Final Report

Ahmad Raaiyan (CEO), Aum Palande (CTO), John-Carlos Breck Ortiz, David-Tyler Ighedosa
Gatekeeper
CSCE 482 - 934
4/25/25

Revision History

Revision Editors	***Revision Number***	***Date***
Ahmad, Aum, David, Breck	1	4/25/25

Introduction

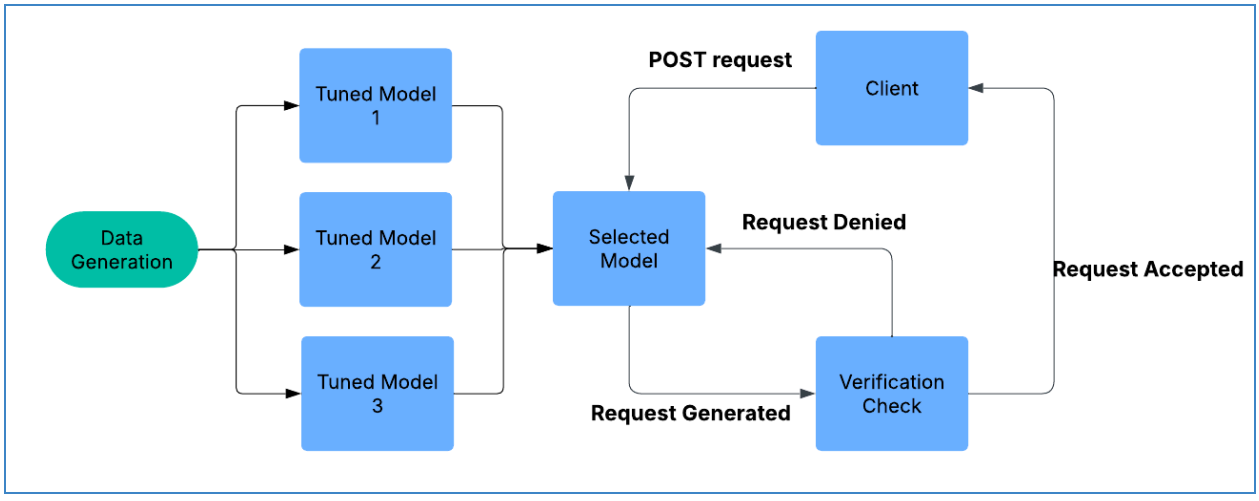
As a disclaimer, this project is not sponsored by Google. Our mentor is Nitin Mittal, a Software Engineering Manager at Google who works on the Cloud Identity and Access Management (*IAM*) team. As a mentor, Mr. Mittal has created a project related to the Cloud Service he works with. The project involves using a Large Language Model (*LLM*) to generate accurate allow policies for the Cloud IAM service through given prompts.

Project Overview and Scope

A more detailed description of the project; defining the major subsystems, their interactions, and the general scope of the project. This section SHOULD include a diagram showing how the entire project works, with meaningful interactions and labels between the different subsystems.

Overview

This project is intended to help people create policies on the GCP platform for controlling the abilities of users within a given organization; these policies are similar in nature to roles in an RBAC system, however given the large scope of GCP's offered services there are a plethora of individual policies that must be configured manually. Our aim is to have the user provide plain-English text describing the policies they would like to implement, and have our final product use Large Language Models to determine the appropriate policies as well as verify them. Finally, we plan on making a seamless way to integrate and apply policies into the user's project.



Scope

For the overall scope of the project, we will start relatively simple and easy for the complexity of the LLM model we use and its functionality. We aim to generate Identity and Access Management (*IAM*) policies with our chosen LLM by giving it specific and detailed prompts. This means that when prompting the model, the user would need to be more knowledgeable of the IAM system and its policies in order to get effective results from the model that they can use. From this goal, we hope to increase the complexity as time allows by allowing prompts to be less detailed, with the model still performing just as well or relatively close. This means that prompts can generally be broader and the output would be more specific in generating a policy while informing the user of where to use it.

