INTRODUCTION

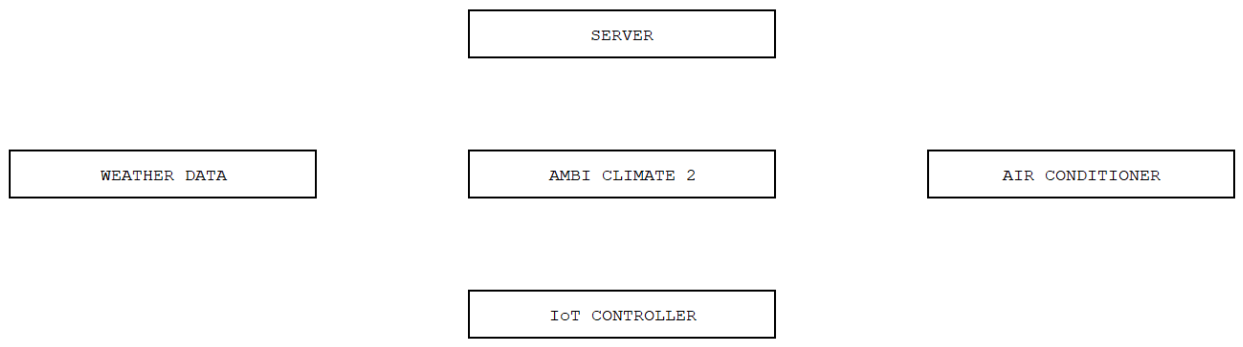
Before I begin, I think it is important to clarify what this document is.

Virtually everyone I know has a love-hate relationship with air conditioners. My mother, for example, opts not to turn on the air conditioner at all because she is afraid it will give her a cold. My grandparents, on the other hand, crank the air conditioner so cold in the summer that the rest of the family have to wear sweaters. Air conditioner preferences have actually caused some spats between my family members. I think your company is solving a really interesting problem, and I would like to work at Ambi Labs as a programmer / software developer.

I am very interested in programming and have been teaching myself Python for the past two years. But alas, my college degree has nothing to do with computers (or even math, for that matter). Other than some side projects, I have no work experience that requires computer programming. How do I gain IT work experience without IT work experience? How can I prove that I have baseline domain knowledge of programming when my resume says otherwise?

My solution is this thought experiment. Using publicly available information, such as job postings, GitHub, instruction manuals, and promotional materials, I am going to try my best to reverse engineer different parts of the Ambi Climate system. I am relatively new to this and I obviously don't have access to all the information, so I am probably going to make some mistakes. But it is my hope that this document will help me stand out, prove my skills, and encourage me to learn more about industrial-scale software solutions.

SYSTEM-LEVEL OVERVIEW



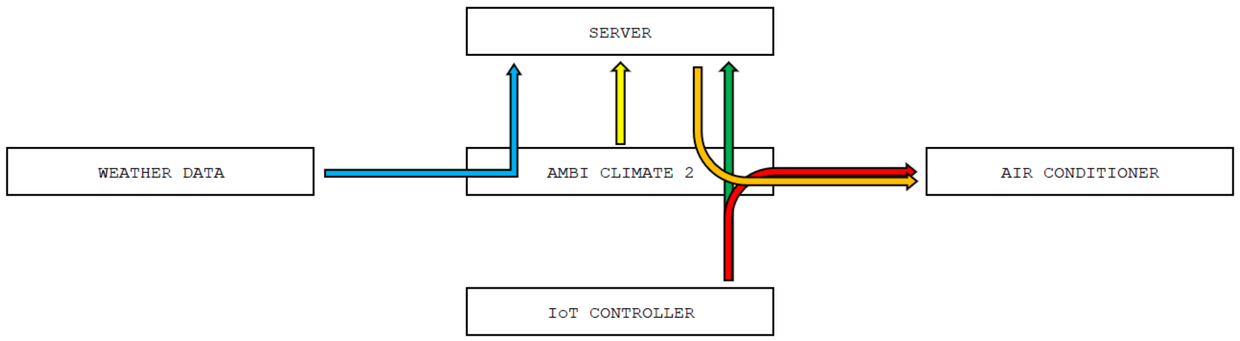
As I understand it, there are five components to the Ambi Climate system:

- **Server:**
    - Collects data from Ambi Climate controllers and stores it in a database
    - Reads training data, trains the machine learning model, and saves the machine learning model
    - Serves predictions and recommendations back to controllers
- **Air Conditioner:**
    - Receives infrared signals from Ambi Climate controller
- **IoT Controller:**
    - Examples: Apple app, Android app, Alexa, Google Home, IFTTT, etc.
    - Sends users' commands to Ambi Climate controller
    - Receives usage data from the Ambi Climate controller

- **Weather Data:**
    - Possible examples: Weatherstack, Dark Sky, OpenWeatherMap
    - Sends real-time outdoor climate data (temperature, conditions, and humidity) to Ambi Climate controller via periodic API calls
- **Ambi Climate 2 Controller:**
    - Uses built-in sensors to collect indoor data, such as temperature, humidity, ambient light levels, and thermal radiation
    - Uses API calls to collect contextual data, such as outdoor weather data (see *Weather Data*), time of day, and date
    - Receives user input from IoT controller(s)
    - Receives machine learning predictions from server
    - Sends all collected data to server
    - Controls air conditioner using built-in infrared transmitters

FOCUSING IN: AMBI CLIMATE 2 CONTROLLER

To keep things manageable for both of us, I'm only going to focus on the Ambi Climate 2 in this document, hereby referred to as "Ambi". Let's look at how Ambi interacts with other parts of the system:



Ambi relays commands from IoT Controller to Air Conditioner.

