Lightcast Consulting Cloud Project Report

1. Introduction

The Lightcast Consulting Cloud project was designed to modernize and transform a critical internal workflow at Lightcast. Previously, data modeling projects relied on heavily customized Excel-based systems that had been in place for over two decades. These manual workflows limited scalability, caused maintenance challenges, and introduced human errors into the consulting process.

The goals of this project were clear: **reduce dependency on manual Excel workflows**, **increase scalability**, **enable cloud-based automation**, and **streamline user interactions** for both technical and non-technical users.

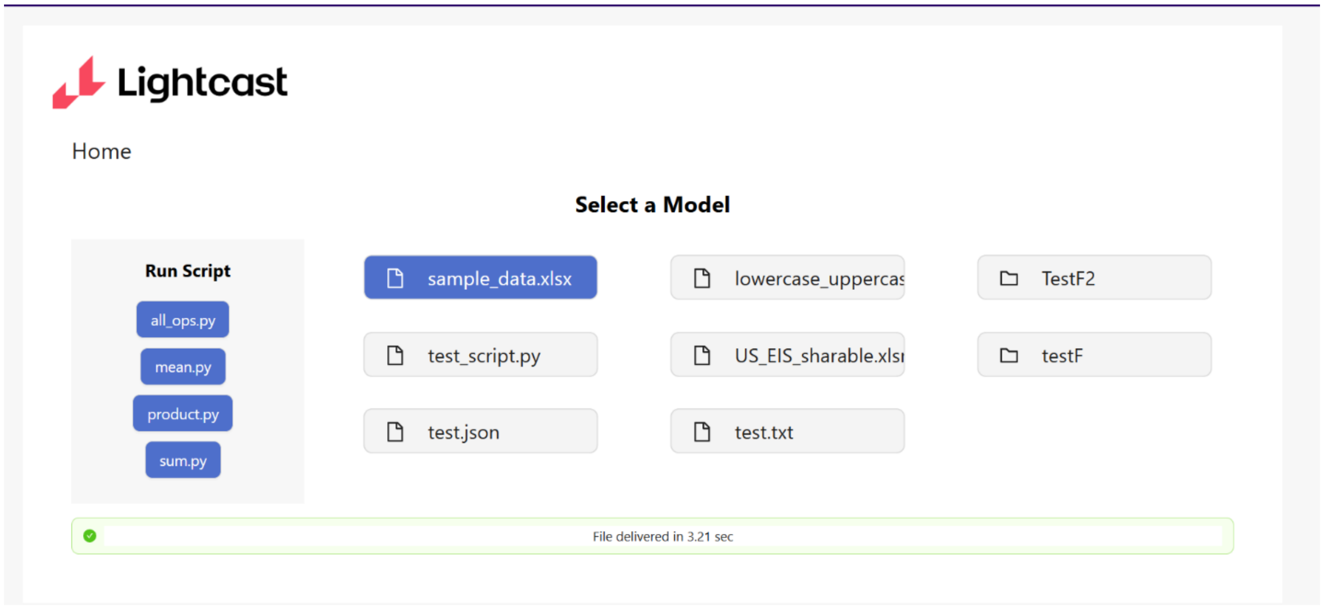
Our final deliverable replaced the legacy system with a **hybrid solution** that combines a **web-based frontend**, a **secure cloud backend**, and a **local helper** to handle computationally intensive tasks without overloading cloud infrastructure. This architecture ensures that Lightcast remains competitive, scalable, and future-ready as they continue expanding their consulting services.

