaction. (B) A groundtruth bounding box in green over an ROI during the interaction. (C) A groundtruth bounding box in green over an ROI at the end of the interaction [Colour figure can be viewed at wileyonlinelibrary.com]

Results. As shown in Figure 18, when the IoU threshold gets lower than 0.4, the accuracy is reasonably high, with over 0.8 recall and 0.52 precision at IoU threshold of 0.3. However, the precision and recall decrease as the IoU threshold value increases, with the degradation rate accelerating as the IoU threshold passes 0.4. Although the accuracy of EgoNet-based ROI detection is only acceptable, the EgoNet-based method is still able to make the working-step segmentation accuracy reasonably high. The average BDA on the entire dataset was 0.85, confirming that the working-step segmentation algorithm effectively detects working steps in the video. However, this result also indicates that significant manual effort is required to eliminate the residual error. To achieve that result, Hanning smoothing must be used while grouping working steps. As shown in Figure 19, using Hanning smoothing increases the average BDA by approximately 13%. Notably, in HAPE toy model 2, the average BDA increases by 16%. On the other hand, it still requires significant manual effort to eliminate the residual error. We will discuss potential improvements to PT in the next section.

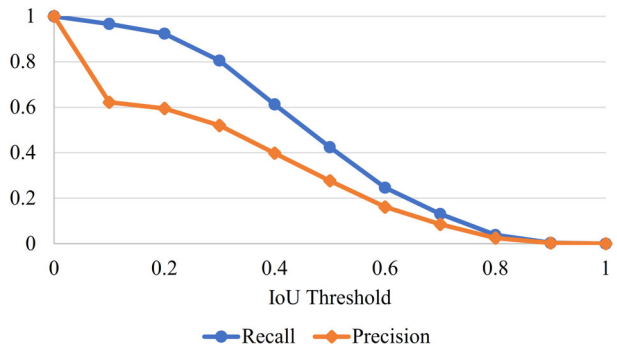


FIGURE 18 Recall and precision of Region of interest detection with different intersection over union threshold values [Colour figure can be viewed at wileyonlinelibrary.com]

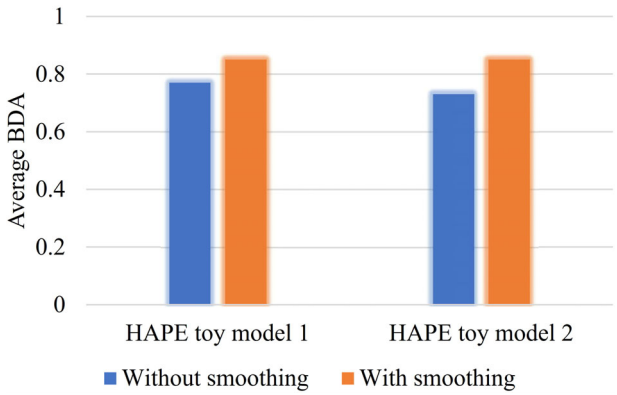


FIGURE 19 Comparison of average boundary detection accuracy with and without Hanning-based smoothing [Colour figure can be viewed at wileyonlinelibrary.com]

4.1.2 | Effectiveness of PTEditor

Approach: A sandwich toy assembly video was used as an editing benchmark. One of the authors, who was very familiar with the assembly process, played the role of a task expert. He was given four sample workflow videos, each showing a different variant of making a sandwich. These four variants consisted of the following items between two slices of bread: (a) cheese, on top of tomato, on top of ham, on top of cucumber; (b) tomato, on top of ham, on top of lettuce, on top of cucumber; (c) tomato, on top of cucumber, on top of egg; and, (d) ham, on top of egg, on top of cucumber. The task expert used PTEditor on two of the videos, and a separate documenting tool on the other two.

Metric and results. Our metric is the amount of time needed to complete workflow information for each working step. Without PTEditor, it took approximately 4.5 min, whereas with PTEditor, it took only 2.6 min, a savings of 42%. With the support of extracted workflow information from PT, the task expert easily captured the process in the video and quickly documented workflow information by using PTEditor.

4.1.3 | Effort saved by OpenTPOD

Approach: Using two different tools and playing the role of a developer, one of the authors built an object detector for six different pieces of the sandwich toy mentioned earlier. The two tools were OpenTPOD and CVAT[27] from OpenCV. CVAT is a widely used tool for data labeling. By holding the GUI for the labeling of a single frame constant across the two tools, the comparison isolates the value of OpenTPOD's mechanisms for optimizing labeling effort, and for setting up the training. As mentioned earlier, the mechanisms for optimizing labeling effort include propagation of labels across frames through tracking, and the elimination of training examples that are of low value.

