SupaRAG

Step 7 Deliverable

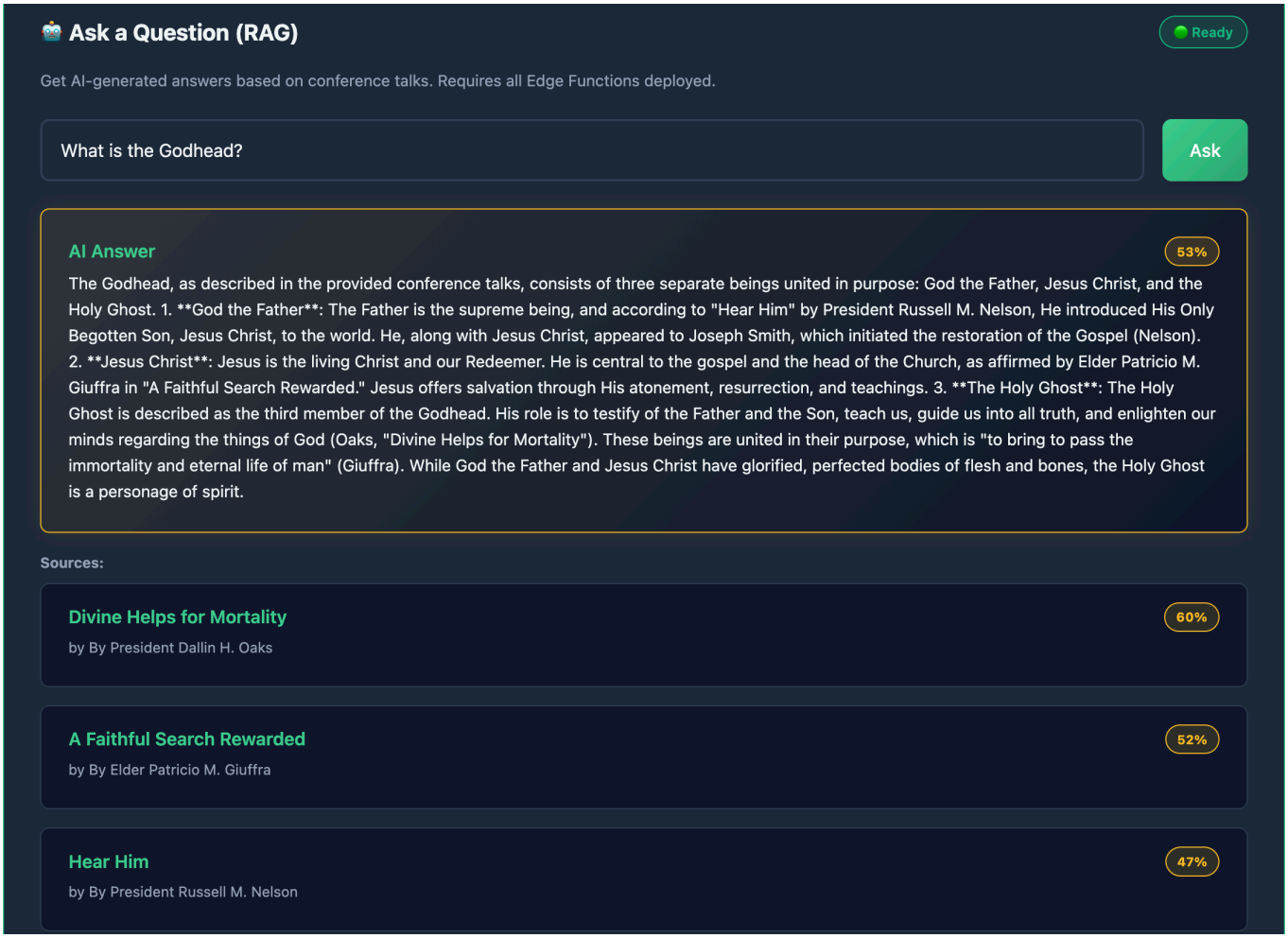
1. GitHub Repository URL

https://github.com/ianjrobertson/conference-rag/tree/main

2. Live Deployment URL

https://ianjrobertson.github.io/conference-rag/#