2. System Overview and Architecture

At the highest level, the system consists of three interconnected components:

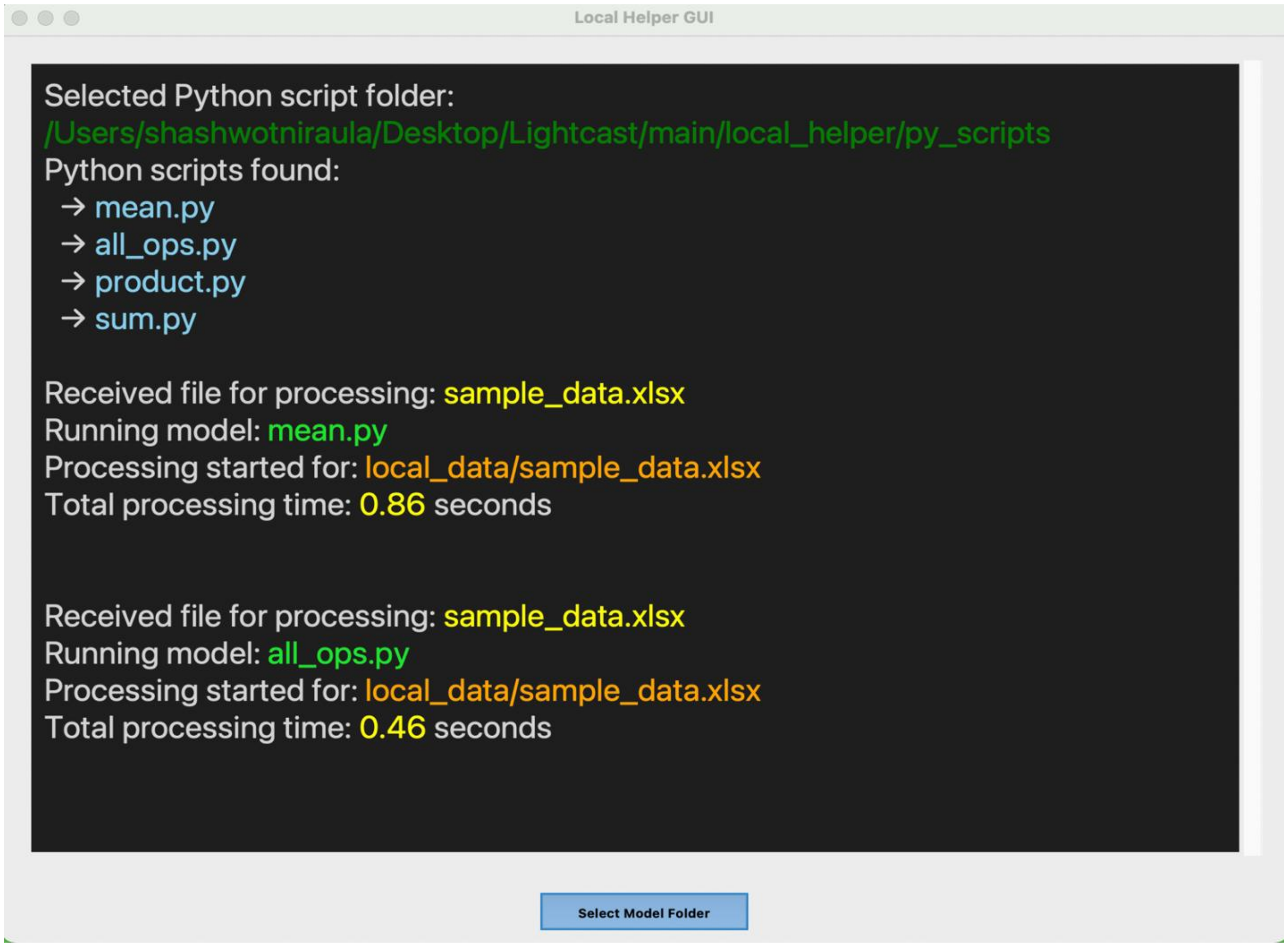
- **Frontend**: A React + TypeScript web application that provides an interface to authenticates user, browse Google Drive, select input files, and trigger local script execution.
- **Backend**: A FastAPI service that handles OAuth2 authentication, manages Google Drive file operations (upload/download/search), and mediates communications between the frontend and the Local Helper.
- **Local Helper**: A desktop Python application built with Tkinter (for GUI) and Flask (for local API service) that runs user-selected Python scripts locally to process Excel files.

This **modular design** allows each component to be **developed, maintained, and scaled independently**, while ensuring smooth communication between them.
As part of deployment, **both the frontend and backend have been hosted on AWS**, specifically utilizing **AWS Elastic Beanstalk** for backend API hosting and a public AWS S3/CloudFront setup for serving the frontend.