Metrics: Three metrics are relevant. The first is labeled objects per frame (*lof*). This is defined as the total number of labels, divided by the number of frames in the videos. The second metric is the number of lines of code (*loc*) that were written by the developer to format annotations, train, and test the DNN. The third metric is the total time in hours (*hr*) that the developer spent in writing these lines of code.

TABLE 3 Benefit of using OpenTPOD

	Task/method	Without OpenTPOD	With OpenTPOD
(a)	Data collecting	0.33 *hr*	
(b)	Data labeling	4497/ 6356 *lof*	1695/6356 *lof*
(c)	Annotation formatting	104 *loc*, 1.08 *hr*	0 *loc*, 0 *hr*
(d)	Coding for training	5 *loc*, 0.58 *hr*	0 *loc*, 0 *hr*
(e)	Coding for testing	7 *loc*, 0.33 *hr*	0 *loc*, 0 *hr*

Abbreviations: *hr*, hour; *loc*, number of lines of code; *lof*, manual labeled boxes over number of frames.

TABLE 4 Performance comparison between non-OpenWorkflow and OpenWorkflow method in modeling the workflow and programming the Gabriel application

	Task/method	Non-OpenWorkflow	OpenWorkflow
(a)	Adding error cases	0.21 *hr/error*	0.29 *hr/error*
(b)	Workflow modeling	0.17 *hr*	0 *hr*
(c)	Programming	352 *loc*, 22.5 *hr*	0 *loc*, 0 *hr*
(d)	Testing and bugs fixing	9.5 *hr*	0 *hr*

Abbreviations: *hr*, hour; *hr/error*, hour per error; *loc*, number of lines of code.

Results: Table 3 illustrates the breakdown of metrics in five steps for the two methods. The five steps include: (a) collection of five videos that contain six sandwich pieces of interest; (b) labeling the videos using OpenTPOD; (c) formatting annotations for training; (d) training the DNN; (e) testing the DNN on a small test set of five images. The results in Table 3 show that OpenTPOD offers a clear advantage. It requires much less manual effort in labeling (fewer than 40% the number of labels), and does not require writing code.

4.1.4 | Effort saved by OpenWorkflow

Approach: To evaluate the benefit of using OpenWorkflow, we conducted a case study. One of the authors, an experienced programmer, was provided with the workflow for the sandwich assembly task and a prebuilt object detector for all the relevant objects in that task. Using these inputs, his goal was to create a Gabriel application for the task twice: once using OpenWorkflow, and once without it.

Metrics: Three metrics are relevant. The first is the time spent in adding error states to the workflow. This is measured by total time in hours divided by number of error states (*hr/error*). The second metric is the number of lines of code written to create the Gabriel application (*loc*). The third metric is the time in hours (*hr*) spent in writing these lines of code, including the conceptual effort of mapping the task workflow and error states to code.

Results: Table 4 presents our results, broken down into four components. Using OpenWorkflow consumes slightly more time to add error cases. However, that modest increase is more than compensated by the fact that zero effort is needed for the other three steps when using OpenWorkflow. The results thus show a clear savings when using OpenWorkflow.

4.2 | End-to-end user study

To complement the microbenchmarks described in Section 4.1, we conducted a user study of the full Ajalon toolchain. This user study answers Questions 5 and 6 that were posed earlier.

4.2.1 | Protocol

Approach: Our user study asks subjects to write a WCA application under two conditions: with and without Ajalon. The target application should guide a user in making a sandwich from a toy kit, as described earlier. We provide a 58-s

first-person video of the *making sandwich* process to the subject. Some key frames of this video are shown in Figure 16. The application produced by the subject should be capable of handling three test cases illustrated in Figure 20. In the first test case, the tester performs the nonerror steps of sandwich building, as presented in the original video. This process corresponds to the state machine represented by the white boxes in Figure 20. In the second and third test case, the tester performed the the nonerror steps and one error, shown as a red parallelogram in the figure. In the second test case, the error was introduced after the first step: *put a tomato on the bread*. In the third test case, the error step occurred after the second nonerror step: *put a cucumber on the ham*. A functional target application for making sandwiches should provide appropriate instructions to users for all test cases.

