## Executive summary

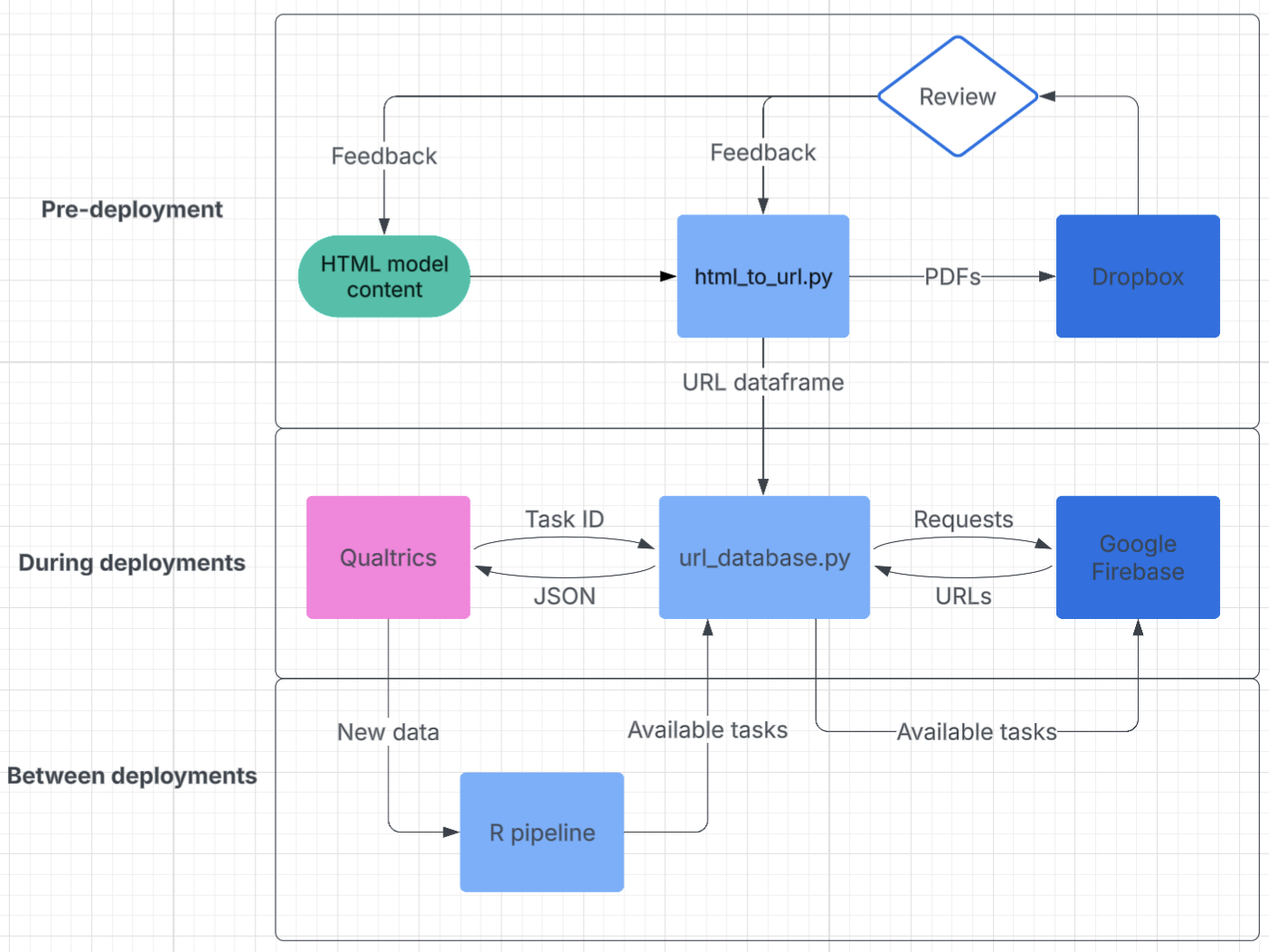
We need to show thousands of participants realistic versions O\*NET tasks and LLM attempts to complete those tasks. Our main task is to show the correct content to each participant. We will use API keys to generate full length example tasks for each O\*NET task (referred to as task “instances”) and corresponding model responses in a standardized HTML format. These HTML will then be converted into formatted PDFs hosted in Dropbox, each with a corresponding preview URL (html\_to\_url.py). The outputs must be vetted for quality and consistent appearance and appropriate content. The preview URLs will be stored in a Google Firestore database and accessed dynamically via an app written with Flask (url\_database.py). Via Javascript, Qualtrics will send a GET web request to url\_database.py for each participant based on the task they are doing. Url\_database.py will return a .JSON with the correct preview links for Qualtrics to process and display to the participant via PDF containers (iframes). Url\_database.py must also mark a task instance as unavailable every time it sends the data for that task instance. When a given job is completed, it must be able to notify the team to alert us to make that task or occupation unavailable. Finally, between rounds we will find a list of uncompleted task instances using an R data cleaning/analysis pipeline, and Url\_database.py must make any uncompleted tasks available for the next round. Depending on the number of surveys we utilize, we may also utilize the Qualtrics API to export data automatically to our R pipeline.