3. Screenshot



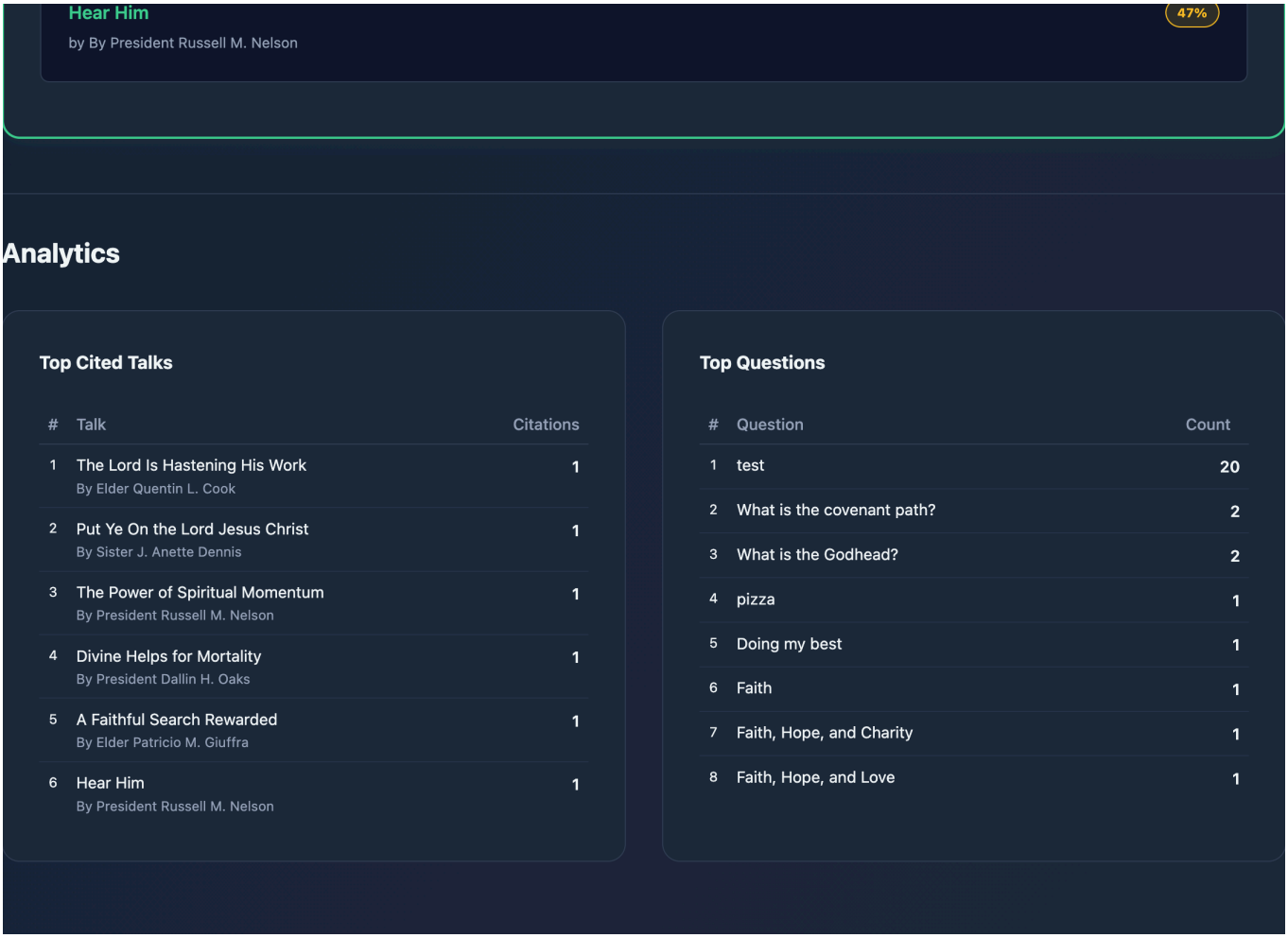
4. Custom Feature — Analytics Dashboard

What I added:

I created an analytics feature for the app that tracks the questions that are being asked and the the talks that are being cited.