Team Responsibilities and Roles

Overall, we will all try and do our fair share of the workload. This means that we strive to be around 25% of the overall contribution of the final result; however, this does not necessarily mean exactly 25% of the code. We as a group understand that a lot of our project will be research, creating synthetic data, and designing. That being said, our CEO will be the main point of contact for Mr. Mittal. He, with the support of the rest of the team, will try and develop a strong relationship with our client, in order to make sure that our project is what he expects and requests. Similarly, our CTO will be designated as the person in charge of team meetings and internal communication. The CTO will establish internal meetings with the team to keep us moving forward and on schedule. Our main communication as a team will be Zoom and Discord, while communication with the client will be through Email, WhatsApp, and Zoom. We have planned to use the in-class time as our main meetings, and if necessary will adjust our schedules on the weekend in order to find a good time to meet.

Our work was primarily divided into frontend and backend. Ahmad and Aum worked primarily on the backend, while David and Breck worked on developing the frontend.

Technical Overview

Tech Stack

For the model, our tech stack consists of using Python with FastAPI to write our program and access the APIs. We will be using are Google Cloud Platform for the policies and other GCP features as well as Groq’s llama for our model. In addition to this, we will need to create a user interface for the final product for users to interact with the model. This frontend will be in React, with the help of Shadcn for our styling framework. This will allow us to utilize node modules in order to create a sleek and modern UI.

Subsystem 1 Requirements: Backend

Functional Requirements

Our main point of the backend is our fine-tuned LLM model. This is called and accessed through FastAPI, where we will be able to send requests to our API in order to generate responses for new users. The specific requests to the backend can be to generate a policy, apply a policy, or get the projects of an authenticated user.

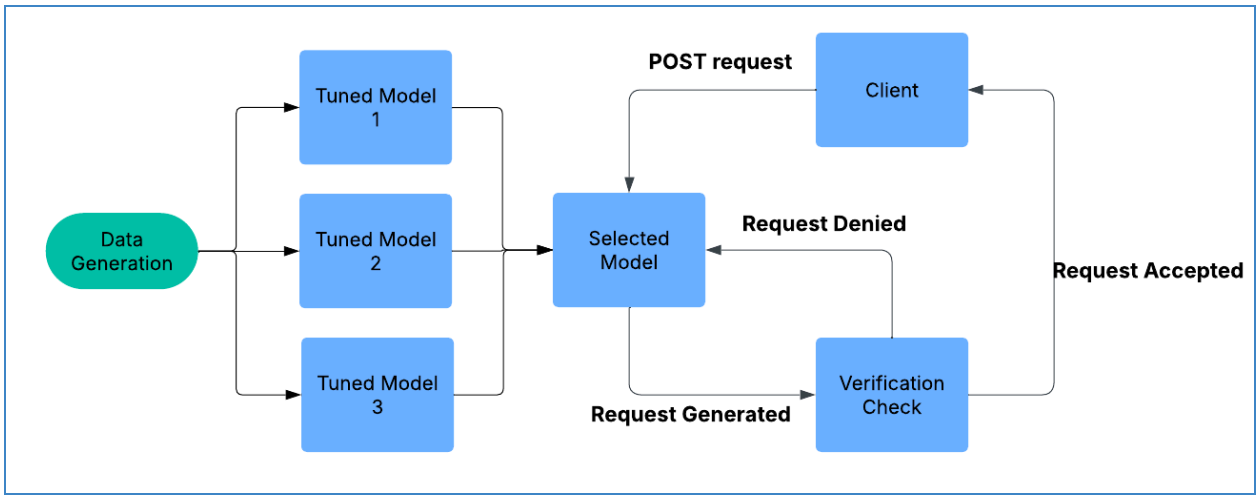
Software Characteristics

Our backend system is an LLM implemented through the llama model served by Groq. Groq is a fast model provider that helps lower the response time. This LLM will be trained on data points that simulate a real response from users. Most of this data will likely be synthetic. This will allow for lots of training data, which, through the help of cross-validation, will be key to combating overfitting.