## Components

| Component | Purpose | Tech stack |
| --- | --- | --- |
| Model content generation | Create realistic task instances and model responses | Python, LLM APIs (Google, OpenAI, Claude, LLama…) |
| html\_to\_url.py | Turns HTML to PDFs in Dropbox | Python, Dropbox API |
| url\_database.py | Communicates with qualtrics to provide | Python, Flask, Firestore, GCP |
| Qualtrics | Survey and response display | Qualtrics + Javascript, Qualtrics API |
| R cleaning pipeline | Automatically perform data cleaning to identify available task instances between rounds | R |

## Architecture

#### Architecture diagram



#### Data structure for URL storage:

| Task ID | Task instance # | Complete? | Task URL | Response URL 1 | Response URL 2 | … |
| --- | --- | --- | --- | --- | --- | --- |
| #x | 1 | TRUE | http:// | http:// | http:// |  |
| #x | 2 | TRUE | http:// | http:// | http:// |  |
| #x | 3 | FALSE | http:// | http:// | http:// |  |
| #x | 4 | FALSE | http:// | http:// | http:// |  |
| #x | 5 | FALSE | http:// | http:// | http:// |  |

#### Component descriptions

**Html\_to\_url.py**

* Input: HTML task/response files
* Process: Convert HTML to PDF and save preview URLs in correct data structure
* Output: PDF files stored in Dropbox with accessible URLs and dataframe of URLs
* Details
  + Parallelization: Process as many files as possible simultaneously
  + Error handling: Retry logic for failed conversions
  + Testing: need human and/or LLM verification of valid outputs

**Url\_database.py (Flask)**

* During collection:
* Input: GET request from Qualtrics with task ID
* Process: From the first available of the 5 task instances for that task, select the task instance and the 5 associated model responses
* Output: .JSON with the 6 URLs and updated information in Firestore
* Details
  + Size: ~15,000 rows and 8-10 columns
  + Queuing: must handle rapid reading and writing of Firestore with multiple requests without conflicts
* Between collection:
  + Input: List of completed/not completed task IDs
  + Output: updated information in Firestore

**Firestore**

* Input: requests from Flask app to read or write information
* Output: information

**R pipeline**

* Input: most recent data from Qualtrics
* Process: cleans data, applies quality checks
* Output: list of available tasks for use in url\_database.py, list of people who disqualified and the reason so submissions in Prolific can be handled asap

## Project requirements

Model HTML generation

* Must generate realistic task instances and consistently formatted outputs
* Ideally runs in under a week to enable rapid feedback and improvements

Html\_to\_url.py

* runs in < 1 day for an efficient testing and feedback loop

PDF verification routine

* Need VERY low error rate
* These checks must take less than a couple of days

Url\_database.py

* should return URLs within 5 seconds max
* must be able to handle several hundred requests at once (max: 1000)
* Must be able to mark a task as completed and un-mark it based on list provided between collection rounds
* Must notify team when tasks and occupations are completed

R pipeline

* Ideally gets data directly from Qualtrics via API, cleans and returns available tasks to Url\_database.py automatically and under 10 minutes to minimize downtime

## Project planning

#### Major challenges:

* Efficient uploading of URLs to Dropbox (should parallelize)
* Efficient testing of PDF content and format quality
* Efficient verification of URL functionality in Qualtrics
* Handling of high request volume to url\_database.py
* Verifying when occupations are out of tasks and should stop admitting participants
* What to do when there are few tasks left for an occupation and the participant doesn’t understand the available ones

#### Unknowns:

* Final task and task instance count
* Pricing for Firestore and Flask?
* Are Firestore and flask the best for the job?
* How do database apps work/handle requests?
* Can the app handle these requests from my laptop or do I need to host in the cloud?
* How many qualtrics surveys will we need?
* How to add completion codes for Prolific at scale?