1. Person to Alexa: "Alexa, I'm a bit warm"
2. Alexa parses speech data, calls `decrease_temperature()` function in Ambi
3. Ambi decreases the Air Conditioner temperature using infrared signals
   (Allows person to control the air conditioner)

Ambi relays predictions from Server's machine learning algorithm to Air Conditioner.

1. Using real-time input data, the Server serves a prediction regarding the person's comfort level. In this example, the current A.C. temperature is hotter than the person's predicted comfort level.
2. Server calls `decrease_temperature()` function in Ambi
3. Ambi decreases the Air Conditioner temperature using infrared signals
   (This is the "AI" functionality that is advertised on the label)

Ambi sends real-time sensor data to Server.

1. Ambi sends temperature data from built-in temperature sensor to Server
2. Ambi sends humidity data from built-in humidity sensor to Server
3. Ambi sends light data from built-in photoresistors to Server
   (Features for the machine learning data)

Ambi sends IoT Controller commands to Server database.

1. When a person uses an IoT Controller to change the temperature (i.e. provide feedback), Ambi sends that command to the Server database.
   (Label for the machine learning data)

Ambi sends local weather data to Server.

1. During setup, Ambi creates a WiFi access point. This allows Ambi to communicate efficiently with other components of the system (e.g. the IoT Controller). This also fixes Ambi's IP address, which allows it to geolocate itself and request data from third-party weather services.
2. Ambi sends `GET` requests to a weather API as often as possible. The frequency of these requests, however, would depend on the API's rate limits, cost, and data update frequency. (For example, a business plan at Weatherstack would be able to support 22 devices sending 1 request/minute. If the devices were sending 1 request/5 minutes, that plan would be able to support 110 devices.)
3. I'm not sure how often Ambi queries the weather API, but due to the constraints listed above, I am fairly certain it is not real-time data. In other words, the Server is likely querying Ambi more often than Ambi can query the weather API. To remedy this problem, Ambi likely uses Redis to store weather data. I say this because:
   a. Redis is ideal for hardware – it's fast, not CPU-intensive, and in-memory
   b. Redis is NoSQL, so it is very JSON-friendly
   c. Redis keys can be set to expire after a certain amount of time, which is perfect if Ambi is making periodic API calls
4. To summarize, Ambi uses its IP address to geolocate itself. It uses its location data to query local weather conditions every `x` minute(s) and saves that data to Redis. The Server queries Ambi's Redis database every `y` minute(s).
   (More features for the machine learning data)

AMBI'S HARDWARE

Ambi has three inputs – 1 photoresistor, 1 temperature sensor, and 1 humidity sensor.

Ambi has eight outputs – 7 infrared transmitters and 1 RGB status LED.

Ambi is powered by a +5.1V micro USB.

Ambi has 2.4 GHz 802.11.b/g/n WiFi capabilities.

Ambi is running an Ubuntu Linux system.

If I were to hazard a guess, I would say that Ambi's hardware closely resembles a Raspberry Pi 3B+. The extended 40-pin GPIO header would provide an easy way to connect the inputs and outputs via a breadboard. A micro USB could power a Raspberry Pi 3 and all of its accessories. You understandably didn't splurge and get the WiFi dongle for 5 GHz WiFi. I'm guessing Ambi uses Ubuntu Core, which is lightweight and secure.

I would like to learn more about the infrared transmitters though. How do they know where the air conditioner is? Why are there so many? One hypothesis is that Ambi knows where the air conditioner is, and it aims all seven transmitters at the air conditioner to increase the chances of success. Another hypothesis is that Ambi doesn't know where the air conditioner is, and all seven transmitters are aimed in different directions or at different angles to increase the chance of finding the air conditioner.

AMBI'S SOFTWARE

I tried my best to recreate Ambi's software, and you can find my GitHub repository here: https://github.com/neil-rutherford/iffy-weather. I used Flask for the framework/API, ZeroMQ for messaging to the Server, Flask-ASK for Alexa integration, Redis for persistence, and Requests for outbound weather queries.

I was stumped by a few things. First, I couldn't figure out the software behind the infrared signals to the air conditioner. If I had to guess, I would say that you were using LIRC, but that seems a bit outdated. I also couldn't figure out how Ambi's access point receives WiFi credentials from the app and connects to the internet.

I am also realizing in retrospect that I forgot to deal with the status LED. Oops!

CONCLUSION

Depending on my time and obligations, I will try to make another report for a different aspect of the system. Machine learning is what got me into Python in the first place, so I am really interested in digging into how your algorithm works.

I had a lot of fun putting all of the pieces together. Because I always work on small-scale projects, I have always been curious to see how things like this scale up to an enterprise level. I think the most important takeaway was how servers can handle all those requests using asynchronous programming. I also got to learn about NoSQL databases, and I can't wait to play with Redis more, since it seems like a really cool and versatile platform. Most of my previous experience had been with SQL databases, so I also got to learn how the two database types can complement each other. Thank you very much for the learning opportunity!

I asked a few friends for advice and copied some sample code (especially with the sensors), but everything else is my work. The whole process, from first identifying your company to finishing this report, took about five days. I was originally going to apply for this job, but I realized after I started that it was no longer accepting applications. I sincerely hope that you will consider me for your team if something opens up!