Subjects: The subjects were eight students pursuing their Master's degrees in Computer Science or Electrical Engineering at Aalto University in Finland. They were taking part in a course that included three lectures on the concepts and development process for producing a Gabriel application. The subjects had an average of 5.38 years (SD = 2.13) of programming experience. Their proficiency with different programming languages is shown in Table 5. Their knowledge of computer vision and deep learning is shown in Table 6, with values interpreted as follows: (a) "No experience:" subject has no knowledge about the area; (b) "Beginner:" subject has taken some courses related to the area; (c) "Intermediate:" subject has studied or worked in the area for at least 3 years; (d)"Advanced:" subject has studied or worked in the area for at least 5 years. We divided our subjects into two groups, junior and intermediate, with four subjects in each. Intermediate developers had at least 6 years of programming or were experienced with both computer vision and machine learning, while junior ones had less than 6 years of programming experience and 1 year of computer vision or machine learning study.

Methods. The subjects took part in two application development conditions. The first required them to develop the *making sandwich* application by using Ajalon as described above. One week later, the same subjects were asked to develop the same application without using Ajalon. The time gap between the two experiments is expected to reduce any memory bias on the same workflow video. Moreover, any carryover learning from Ajalon should aid performance in the manual version and reduce efficiency differences between the methods. The total time for the user study was 3 weeks, including one week for the Ajalon method, one week for the non-Ajalon method, and one week for the time delay.

All subjects played dual roles of task expert and developer. This is possible because the task is simple: it can be mastered by a 5-min training session. Future work with more complex domains will test teams that combine developers and task experts. The subjects carried out development with and without Ajalon. For the non-Ajalon case, we provided additional

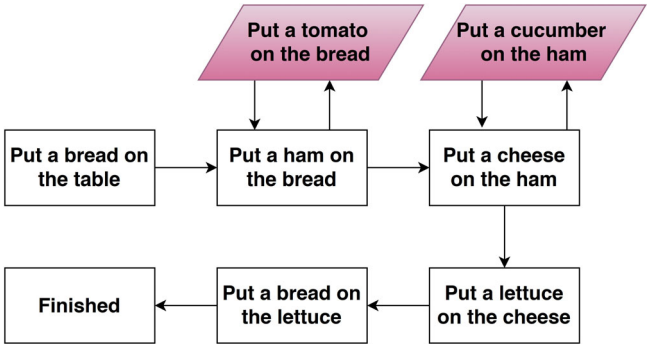


FIGURE 20 Test cases for sandwich application evaluation. Nonerror steps for case 1 are in the white boxes; errors for case 2 and 3 are introduced as shown in pink [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 5 Proficiency of study participants

Programming language	C/C++	Python	Java	Others
Participants proficient	63%	88%	38%	13%

TABLE 6 Participants' backgrounds in computer vision and deep learning

Background level	Computer vision	Deep learning
No experience	25%	37%
Beginner	62%	50 %
Intermediate	13%	13%
Advanced	0 %	0%

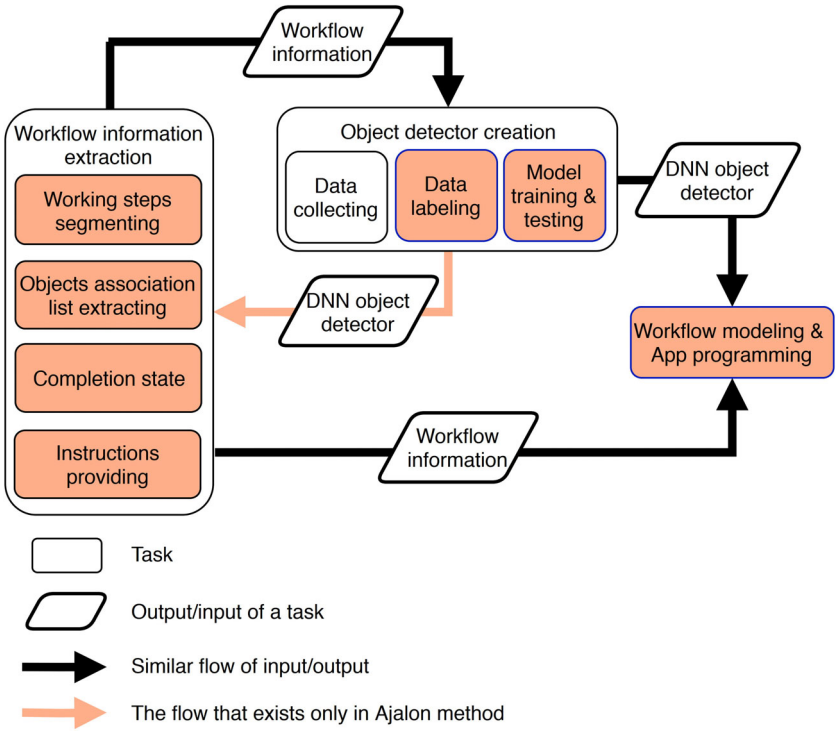


FIGURE 21 The Gabriel application development process with and without Ajalon. Orange boxes represent tasks that Ajalon offers a specific tool for [Colour figure can be viewed at wileyonlinelibrary.com]