#### Timeline

Phase 0: PDF/URL validation - 2-3 weeks?

(This should be performed in parallel as the other phases progress)

1. Create HTML for model content, when possible - **HARRY**
2. Testing outputs of Html\_to\_url.py - **BRITTANY/ADAM**
   1. Create plan for testing formatting, content, and URL function

Once initial testing looks good, scale to all PDFs

1. Write Html\_to\_url.py - **ADAM**
   1. Adapt current script to new Data structure
   2. Learn how to parallelize upload without loss of organization
   3. Design tests

Phase 1: Analysis and development - 1-2 weeks?

1. Configure and prototype Url\_database.py - **ADAM**
   * Learn if Firestore and Flask will work, and the pricing models
   * Develop testing and error handling plan
   * Reach out to others when/if roadblocks are found
   * Design tests
2. Prototype Qualtrics side - **ADAM**
   * Adjust flow to sort individuals into different tasks within a job
   * Implement javascript to request JSON from url\_database.py
   * Save JSON urls to appropriate embedded data fields
   * Create conditional logic to get correct embedded data field
   * Determine how many surveys we will need, if we can use qualtrics API, and how to scale use with Prolific
   * Design tests
3. See if R pipeline can mesh with url\_database.py to update available tasks automatically between waves

Phase 2: Design and Implementation - 2-4 weeks?

1. Complete and test url\_database.py - **ADAM**
   1. Should be able to:
      1. use full size dataframe
      2. handle requests properly
      3. update between rounds
      4. Handle high volume
      5. Survive all tests designed earlier
2. Complete and test Qualtrics survey(s) and scale
3. Finish Html\_to\_pdf.py if not done already
4. Implement R pipeline
5. Ensure all the pieces work together

Phase 3: Testing - 1-2 weeks?

1. Test the complete pipeline together and piecewise in as many tests as we can determine

## Operational flow walkthrough

***Before collection:***

1. Harry generates tens of thousands of HTML responses
2. **Html\_to\_url.py** converts these HTML into PDFs, uploads them to Dropbox, and stores the URLs in our dataframe structure.
3. The dataframe structure is built into **url\_database.py**
4. We test some n out of N total PDFs for quality and iterate our approach so PDFs are consistent and high-quality
5. We test that the PDFs have a good appearance in Qualtrics
6. We test that the below functionality works in all cases

***During collection***

1. The **url\_database.py** receives task code for task “foobar”
2. It selects the first available URLs for first task instance and associated models (6 URLs) and marks that task instance as unavailable
3. These 6 URLS are returned in a structured format and saved to 6 different embedded data fields Qualtrics.
4. When the user progresses to the task page, the PDF container utilizes the stored PDF URL to show their task instance.
5. When the user progresses to the first response, there is a conditional statement which selects the stored URL for the first model response and displays it. On the second loop, the conditional statement selects the second URL.

***After collection:***

1. When the wave is completed, the data is analyzed with the **R pipeline** and the list of remaining task instances is acquired. This could be the task URL or the task ID.
2. This list is provided to the **url\_database.py**
3. The application must mark these task instances as unavailable for all future rounds.

Pseudo code

Javascript in Qualtrics

Qualtrics.SurveyEngine.addOnload(function() {

// Get the task ID from embedded data or question logic

var taskId = "${e://Field/TaskID}"; // This assumes you have TaskID as embedded data

// Create the API URL with the task ID as a parameter

var apiUrl = "https://your-flask-app-url.com/get\_task\_urls?task\_id=" + encodeURIComponent(taskId);

// Make the API request

jQuery.ajax({

url: apiUrl,

type: "GET",

dataType: "json",

success: function(response) {

// Store the returned URLs in embedded data for later use

Qualtrics.SurveyEngine.setEmbeddedData("TaskURL", response.task\_url);

Qualtrics.SurveyEngine.setEmbeddedData("ResponseURL1", response.response\_urls[0]);

Qualtrics.SurveyEngine.setEmbeddedData("ResponseURL2", response.response\_urls[1]);

// Add more as needed

// Optionally, you can immediately display the task PDF if needed

jQuery("#task-pdf-container").html('<iframe src="' + response.task\_url + '" width="100%" height="500px"></iframe>');

},

error: function(jqXHR, textStatus, errorThrown) {

console.error("Error fetching task data:", errorThrown);

// Handle the error as needed

jQuery("#error-message").text("Error loading task data. Please contact the survey administrator.");

}

});

});