We have 3 main routes for our backend, which are all necessary and are relied on heavily for the full flow of the system. This backend handles requests and also acts as a gateway to the Google Cloud API. We use portions of the Google Cloud API Python client, such as the Google OAuth2 library for authorization and logging in, the Google API Client for accessing the Google Cloud, and the Google Lint library for our verification check.

Inter-subsystem Communications Requirements

Our backend has the following flow of data: We first tuned models using synthetic data we generated, which then selects a specific model to use in our requests. When a frontend client posts to our backend model, we get the model to generate a request. It then goes through the Google Lint library to verify that the policy is correctly formatted, giving the correct roles, and not giving too much permission. If the policy is denied, then it gets regenerated by the selected model until it eventually gets accepted by the verification check. Once it has been accepted, the policy gets sent to the client.



Failure Propagation

If any of the subsystems go down or fail to work properly, we will not have a way to request our models or have any of our Google Client connections. This would mean that our frontend will have no real functionality. To prevent this, we have created lots of unit tests for our CI/CD pipeline, which must be working before any commits are made into the deployed branch. These tests are enough to ensure that all forms of possible requests are handled, and will need to be expanded as the backend grows.

Subsystem 2 Requirements: Frontend

Functional Requirements

The frontend is responsible for providing a clean and intuitive interface for the user to interact with the backend. It must support the following core functions: signing in with Google OAuth, generating a policy from user input, selecting a GCP project to import the policy to, copying the generated policy, manually editing it if needed, and applying the policy to the selected project.

Software Characteristics

The frontend is written in TypeScript with React.js, and styled using the Shadcn UI component library. It prioritizes responsiveness and ease of use, focusing on keeping the workflow simple and linear. The UI is designed to guide users through the policy creation process step by step, minimizing confusion and errors. Error handling and user feedback are built-in to improve overall usability.

Inter-subsystem Communications Requirements

The frontend communicates with the backend via API calls to perform key actions such as policy generation, Google Cloud project retrieval, and policy application. It needs to structure these

requests correctly and render the backend's responses in a readable and actionable way for the user. Any changes made by the user (e.g., manual policy edits) must also be sent back to the backend when applying the policy.

Failure Propagation

If the frontend goes down or fails to respond, the project essentially loses all core functionality. If the frontend goes down, users won't be able to access or interact with any part of the system. Even though the backend and LLM services might still be running, users won't be able to generate policies, sign in, or apply anything to their GCP projects. Since the frontend acts as the entry point for all user actions, its failure essentially halts the entire product.

Ethical Considerations

Data Privacy Compliance

We will ensure that sensitive data is safeguarded and complies with AI regulations such as the GDPR and HIPAA.

Data Minimization

We plan not to have data from users. As most of our data will be synthetic, we can train and deploy our model to production without any real data. The only possibility of holding personal data would be in judging the responses from real-life inputs, but this will be anonymized and discarded after fixing the issue.

Anonymization and De-identification

We will hold as little personal data as possible. In training, we plan on removing any sort of personal identification data, like names of groups, organizations, or users. In any other case where we would need to hold data, whether it is synthetic or real, we will remove any names that can trace the data back to the source.

Data Security and Encryption

We will make sure to secure our data and encrypt it to better ensure that any personal or sensitive information is not violated by malicious actors.

Third-Party Tools and API Compliance

We are using the Google Gemini API, which has strict standards for providing data. Due to these concerns, we plan on following the API guidelines, as well as verifying every request to the API. This entails making sure the response is not flagged as hate speech, harassment, or other safety filters provided by the API guidelines

Data Retention and Disposal Policy

There is only one scenario where we will hold data that was provided by a user. In the event that a response is incorrect and we want to study the error, we might need to download the text conversation. Once the error has been noted and fixed, the text conversation will be disposed of.

AI Considerations

When it comes to AI, we will be using Google Gemini as our LLM. Other than that, tools such as ChatGPT or GitHub Copilot may be used to assist in code development. We plan on using AI to assist us with issues, rather than solve problems that we should be doing by hand.

Development Schedule

Initial commit and proof of concept: (Feb 17th) This is when we first decided on our project and Tech Stack. We wanted to ensure that finetuning would be possible for these models, specifically the Gemini model. We had made a Python Notebook by this point, which allowed us to continue.