tools for data labeling[27,28] and DNN training.[3,18,29] Figure 21 shows how the four major development steps of the Gabriel development process differ in the two cases. Workflow extraction in the Ajalon case is done using PT and PTEditor. In the non-Ajalon case, this is done using a video player. Creation of object detectors in the Ajalon case is done using OpenTPOD. In the non-Ajalon case, this is done using a manual setup of data labeling tools, training, and testing. The implementation of the FSM for the Gabriel application, including additions of error states, is done via OpenWorkflow in the Ajalon case. Since code is generated by OpenWorkflow, no manual coding is necessary. In the non-Ajalon case, the representation of the FSM is created using a UML drawing tool, and the code for the FSM is written manually.

Metrics: As a basis for efficiency measures, subjects were required to create working logs recording the time spent on their specific steps of developing the application. A quantitative measure of savings is provided by the difference in time for the two conditions, relative to the with-Ajalon time baseline.

$$\text{savings factor} = \frac{\text{Non-Ajalon time} - \text{Ajalon time}}{\text{Ajalon time}}. \tag{2}$$

Qualitative measures of the two conditions were obtained from a postexperiment survey questionnaire that subjects filled out upon completion of the study. This questionnaire rated the usefulness, ease, and enjoyment of each component of Ajalon on a discrete numeric scale (1 = strongly disagree and 7 = strongly agree). PT, which was fully automated, was not rated. The survey questionnaire was designed based on the guidance of Brooke et al.[30] and Gordillo et al.[31]

4.2.2 | Qualitative results

Figure 22 shows the results of the ratings for each component of Ajalon. Mean ratings ranged from 4 to 6 on the 7-point scale, indicating fairly high levels of usefulness (mean: 5.1) and ease of use (mean: 5.5), and moderately high enjoyment (mean: 4.2). Ratings were similar across all components. On the whole, these results indicate that subjects perceived benefit from using Ajalon in developing the application.

Some insight into the enjoyment scores being relatively low can be gained from feedback given in the survey. Subjects found the interface components, such as buttons and scroll bars, somewhat annoying to use. They also experienced some problems with the tracking algorithm of OpenTPOD during labeling. These comments underscore the importance of user experience design for Ajalon, as the enjoyment during use affects the performance of developing an application.

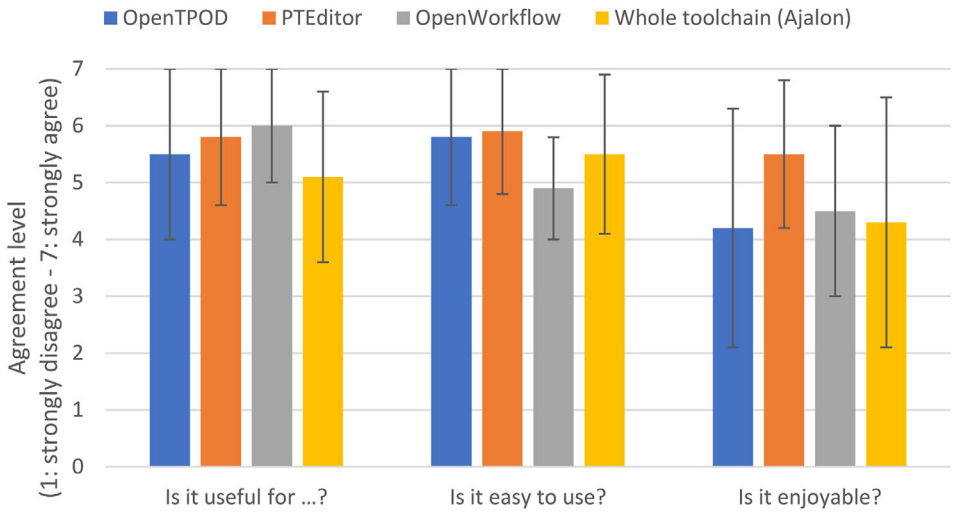


FIGURE 22 Rated user experience of using Ajalon; the scales ranged from 1 to 7 [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 7 Time spent in component steps and the whole process of developing a Gabriel application for sandwich making, by junior and intermediate subjects when working with and without the aid of Ajalon. With Ajalon, OpenTPOD is used for creating the DNN-based object detectors, PT and PTEditor are used for workflow extraction, and OpenWorkflow is used for workflow modeling. Without Ajalon, the subjects can freely select their desired tools to achieve the required task

