

# Apollo - An Immersive Approach to Web Tutorials

Connor Hamlet  
University of North Carolina at Chapel Hill  
cphamlet@cs.unc.edu

## ABSTRACT

In this paper, we demonstrate the use of a chrome extension which allows instructors and students to create and view tutorials that are embedded into the task at hand using a Chrome extension. Our tutorial creation software, *Apollo*, only works for web tutorials, but highlights the important use case for embedding the tutorial into the task at hand, an important distinction from traditional tutorials made with PDFs and videos. We dive into the specifics of the tool and walk through several scenarios demonstrating its utility.

## 1 INTRODUCTION

As of 2017, there are an estimated 1 billion websites [1] which offer a variety of services. The inherent challenge with introducing many of these web based services is teaching foreign or new concepts. Given the widespread use and exponentially growing number of website services, this encourages the development of easy to follow, scalable tutorials. Distributing tutorials and instruction has remained an inherently tedious task in the Internet age. Current methods involve either meticulously taking screenshots and collating an annotated pdf or the use of desktop recording software to share a video. These methods, while common standard, are time consuming for large tutorials and need to be remade in case a part of the tutorial changes. Most importantly, these methods divert the user's attention away from the task at hand through continuous context switching between task and tutorial.

The amount of structured data that is present on the web, specifically HTML led us to develop a chrome extension to easily create, navigate, highlight, and annotate steps for the web. Named *Apollo*, our tool embeds tutorials into the task at hand.

For clarity, we denote a subtasks of a tutorial as a *recording*. Recordings are created by tutorial creators and are defined as a series of annotated steps over a series of webpages. Together, these steps make up the entire tutorial. Recordings do not capture mouse movements and are not timed, but instead capture instructor denoted sequences and can span multiple websites. The instructor is able to create a step by first selecting and then annotating an HTML element.

## CCS CONCEPTS

• Computer systems organization → Web Development

## ADDITIONAL KEYWORDS AND PHRASES

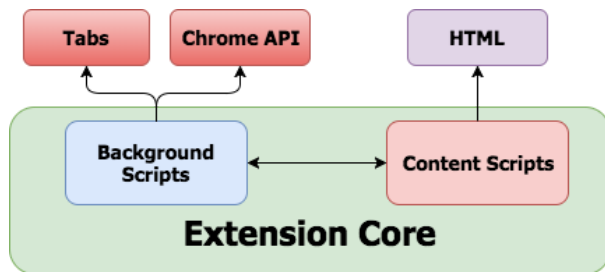
Web tutorials, scalable learning, distributed learning, web navigation

## 2 CONTENT SCRIPTS VS BACKGROUND SCRIPTS

### *What is a Chrome Extension?*

Chrome extensions are pieces software designed to enhance the Chrome web browser. They add additional functionality such as adding and changing content on webpages, or offering a personalized experience to the end user. For example, the popular AdBlock™ extension actively targets and prevents the loading of web advertisements. Other ones such as Grammarly™ aim to provide corrections to a user's grammar and spelling in text forms, preventing mistakes. All of the programming in these extensions use JavaScript in order to manipulate the page and communicate with remote servers.

Chrome Extensions consist of two main sections, content script and background scripts. Content scripts are JavaScript files that have direct access to the HTML webpage. They have the ability to manipulate, save, and analyze any HTML. Their memory and process state is ephemeral to the specific page the user is currently viewing. Thus when a user changes the page the content script is started again and all variables are cleared. Secondly, there is one content script instance per tab that is open in the user's browser. If you want data that is persistent across pages or tabs, you need what is known as a background script. Unlike content scripts, there is only one background script per browser instance. Google Chrome allows communicating between the background script and content scripts using the Message Passing API. This allows data in JSON format to be shared between both processes.



**Figure 1.** Content scripts and background scripts communicating with each other. Content scripts manipulate the HTML and background script make calls to the browser API.

Background scripts also allow developers to access deeper and more technical Chrome API calls, such as utilizing and processing a user's browser history. Lastly, there is the manifest.json file, which is responsible for listing out what permissions your app requires, how many content scripts you have, and all of the other required JavaScript files that are necessary to run the extension.

### 3 SCENARIOS

Next we will look at two separate case studies to demonstrate the utility of our tool. We explore the following:

1. Jon is a faculty member looking to distribute an easy to follow and highly scalable tutorial for launching a virtual machine using the

Atmosphere cloud platform for 300 students. In this case study we show how fitted our tool is for a complex task such as launching a virtual machine.