Setup Frameworks and initial data generation script (Feb 26): We made our first frontend/backend linkage here, with FastAPI and NodeJS. This was also where we started generating synthetic data to fine-tune our model. We also had a basic benchmarking script created so that we can start working on gathering data about which model to use.

Major frontend improvements (Mar 4): By this point, we had made big strides on our frontend and backend, specifically in how our frontend looks and usability. It was a lot more usable and fleshed out by this point. We had also made some changes to our benchmarking and API logic.

Project Selector with GCP connection (Mar 25): This was when we started integrating with the GCP to import projects and allow the user to apply the policy to the account automatically. We also improved our prompt for our fine-tuned model to have higher accuracy and added a verification check.

Front End overhaul (Apr 7): While our backend was close to being finished, our frontend looked odd and didn’t fit the theme. We went through and cleaned it up with Shadcn and fully remodeled our UI to look more professional, modern, and sleek.

Testing (Apr 20): We implemented, tested, and fixed our frontend and backend linkage to ensure that everything was working properly and covered all the possible situations that might arise. Most of these fixes were tiny UI changes or bug fixes.

Final Product (Apr 25): Now that everything is tested and such, we made changes to the ReadMe, our prompt, and added another verification check from the Google Cloud API for a final check.

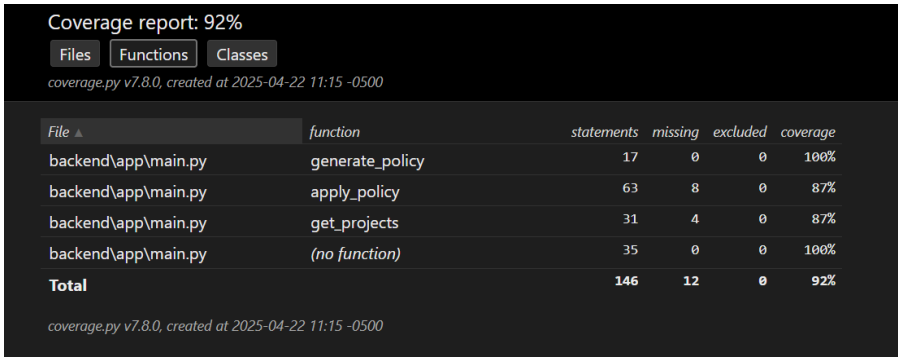
Testing

Backend Testing

Our backend tests are very detailed and need to be diverse enough to cover all possibilities of requests. Because we do not know much about how the Google Cloud API could react based on the data that we give it, we want to control how different cases are handled ourselves. This comes in the form of diverse testing and edge case handling in our routes.

We did our backend testing with PyTest, making sure to get and maintain a high coverage throughout our whole development process. These made sure to get all edge cases and pathways of our routes and the Google Cloud APIs. We also made sure to have sunny day tests, which we expect to work, and they simulate normally formatted requests from the frontend. These are especially useful to make sure that our backend is not rejecting requests that should be returning proper data.

At this point, all of our test cases are working. As we were creating test cases, it exposed some bugs that could have happened in our routes, so we made sure to patch those issues. As the project expands, there also needs to be an expansion of the testing suite and the coverage of our test cases.



Coverage report: 92%

Files | Functions | Classes

coverage.py v7.8.0, created at 2025-04-22 11:15 -0500

File ▲	function	statements	missing	excluded	coverage
backend\app\main.py	generate_policy	17	0	0	100%
backend\app\main.py	apply_policy	63	8	0	87%
backend\app\main.py	get_projects	31	4	0	87%
backend\app\main.py	*(no function)*	35	0	0	100%
Total		**146**	**12**	**0**	**92%**

coverage.py v7.8.0, created at 2025-04-22 11:15 -0500

Frontend Testing

Our frontend testing focused mainly on end-to-end functionality and handling edge cases in the interaction between the UI and other components. Since most of the frontend relies on calling backend routes and rendering dynamic data, we made sure that our core user flows worked as expected under different conditions.