Mean time (SD) in hours		**Junior**		**Intermediate**	
Task	**Subtask**	**Without Ajalon**	**With Ajalon**	**Without Ajalon**	**With Ajalon**
	Data labeling	7.63 (1.6)	3.25 (0.96)	5.56 (0.52)	3.13 (0.95)
Object detector	Coding for training	3.84(0.74)	0	0.35 (0.10)	0
creation	Coding for testing	0.88 (0.32)	0	0.17 (0.07)	0
Workflow extraction		0.79 (0.17)	0.35 (0.10)	1.08 (0.24)	0.42 (0.10)
Workflow modeling		0.75 (0.17)	0.83 (0.53)	0.90 (0.24)	1.88 (1.11)
Other aspects		15.54 (0.95)	0	7.98 (4.18)	0
All tasks		29.41(2.20)	4.44 (1.21)	15.91(4.78)	5.42 (1.40)

4.2.3 | Quantitative results

The subjects' logs of time spent in different stages of developing the application with and without Ajalon revealed clear advantages for using Ajalon. Means by stage of application development for each of the two subject groups, working with and without Ajalon, are shown in Table 7.

Overall, subjects spent 22.7 h in development without the aid of Ajalon, as compared to 4.9 h with it. This *savings factor* amounts to 3.6 overall and is greater for junior-level subjects (5.6) than intermediates (1.9). Another implication of the table is that Ajalon aid leveled the playing field for the two groups, bringing the average time for juniors and intermediates to a few hours each. Figure 23 shows the savings factor for individual subjects in each component of application development and Figure 24 for the application as a whole.

Total application time: Although the statistical power is limited by the small number of subjects, we conducted a mixed-model ANOVA on the between-subject factor of expertise level (junior, intermediate) and the within-subject factor, presence/absence of Ajalon.[32] The ANOVA indicates whether each factor has an effect, averaged over the other (the main effects), and whether there are mutual dependencies (indicated by an interaction). For the overall application time, both main effects were significant, showing the expected advantage for greater expertise, ($F_{1,6} = 15.90$, $p = 0.007$), and for the presence of Ajalon ($F_{1,6} = 221.51$, $p < 0.001$). In addition, there was a significant interaction term ($F_{1,6} = 36.92$, $p = 0.001$), reflecting the finding that subjects at the junior level of expertise had a greater overall advantage from using Ajalon. This is evident from the individual-subject data in Figure 23, where all four junior subjects show greater savings than any intermediate.