2. Karen is a library manager who wants to create a tutorial on how to use the UNC library online portal to check out a book.

Atmosphere is a service provided by Cyverse for launching virtual machines for data analysis and research purposes. In our case, Jon's objective is to:

1. Have all 300 students register for an account on Cyverse.
2. After registering, have the students launch a minimal resource Ubuntu image.
3. Create a scalable and easy to follow tutorial which enables his students to accomplish this goal.

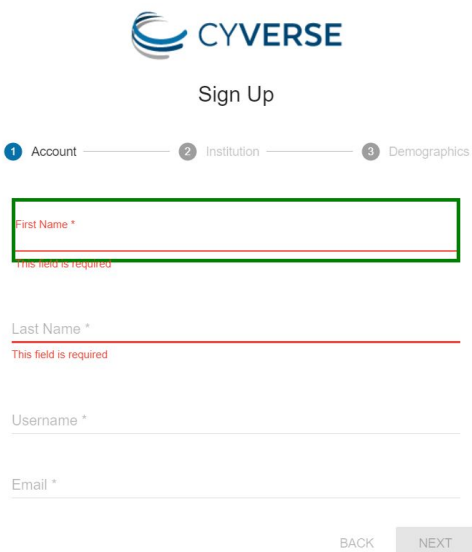
Jon can begin recording the tutorial by navigating to the start page and can begin by selecting specific HTML elements and annotating them. For example, in this tutorial registering for a Cyverse account is the first step. Therefore, Jon navigates to the registration page, selecting and saving the "Sign up" step.



**Figure 2.** The first step in Jon's recording. Note that the link is bordered with a green color, indicating where the student should click.

Now that Jon has selected his first item, he annotates it by providing a text description. When the tutorial is replayed, the HTML element Jon selected will be highlighted, and the text description he wrote will appear before the user. Jon proceeds through the steps, highlighting areas which

require user interaction or areas of importance. In the meantime, our extension is monitoring for this activity and is storing it locally.



**CYVERSE**

Sign Up

1 Account — 2 Institution — 3 Demographics

First Name \*

This field is required

Last Name \*

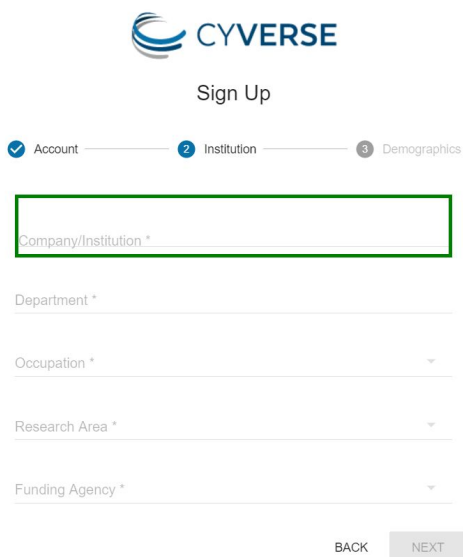
This field is required

Username \*

Email \*

BACK NEXT

**Figure 3.** Jon selects the demographic area to indicate where the student should type.



**CYVERSE**

Sign Up

✓ Account — 2 Institution — 3 Demographics

Company/Institution \*

Department \*

Occupation \*

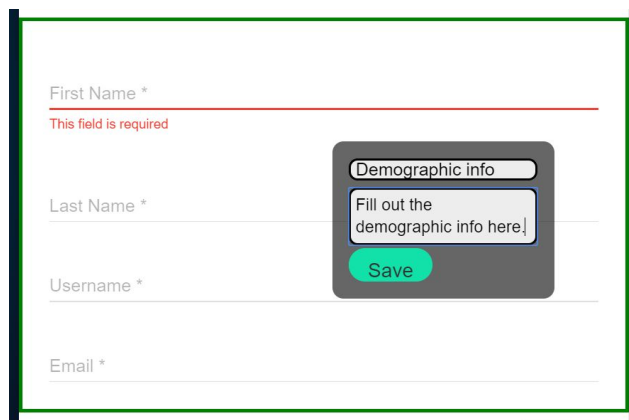
Research Area \*

Funding Agency \*

BACK NEXT

**Figure 4.** Any HTML element can be selected, from buttons, to textboxes, to entire webpages.

When Jon selects an HTML element, a draggable textbox will overlay on top of the webpage which allows him to type in detailed instructional information for the student. When this tutorial is replayed, the student will see Jon's description next to the selected element, minus the save button.