We tested policy generation at various prompt sizes to ensure that the model's response could be properly received, displayed, and copied from the UI. This also helped stress test our models, which is something that is not always tested in benchmarking. We also ran full

end-to-end tests for each of those prompt sizes, going through the entire flow: sign in, enter a prompt, receive a response, select a project, and apply the policy. This also included accounts that either had never used Google Cloud or didn't have projects and such.

Other key areas we tested included the copy-to-clipboard feature for generated policies and displaying messages from the LLM in the chat window. These tests made sure the interface remained stable and consistent across a variety of usage patterns.

Future Work

If we were to take this project further, there are several areas that could be improved or expanded. These changes would mainly focus on making Gatekeeper smarter, easier to use, and more maintainable over time.

One big area of improvement is making the model better at handling less specific prompts. Right now, users need to be pretty familiar with how IAM works to get useful output from the model. In the future, we'd want to support more natural, vague, or broad prompts while still generating accurate policies. To do this, someone could implement a retrieval-based system (like RAG) that gives the model extra context from real documentation or past policies when generating responses. Another potential upgrade is training the model on real (but anonymized) policies from different industries. This would help it give more accurate responses in specialized use cases like healthcare or finance, where permissions get pretty specific.

The current frontend works, but there's definitely room for improvement when it comes to helping users write better prompts. One idea would be to add a guided prompt builder or auto-complete feature that walks users through what they need. For example, instead of typing "give read access to BigQuery," users could select "BigQuery" and "read access" from dropdowns, and we'd generate the prompt for them. This would make the tool easier to use, especially for people who aren't familiar with how IAM is structured. To build this, someone could fetch service metadata from GCP and generate dynamic forms or suggestions based on that. Not only that, but pulling more data from GCP could help with generating more accurate responses and allow the user to make a more helpful policy.

Right now, the policy either passes the verification check or it fails. A more helpful approach would be to show the user why a policy failed and what changed compared to their current one. This context could also go back to the model when generating a second policy. A future feature could be a "diff" view that compares the new policy with the existing one and gives a quick breakdown of the risks or changes, and it could let the user choose the best one as future training and validation data.

Conclusion

Working on Gatekeeper this semester has been a great learning experience for all of us. We had the opportunity to build something that sits at the intersection of cloud infrastructure, large language models, and real-world security needs.

One of the biggest takeaways was just how tricky it can be to work with complex APIs like GCP's. Between understanding the IAM system, generating synthetic data, and building a reliable pipeline to test everything, we had to think through a lot of edge cases and unexpected behavior. That said, it also showed us how powerful and flexible these systems can be when things are working smoothly.

If we could go back and do something differently, we probably would've spent more time early on figuring out how to make our model better at understanding vague prompts. It turned out to be a bigger challenge than we expected, and while we made a lot of progress, there's still more that can be done. That said, we're proud of what we were able to build in a semester — a working tool that can take a user's input and turn it into a verified, usable IAM policy.

What we enjoyed the most was seeing all the pieces come together — from training the model on synthetic data to getting that first working policy applied directly to a GCP project. It's easy to underestimate how much work goes into making something feel seamless, and now we definitely have a deeper appreciation for what that takes.

We hope Gatekeeper can help make IAM management a little easier for developers, teams, and IT personnel working in the cloud, especially those who don't want to write JSON files by hand. We're excited about the potential for Gatekeeper to grow into a powerful tool that simplifies cloud security, and we hope that we can take it even further.

Appendix A: Acronyms and Abbreviations

LLM: Large Language Model
IAM: Identity and Access Management
CEO: Chief Executive Officer
CTO: Chief Technology Officer
API: Application Programming Interface
GCP: Google Cloud Platform
RAG: Retrieval Augmented Generation
UI: User Interface

Appendix B: Definition of Terms

Policy: A collection of bindings that specify who has what permissions on a determined resource.

Backend: The part of the application or system that handles operations that users do not see, such as the application of a policy or authentication.
Frontend: The part of the application that users see and directly interact with through communication with the backend.