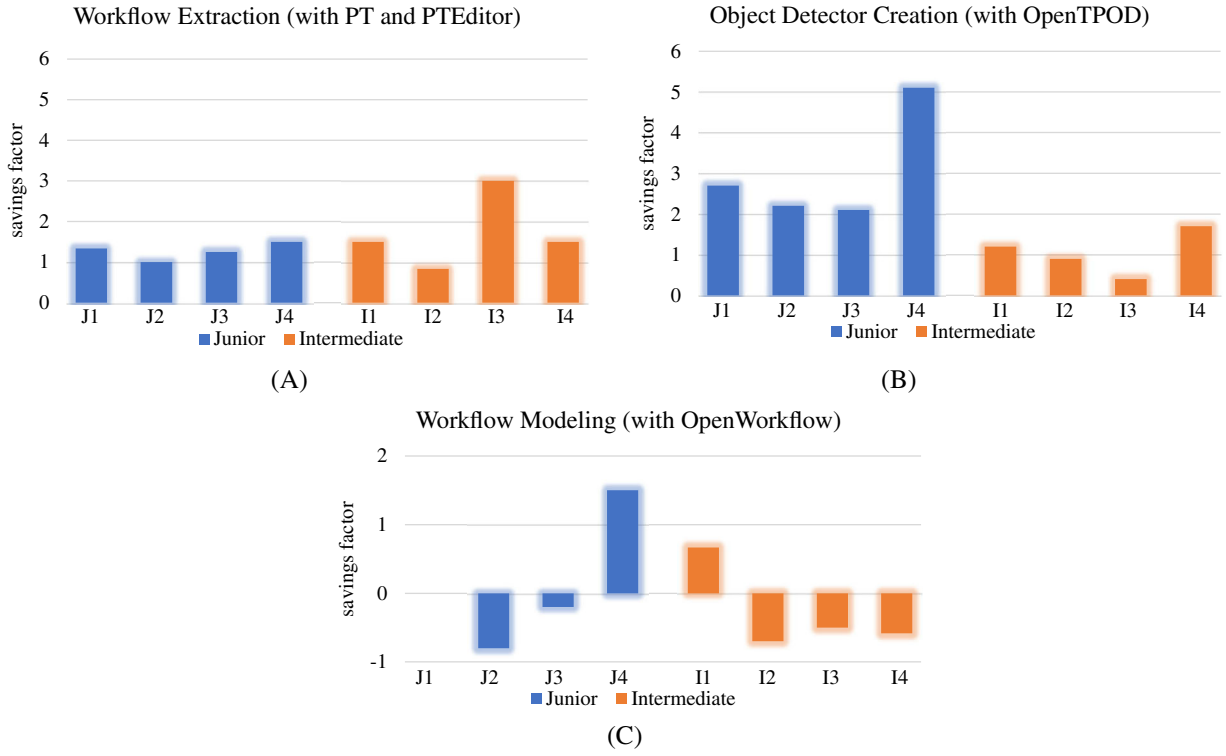


FIGURE 23 Breakdown of the *savings factor* for each of the three steps, by junior and intermediate subjects. (A) Workflow extraction with PT and PTEditor. (B) Object detector creation with OpenTPOD. (C) Workflow modeling with OpenWorkflow [Colour figure can be viewed at wileyonlinelibrary.com]

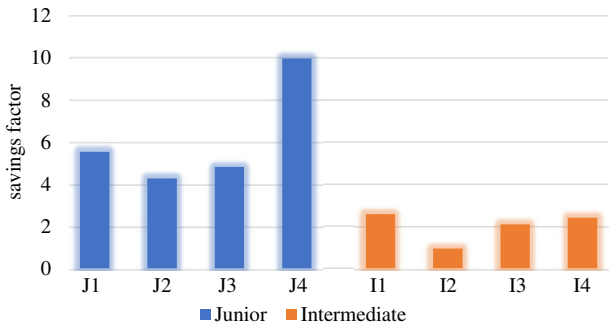


FIGURE 24 Savings factor (see Equation (2)) across all tools for Ajalon by junior and intermediate subjects. The higher the savings factor, the greater the time-cost of not using Ajalon tools [Colour figure can be viewed at wileyonlinelibrary.com]

We next assess the contributions of the various components of Ajalon, using the same ANOVA approach on the times for each component. Only statistically significant F-tests on the presence/absence of Ajalon will be reported below.

Data labeling from OpenTPOD: Essentially, OpenTPOD halved the time for the data-labeling step in building the expected-object detectors. As shown in Figure 23, all subjects showed a positive savings for the object detector creation, resulting in a significant effect of using Ajalon ($F_{1,6} = 69.75$, $p < 0.001$). The magnitude of the savings, 1.06 overall, was similar for the two subject groups. In comments, four of six subjects pointed to the benefits of using OpenTPOD to create object detectors, which reduced the amount of time spent on tasks like data annotation, converting data to different file formats, and complicated installation steps.

Workflow extraction using the PTEditor: Note that workflow extraction with Ajalon uses PT and PTEditor, but as the former is fully automatic, the time essentially measures the use of PTEditor. Although the average *savings factor* for this step (1.43) is comparable to that of the data labeling component of building object detectors with OpenTPOD, the absolute time benefit of using Ajalon to extract workflow information is small. On average, subjects spent 0.9 h without PTEditor while spending 0.4 h with it, a significant difference, ($F_{1,6} = 64.374$, $p < 0.001$). Three of six subjects reported that PTEditor helped them determine what information was needed.

Workflow modeling from OpenWorkflow: This component is used to include error cases when describing and modeling workflow. In contrast to the other tools of Ajalon, OpenWorkflow did not reduce the time spent on this task. On average, subjects spent 0.8 h for describing the error cases and modeling the workflow without OpenWorkflow, as compared to 1.4 hours with it. As illustrated in Figure 23, only two out of six subjects benefitted from OpenWorkflow. Feedback from subjects suggested two main reasons that lead to the results. The first is that the interface of OpenWorkflow is not easy to use, particularly because the visualized state machine window is not large enough to manipulate the workflow for adding error states. The second reason is that adding error cases is complicated because subjects must report processing functions and conditions in each node. Subjects suggested that the tool should provide a *copy and paste* function to reduce the manual effort required to add text. Although OpenWorkflow has low points in enjoyment and ease, its output, which is the modeled workflow, allows the automatic code generator to build a Gabriel application in 1 s.