First Name \*

This field is required

Last Name \*

Username \*

Email \*

Demographic info

Fill out the demographic info here

Save

**Figure 5.** Jon's fourth step in his recording. Note the annotation he gives to describe the steps the student should take.

After registering an account, Jon follows the steps to launch a virtual machine by selecting the search bar for available images.

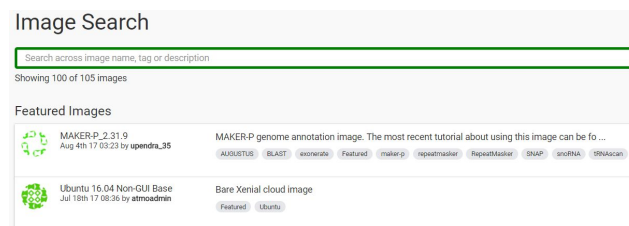


Image Search

Search across image name, tag or description

Showing 100 of 105 images

Featured Images

MAKER-P 2.31.9  
Aug 4th 17 09:23 by upendra\_35

MAKER-P genome annotation image. The most recent tutorial about using this image can be fo ...

ALDIUSIS BLAST ecorelate (Featured) maker-p repeatmaker RepeatMasker SNAP snRNA SRNAcan

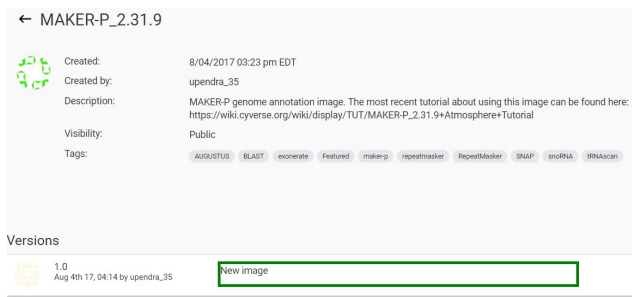
Ubuntu 16.04 Non-GUI Base  
Jul 18th 17 08:36 by atmoadmin

Bare Xenial cloud image

(Featured) Ubuntu

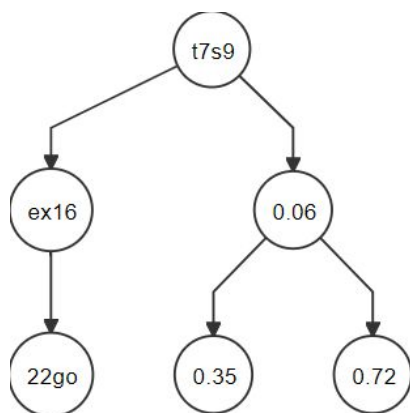
**Figure 6.** A highlighted search bar which Jon selected.

Typing in an image type then presents the student with more detailed information, giving them an option to launch the machine.



**Figure 7.** As this step, the student has the ability to launch a virtual machine.

After finishing the recording, Jon saves it by opening the extension menu and clicking “End Recording”. He then is presented with a visualization of his tutorial as a Directed Acyclic Graph (DAG), and has the option to create branches at different steps. This will present the student taking the tutorial with the option to determine what their destination will be.