Why I chose this:

I thought this feature would be simple to implement and it would be interesting to see which questions and talks kept coming up repeatedly.



---

5. Written Reflection

1. Security Architecture

The anon key in supabase is a credential that is allowed to be exposed in the client/frontend code. This secret relies on row level security to keep the database secure. Supabase is cool because you don't technically need an authenticated server, and can make API calls directly from the frontend because of the publishable key and RLS.

The service key bypasses RLS, so it is not secure to put in the frontend. Edge functions protect the keys, because they run in a secure environment on the cloud and don't expose credentials to the frontend.

RLS provides fine grain access over the indidual queries at the row level. There are checks that run before any database action that can check if the user performing the action is authenticated. In supabase, this happens if a user in the auth.user table has an active session cookie and supabase does some auth logic behind the scenes to check if the user is authenticated before the database actions can occur.

2. Edge Functions & the "Secure Middle"

If we called OpenAI directly from the browser, we would need to expose the API key in the frontend code. Then anyone could see what the secret was, which would be expensive!

The browser sends an http request to the supabase edge function, that contains our supabase session cookie we got when we authenticated. With the cookie we are now authorized to use the supabase edge functions and perform database operations. Our edge function sends an http request to openAI's api using the key we have in our secrets, this key get's loaded from the secret manager at runtime in the secure environment.

The JWT is used to validate our request and ensure that we are authorized to perform the request, likely by comparing secret hashes. This pattern is commmon in many applications. Secrets always need to be handled in a secure way to ensure that they don't get leaked to the client. All endpoints should require validation using something like JWT.

3. From SQL to Semantics

Keyword search is faster and simpler, it just loops through the entire text to find any occurences of the keyword. It works well if you know the word you are looking for, because you can find it quickly and simply.

Semantic search relies on the embeddings to compare meanings. This search doesn't require the word or phrase to exist in the dataset in order to get relevant information. It also allows for fuzzy searching, so you can say something similar or close and still get relevant answers.

Keyword search fails if the word is not found in the dataset. For example if I search `Faith, Hope, and Love`, nothing comes up in the keyword search, likely because `Faith, Hope, and Charity` is a more common phrase. However `Faith Hope and Love` returns plenty of talks for the semantic search, because the embedding for that phrase is similar to talks that probably also reference `Faith, Hope, and Charity`.

In the reverse direction, if I search `pizza`, nothing comes up in keyword search, which is okay. But If I search `pizza` in semantic search, it's going to try to find an answer regardless, even if there is not a great one. In this case, the semantic search times out because there is nothing good, but it can't fail gracefully, which would cause hallucination if it eventually found one.

4. RAG vs Fine-Tuning

Fine-tuning happens when you keep training the internal weights in a model to adjust for more data. For example if you took GPT and trained it on more relevant information for 2026. Fine-tuning puts the relevant information right into the model, so it doesn't need to look anything else up.

Fine-tuning takes time and is expensive, but also does not require extra lookups. RAG is cheaper and can be done at query time to provide releative answers for a question, even if the model does not have those answers right away. However RAG requires embeddings of a dataset to work.

RAG works well for this project because we can add more conference talks to our data easily, instead of having to use a GPU to do more expensive training runs. We can also look at which talks the model is quoting.

Fine tuning might work better if the domain we are working in is very specific, and we need specific tone, style, or behavior. Or if we have a large budget for re-training models.

5. AI-Assisted Development

The AI assistant worked well for building the edge functions and debugging when issues happened with the code.

It didn't add much value for working through the quide. The information in the steps looked like it was made by AI already, so having AI regurgitate it again didn't add much. It also tried to run commands incorrectly.

AI is good at following directions, but it's not good at seeing the big picture about what we're doing.