Other aspects: This component refers to programming that is required to complete developing a Gabriel application with the modeled workflow and qualified object detectors. Because the OpenWorkflow of Ajalon appropriately models workflow, OpenWorkflow can directly generate the target Gabriel application. In contrast, programming without OpenWorkflow requires several hours, as reported in other implementation task of Table 7. Specifically, Ajalon saved approximately 20.3 h for junior subjects and 8.5 h for intermediate ones. The reduction in manual effort from developers and experts in building the required Gabriel application is an affirmative answer for questions 5 and 6 above.

5 | CONCLUSION AND FUTURE WORK

WCA has gained considerable visibility in a short time, and is likely to be transformative in education, health care, industrial troubleshooting, manufacturing, assisted driving, and sports training. An important enabler for widespread adoption of WCA will be the lowering of barriers to creating applications of this genre. Today, WCA development is much more difficult and slower than, for example, web development. Further, it requires skills in areas such as machine learning and computer vision that are not widespread among software developers. Lowering this barrier is an important step toward democratizing the creation of WCA applications.

To this end, we have created Ajalon, an authoring toolchain for WCA applications that reduces the skill and effort needed at each step of the development pipeline. Starting from a video of an expert performing a task, Ajalon automates the extraction of working steps. It enables the creation of DNN-based object detectors and an FSM that models the task, without writing any code. Our evaluation shows that Ajalon is effective in significantly reducing the barrier to creating new WCA applications.

While Ajalon is clearly a helpful and effective toolchain, challenges remain. Our user study has given us valuable insights into areas for improvement. First, the GUIs of OpenTPOD and OpenWorkflow can be improved. In OpenTPOD, subjects wanted an estimate of the time remaining until models would be fully trained. In OpenWorkflow, nearly half the subjects expressed a desire to interact with the editor using hotkeys and a command line. In addition to these improvements, we are also considering the use of voice commands rather than mouse-and-keyboard interaction.

The format of workflow information in Ajalon could be improved along the lines suggested by Jelodar et al.,[33] who propose a more comprehensive method that describes both activities and object association lists. They found that describing activities leads to a boost in the accuracy of modeling a cooking activity. We plan to incorporate activity recognition into Ajalon, in order to automate more of the process of generating WCA applications.

The size of our user study was limited, with only eight subjects. A principal reason for the small number is the amount of time required for participation. As Ajalon improves and matures, we anticipate that it will be easier to recruit users. This will enable us to run user studies that are larger, ideally closer to the 15–20 subjects that are typical for a user study of this kind.[34,35]

Our user study involved a task (building a sandwich) that almost all subjects were familiar with. This enabled them to play the dual roles of task expert and developer. In the future, we plan to run studies involving more complex and unfamiliar tasks, where the roles of a task expert and a developer are distinct.

ACKNOWLEDGEMENTS

We thank the editor, Dr. Satish Srirama, and the anonymous reviewers for their guidance in improving the presentation of our work. This research was supported by Business Finland under the grant number 1660/31/2018, and by the National Science Foundation (NSF) under grant number CNS-1518865. Roger Iyengar was supported by an NSF Graduate Research Fellowship under Grants DGE1252522 and DGE1745016. Additional support was provided by CableLabs,

Crown Castle, Deutsche Telekom, Intel, InterDigital, Microsoft, Seagate, VMware, Vodafone, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