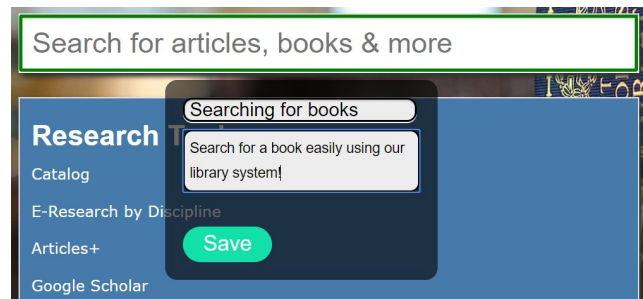


**Figure 8.** Our DAG view which gives the instructor the ability to make links between different steps

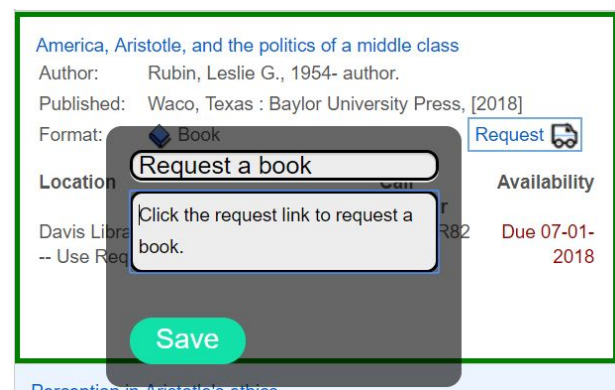
As we can see from the scenario, creating this tutorial does not involve taking a large amount screenshots, nor dealing with desktop recording software and its voice annotation. Moreover, Jon does not need to spend an excessive amount of time editing a video, or formatting a document. In our case, creating the tutorial requires

completing the task at hand along with providing text descriptions.

The task need not be complicated either. Let’s take a look at our second scenario, where Karen is building a tutorial titled “How to Checkout a Book at the Davis Library”.



**Figure 9.** Karen describes using the search bar to search for “Aristotle’s early works”



**Figure 10.** You may request a book through any of the “Request a Book” links.

**New and Returning Users**

Onyen/Username

Password

Logon

UNC Affiliates without an Onyen

Document Delivery

Interlibrary Loan FAQ

Login

First you will need to login

Save

**Figure 11.** Authentication is required to check out a book from Davis.

Interlibrary Borrowing and Campus Delivery Services

Loan Request Received: Transaction Number 2809790

**Request non-UNC Material**  
Book (e-book, microfilm, etc.)  
Article (book chapter, conference paper, etc.)  
Dissertation/Thesis

**Request UNC Material**  
UNC Book for Delivery (including Recall & Storage Requests)  
UNC Article

**Retrieve**  
Electronically Received Article(s)

**Renew**  
Interlibrary Loan Material  
UNC Library Books via MyLibrary  
TRLN Direct Books via MyLibrary

**View**  
Outstanding Requests  
Cancelled Requests  
Finished Requests  
All Requests  
Notifications

Outstanding Requests	
Request Title	
2809790	Seeds of life : from Aristotle to Da Vinci, from shark's teeth to frog's pants, the long and strange quest to discover where babies come from

Interlibrary Borrowing: email or 919-962-1326  
Carolina BLU: email or 919-962-1053  
Suggestions on Library Services? Give us your feedback.

Copyright © 2016 Atlas Systems, Inc. All Rights Reserved.

**Figure 12.** The confirmation page, showing that the request for a library book succeeded.

When this tutorial is accessed by the student, it will lead the student through Jon and Karen's steps and display their descriptions.

We denote the functionality of Apollo which creates the tutorial as the *Instructor View*. The functionality which allows the student to view the tutorial is the *Student View*. Next we describe the implementation details of both.

## 4 THE STUDENT VIEW

From a student's perspective, the student first needs to install the chrome extension. Once this is installed, they need to obtain the instructors recording of a tutorial.

An important part of these recordings is in their distribution to students. This begs the following questions:

1. What is the data format that the tutorials are stored in?
2. How can the students obtain the recording?

Given that the recording is a series of actions in a web browser, we can store all selected elements and text descriptions as an ordered JSON text file. This significantly reduces the size of tutorial compared to a video. This also allows for great data compression ratios.

As for obtaining the tutorial, instructors can host their custom tutorial JSON file themselves or by uploading it to a tutorial distribution service. Having a central server as a distribution method would allow additional features, such as searching and rating.

Once a user has properly authenticated themselves, we proceed to show them a search bar where they can search for public tutorials, view popular ones, or be allowed access to a private instructors tutorial. There are several other options available including to start a recording, save one, or load a new one.

## Web Tutorials

### Student View

Load Record

### Instructor View

New Recording

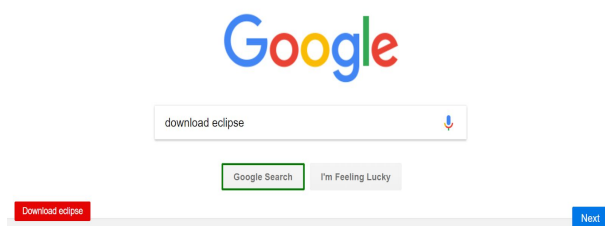
Clear recording

Edit/Load

**Figure 13.** *An authenticated user session*

Once the user has selected the tutorial they would like to view, they may click on “Load Record” and begin the tutorial. First they are automatically navigated to the first URL of the tutorial and a green box highlights the important information that and they are presented with along with an annotated text description. A “Next” button also appears on the bottom right. When the user is ready to continue to the next step, they may click the next button, which navigates them to the next step. Currently, this version does not support going in the reverse direction or jumping in the middle, but there will be support for doing so in the future.

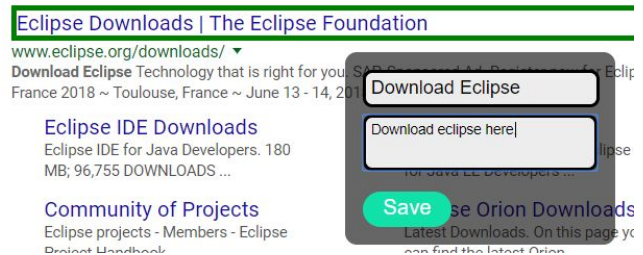
Let’s take a simple example where an instructor would like to teach his or her students how to download Eclipse. The first node in this case will be Googling “Download Eclipse”. A green highlighted box appears over the google search button, indicating where the user should click.



**Figure 14.** *The first node in the tutorial: “Downloading Eclipse”. Note the blue button in the bottom right corner*

indicating “Next” and the associated message on the bottom left.

Once user has clicked on the next button and has navigated to the next page, our Chrome Extension will automatically execute a string matching algorithm to locate the next portion of the HTML page to highlight the next instructor indicated DOM element, which indicates the next step in the tutorial. This process repeats until the last node in the tutorial has been reached.



**Figure 15.** *The dialogue box that appears when selecting an HTML Element.*

## 5 THE INSTRUCTOR VIEW

Users can create a recording of a tutorial when they enter what we call the *Instructor View*. When the instructor view is entered, actions and page navigations can start to be recorded.

As shown in the scenario above, the workflow of creating a recording is as follows:

1. The instructor navigates to a page and selects an HTML element on the page to draw user attention to.
2. The element is then highlighted by a green box and a draggable text box with a “Save” button appears.
3. The instructor may then type in some text describing what the student should know in this part of the tutorial and click “Save”.
4. The highlighted portion of the HTML element disappears and is then saved to the background



script. The instructor can again complete Steps 1-4 until the entire tutorial have been completed.

5. The instructor clicks on the extension icon in the upper right corner and clicks “Save Recording”.

Once a workflow has been saved, the tutorial is now ready for use by students. With a saved tutorial in hand, the instructor may choose to either copy the JSON file to their own website, or host it centrally on our server using an upload feature through the extension.

## 6 TECHNICAL IMPLEMENTATION

Our product works by implementing three separate scripts.

1. A background script which maintains the internal state of the recording and receives saved annotations from the content script.
2. A content script which manipulates the user’s webpage by adding highlighted annotations.
3. A popup script which is responsible for displaying the user authentication UI and the “Load Recording” and “Save Recording” buttons.

Earlier we discussed that when the user clicks on the “Next” button, the next step is loaded by navigating to the next URL in the list and highlighting the specified HTML element. In order to maintain this data persistently and to not have the page cleared on every load, this necessitated the use of a background script. We also incorporate plenty of HTML manipulation through the use of displaying annotated text. In order to save these recordings and maintain their persistence, this extension relies heavily on communication between the background script and content script.

For example, when an instructor selects an HTML object on a page, all of the logic for highlighting and searching occurs in the content script. After a particular node has been saved, our content script will then send this section of the recording to the background script through the `chrome.runtime.sendMessage` call in a JSON format.

```
//Handles the click function for newly created save button
$(save_button).click(function(){
  //when you click the save button, this reverts the green block of text back to normal.
  if(unborderedElementPointerHTML!=null){
    //This replace with function, removes the element with the green border if one already
    $(elementOnMouseOver).replaceWith($(
      unborderedElementPointerHTML).prop('outerHTML'));
  }
  chrome.runtime.sendMessage({command: "send", element_html:unborderedElementPointerHTML,
    entered_text:$(editable_text).text(), url>window.location["href"]},
    function(response){
      // alert(response.msg + " : " + response.enteredText + " : " + window.location["href"]);
    });
  created_element.remove();
});
```

**Figure 16.** *The send message call which sends the instructor selected HTML block and instructor message to the background script.*

Simultaneously, the background script is running a `chrome.runtime.onMessage` listener, which is listening for a message from the content script in an event loop. When this message is received in JSON, our background script will dynamically add it to a queue that it maintains, which represents the entire recording from start to finish. This queue will maintain the URL of the page, HTML of the instructor selected element, the text description, and the order which the elements were placed.