ORCID


Truong An Pham https://orcid.org/0000-0002-6861-1495

REFERENCES

1. Ha K, Chen Z, Hu W, Richter W, Pillai P, Satyanarayanan M. Towards wearable cognitive assistance. *MobiSys '14*. New York, NY: Association for Computing Machinery; 2014:68-81.
2. Ray T. An angel on your shoulder: who will build A.I.? Barron's; 2018.
3. Ren S, He K, Girshick R, Sun J. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans Pattern Anal Mach Intell*. 2016;39(6):1137-1149.
4. Satyanarayanan M. The emergence of edge computing. *IEEE Comput*. 2017;50(1):30–39.
5. Satyanarayanan M, Bahl P, Caceres R, Davies N. The case for VM-based cloudlets in mobile computing. *IEEE Pervas Comput*. 2009;8(4):14–23.
6. Satyanarayanan M, Davies N. Augmenting cognition through edge computing. *IEEE Comput*. 2019;52(7):37–46.
7. Chen Z, Hu W, Wang J, et al. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance; 2017:14, ACM.
8. Yampolskiy RV. *AI-Complete, AI-Hard, or AI-Easy: Classification of Problems in Artificial Intelligence. Technical Report No. 2*. Louisville, KY: University of Louisville; 2011.
9. Gabriel: platform for wearable cognitive assistance applications; 2018. https://github.com/cmusatyalab/gabriel.
10. Rosenberg D, Boehm BW, Wang B, Qi K. The parallel agile process: applying parallel processing techniques to software engineering. *J Softw Evolut Process*. 2019;31(6):e2144.
11. Minor M, Bergmann R, Görg S, Walter K. Adaptation of cooking instructions following the workflow paradigm; 2010:199-208.
12. Mura K, Petersen N, Huff M, Ghose T. IBES: a tool for creating instructions based on event segmentation. *Front Psychol*. 2013;4:994.
13. Kim J, Nguyen PT, Weir S, Guo PJ, Miller RC, Gajos KZ. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. *CHI '14*. New York, NY: Association for Computing Machinery; 2014:4017-4026.
14. Nater F, Grabner H, Van Gool L. Unsupervised workflow discovery in industrial environments; 2011:1912-1919.
15. Carmines EG, Zeller RA. *Reliability and Validity Assessment*. Los Angeles, CA: SAGE Publications; 1979:11–12.
16. Pham TA, Xiao Y. Unsupervised workflow extraction from first-person video of mechanical assembly; 2018:31-36; ACM.
17. Bertasius G, Park HS, Yu SX, Shi J. First-person action-object detection with EgoNet. *Robotics: Science and Systems*. Cambridge, MA: MIT Press; 2017.
18. Redmon J, Farhadi A. Yolov3: an incremental improvement; 2018. arXiv preprint arXiv:1804.02767
19. Jalal A, Kim Y-H, Kim Y-J, Kamal S, Kim D. Robust human activity recognition from depth video using spatiotemporal multi-fused features. *Pattern Recogn*. 2017;61:295-308.
20. Everingham M, Eslami SMA, Van Gool L, Williams CKI, Winn J, Zisserman A. The pascal visual object classes challenge: a retrospective. *Int J Comput Vis*. 2015;111(1):98-136.
21. Russakovsky O, Deng J, Su H, et al. Imagenet large scale visual recognition challenge. *Int J Comput Vis*. 2015;115(3):211-252.
22. Li X, Grandvalet Y, Davoine F, et al. Transfer learning in computer vision tasks: remember where you come from. *Image Vis Comput*. 2020;93:103853.
23. Zhang S, Yao L, Sun A, Tay Y. Deep learning based recommender system: a survey and new perspectives. *ACM Comput Surv*. 2019;52(1):1-38.
24. Danelljan M, Häger G, Khan FS, Felsberg M. *Accurate Scale Estimation for Robust Visual Tracking*. Nottingham, UK: BMVA Press; 2014.
25. Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. *J Big Data*. 2019;6(1):60.
26. Wang J, Xu C, Chng E, Duan L, Wan K, Qi T. Automatic generation of personalized music sports video. *MULTIMEDIA '05*. New York, NY: ACM; 2005:735-744.
27. Computer vision annotation tool (CVAT); 2018. https://github.com/openvinotoolkit/cvat. Accessed October 1, 2019.
28. Russell BC, Torralba A, Murphy KP, Freeman WT. LabelMe: a database and web-based tool for image annotation. *Int J Comput Vis*. 2008;77(1):157-173.
29. Lin TY, Goyal P, Girshick R, He K, Dollár P. Focal loss for dense object detection; 2017:2980-2988
30. Brooke J. SUS-A quick and dirty usability scale. *Usability Evaluation in Industry*. Boca Raton, FL: CRC Press; 1996.
31. Gordillo A, Barra E, Quemada J. An easy to use open source authoring tool to create effective and reusable learning objects. *Comput Appl Eng Educ*. 2017;25(2):188-199.
32. Gagnon JC, Barber BR. *The SAGE Encyclopedia of Educational Research, Measurement, and Evaluation*. Los Angeles, CA: Sage; 2018:668.
33. Jelodar AB, Paulius D, Sun Y. Long activity video understanding using functional object-oriented network. *IEEE Trans Multimed*. 2019;21(7):1813-1824.

34. Kim Y, Choi Y, Lee H, Lee G, Bianchi A. VirtualComponent: a mixed-reality tool for designing and tuning breadboarded circuits. *CHI '19*. New York, NY: ACM; 2019:177:1-177:13.
35. Lo J-Y, Huang D-Y, Kuo T-S, et al. AutoFritz: autocomplete for prototyping virtual breadboard circuits. *CHI '19*. New York, NY: ACM; 2019:403:1-403:13.

How to cite this article: Pham TA, Wang J, Iyengar R, et al. Ajalon: Simplifying the authoring of wearable cognitive assistants. *Softw: Pract Exper*. 2021;51:1773–1797. https://doi.org/10.1002/spe.2987