One important item to note about tutorials is that not all tutorials are a single linear sequence of steps. For this reason it was important to us that we store the tutorial as a DAG which supports a number of features such as merging, and branching. With a DAG, a student has the option to *choose* what path they will take when they continue the tutorial. For example, the outcome of a tutorial might depend whether or not a user has a Windows or Linux computer. This is important for OS specific tutorial, such as downloading different versions of software.

Our background script provides the following API services:

1. Peek. This command will return the front of the queue which holds the next step.
2. Get Next. This command takes in as input what path a user would like to take outputs the DAG out in JSON form, sending it to the content script and sending the user to the previous step.
3. Get Prev. This command takes in no input, and reverts back a step, sending the user to the previous step.

4. Save. This command sends a copy of the DAG from the content script to the background script to save persistently.
5. Get Options: This command will output all of the outgoing edges from the current node. This is important for presenting to the user choices on which path to take.
6. Record Action: This command inputs a “node” and inserts it into the DAG. This occurs when an instructor saves a step. A step consists of the page url, the html they selected, a title, and the caption for the step.
7. Clear: This command will delete the DAG.

If our background script detects an unrecognized command, it will ignore it.

```
chrome.runtime.onMessage.addListener(
function(request, sender, sendResponse) {
  console.log(sender.tab ?
    "from a content script:" + sender.tab.url :
    "from the extension");
  if (request.command === "peek"){
    sendResponse({msg: "Background: sending element from background script"})
  }else if(request.command === "shift"){
    sendResponse({msg: "Background: sending element from background script,
  }else if (request.command === "send"){
    items.objs.push({"element_html": request.element_html, "entered_text":
    console.log(request.element_html);
    sendResponse({msg: "Background: Message received", enteredText: request.
  }else if (request.command === "save"){
    sendResponse(JSON.stringify(items));
  },
  ...
);
```

**Figure 17.** Our background script listener implementation highlighting the commands it accepts.

For storing the tutorial, we use a JSON formatted as follows:

```
{
  DAG: { [graphlib object] },
  current_node_id: ...,
  root_node_id: ...,
  tutorial_name:
}
```

For storing the DAG we use *graphlib* [2], an open-source JavaScript Graph Library and *DagreD3* [3] for graph

visualization. The graph view gives instructors a visual way to make edits, as well as add and delete steps. Creating links between nodes are accomplished by clicking on a node and dragging it to another to connect them. This gives users multiple choices when stepping through the tutorial. .

## 7 CHALLENGES FACED

The biggest overall challenge on this project was persevering against the troubles imposed by learning the Chrome Extension platform. At first, it did not seem intuitive that content scripts and background scripts exist in isolated environments with only the message passing API as the available communication method. Before we moved to a background and content script combination, all recording functionality originally existed as a content script. This presented a few now obvious problems because when the next page was loaded, the recording kept automatically resetting and all ephemeral data was lost. Implementing a background script allowed data to be stored between pages and upon refreshes.

Another main issue which took a few weeks to solve was obtaining the instructor selected HTML from the recording and using it to parse the student view HTML page and find a match. This proved to be quite challenging because there is no intuitive built in method of string matching entire HTML blocks in either JavaScript or JQuery. In order to solve the string matching problem we were required to iterate over all blocks of HTML and manually attempt to match each one to the instructor selected element. Another issue involved how to edit the HTML page in a manner which works on a variety of different colored backgrounds.

## 8 LIMITATIONS AND FUTURE WORK

It is worthy to consider the application of supporting embedded tutorial creation for the entire desktop environment. However this is a challenging problem to solve given the unstructured layout of user programs, unlike HTML. It would be interesting to further pursue an effective software program which allows stepping you



through a series of steps across a multitude of applications, such as the web browser and command line.

Unlike web applications, such a product for desktop applications could not rely on data structured as HTML. Apollo for the desktop environment seems a likely probability given existing image recognition techniques such as OpenCV (Open Source Computer Vision Library)[4]. There are also a number of limitations that are present in the current iteration of this product. We address each of these in detail.

### *An Improved Student View*

There are a few features which can be added which would improve the student view's user experience. First, a DAG editor panel exists, which gives the user to form new links, but the UI is not as intuitive as we'd hoped. Secondly, there is no ability for the user of the tutorial to skip ahead several steps at a time - or at any particular point. For example, a user may want to skip ahead when they've closed and reopened the browser and want to resume a tutorial session. Lastly, if a user diverges from the path, there is no signal or reminder for the student to get back on course. Implementing these features would greatly improve the user's experience.

### *An Improved Instructor View*

For instructors, there is no current way to edit a recording midway through the recording process. This way, instructors can review and edit portions of their recording before making it available to students. This will prevent instructions from having to re-record entire tutorials if there is a simple error.

### *A Sound Server Implementation*

Currently, there is no implemented server which is designed to handle user authentication or to host recordings. Future work will need to address these user functionality concerns, as well as implement the server which will act as a distribution hub for all of the tutorials. Once the distribution server has been implemented, instructors will be able to upload and edit tutorials as well as send out invitations to students for private access. As mentioned previously, a tutorial can be hosted on an instructor's website (the same is such for a video or PDF).

However, a server implementation would create a central hub of tutorials, allowing students to explore different options on their own.

### *Desktop Applications*

As a Chrome extension, Apollo lacks the ability to context switch to the desktop environment. Therefore, Apollo is not suitable for a hybrid based environment, such as if part of the tutorial is over the web and part of it involves a desktop program or an application outside of Chrome. This is not an implementation issue on behalf of the Chrome extension, but a reality of developing a tool designed for the Chrome environment.

## **9 CONCLUSIONS**

Using Apollo is often faster than creating a detailed PDF tutorial and requires no editing or practicing, as is required for screen recording. Most importantly it is embedded in the student's application, completely removing the barriers from switching back and forth from tutorial to target application which saves students' valuable time.

	PDF	Video Recording	Apollo Extension
Effort Required	Requires screenshots + a significant amount of descriptive writing	Requires voice annotation + stepping through the task and editing	Requires installing the chrome extension + stepping through the task
Embedded?	No	No	Yes
File Type	PDF	Video	Text (JSON)

**Table 1.** A comparison between Apollo and alternative tutorial creation methods

As noted in Table 1, Apollo meets the qualitative metric as an embedded tutorial over alternative methods. It also boasts a high data compression comparative over binary data types for videos and PDFs.

One could argue that developers can implement the same features that Apollo [5] provides directly into their application. This is true, however development of such features takes time, where Apollo has the advantage to

create a tutorial in a shorter amount of time and will work for any website.

Apollo creates an easy to utilize overlay for tutorial creation and viewing in web applications. It leverages HTML's structured property to make it an ideal tutorial creation option to be used for virtually any web based application.

## **A HEADINGS IN APPENDICES**

### **A.0 Abstract**

### **A.1 Introduction**

### **A.2 Content Scripts vs Background Scripts**

### **A.3 Scenarios**

### **A.4 An In-Depth View - The Student View**

### **A.5 An In-Depth View - The Instructor View**

### **A.6 Technical Implementation**

### **A.7 Challenges Faces**

### **A.8 Limitations and Future Work**

### **A.9 Conclusions**

### **A.10 References**

## **REFERENCES**

- [1] *Internet Live Stats - Internet Usage & Social Media Statistics*, [www.internetlivestats.com/](http://www.internetlivestats.com/).
- [2] cpettitt, et al. "Dagrejs/Graphlib." *GitHub*, DagreJS, 18 Aug. 2013, [github.com/dagrejs/graphlib](https://github.com/dagrejs/graphlib).
- [3] cpettitt, et al. "Dagrejs/dagreD3." *GitHub*, DagreD3, 18 Aug. 2013, [github.com/dagrejs/dagre-d3](https://github.com/dagrejs/dagre-d3).
- [4] *Open Source Computer Vision Library - OpenCV* [www.opencv.org](http://www.opencv.org)
- [5] cphamlet et. al. "cphamlet/chrome-assistant." *Github*, 18 Apr. 2018, [github.com/cphamlet/chrome-assistant](https://github.com/cphamlet/chrome-assistant)