File delivered in 3.21 sec

Fig 1: Frontend Design



```
Selected Python script folder:
/Users/shashwotniraula/Desktop/Lightcast/main/local_helper/py_scripts
Python scripts found:
  → mean.py
  → all_ops.py
  → product.py
  → sum.py

Received file for processing: sample_data.xlsx
Running model: mean.py
Processing started for: local_data/sample_data.xlsx
Total processing time: 0.86 seconds


Received file for processing: sample_data.xlsx
Running model: all_ops.py
Processing started for: local_data/sample_data.xlsx
Total processing time: 0.46 seconds
```

Fig 2: Local helper Design

3. Technologies Used and Rationale

Component	Technology	Rationale
Frontend	React + TypeScript	Provides a modular, scalable, and responsive UI. TypeScript ensures type-safety, reduces runtime errors and improves developer productivity.
Backend	Python with FastAPI	FastAPI offers asynchronous request handling and automatic API documentation. Python's flexibility made it easy to integrate Lightcast's pre-existing modeling scripts.
Local Helper	Python with Tkinter + Flask	Tkinter offers lightweight GUI capabilities with minimal external dependencies. Flask allows quick local API setup for accepting file and script execution requests.
Authentication	Google OAuth2.0	Industry-standard secure authentication. Integrated seamlessly with Google Drive APIs for file access and management.
Hosting	AWS Elastic Beanstalk + EC2 + S3	Provides scalable, reliable, and cost-efficient hosting environments for production systems.
Build System	Vite (Frontend)	Ultra-fast build and hot-reload times for React-based frontend during development.
Package Manager	Yarn	Offers better dependency management and installation speed compared to npm

This technology stack matches the project's **core needs of scalability, reliability, rapid development, and cloud readiness**, ensuring Lightcast has a robust foundation for future growth.

4. Functional Overview

A user logs into the **web frontend** using their Google account through a secure **OAuth 2.0 flow**. Once authenticated, users can browse their personal Google Drive structure, navigate through folders, and select **Excel-based input files** for processing.
After selecting a file, users also choose a **Python script** (made available through the Local Helper) they want to run.
The **backend server**, hosted on AWS, securely downloads the selected file from Google Drive

using Google APIs, then sends an HTTP request to the **Local Helper** running on the user's personal machine.
The **Local Helper** receives this file and script information, executes the selected Python model as a subroutine on the downloaded file, and displays real-time logs through a **terminal-style Tkinter GUI**.

Thus, this hybrid system effectively **combines cloud scalability (authentication, file storage)** with **local compute power** for heavy Excel-based data modeling.

5. Key Code Snippets and Technical Highlights

Frontend DriveData.tsx – Sending File and Script for Processing

```
const runModel = async (selectedScript: string) => {

    const response = await fetch(`${API_URL}/run-local-model`, {

        method: "POST",

        credentials: "include",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({ file_id: selectedFile, script: 
selectedScript}),

    });

};
```

This function represents the **handoff** from user action in the frontend to backend API processing, transmitting the necessary information (file ID, script name) to start local model execution.

Backend application.py – Running Local Helper Model

```
@app.post("/run-local-model")

async def run_local_model(request: Request, data: dict):

    file_id = data.get("file_id")

    script = data.get('script')

    creds = credential_handler.get_creds(request.session)

    file_path = drive.download_file(file_id, creds=creds)
    response = requests.post(f"{LOCAL_HELPER_URL}/upload-file", json={"file":
os.path.basename(file_path), 'script': script})
```

The backend **downloads** the selected file from Google Drive securely and **forwards** its information to the Local Helper server, managing session security and error handling throughout.

Local Helper local_helper.py – Processing the Uploaded File

```
@app.route("/upload-file", methods=["POST"])

def upload_file():

    file_path = request.json['file']

    py_script = request.json['script']

    run_model(os.path.join("local_data", file_path), py_script)

    return jsonify({"success": "File delivered."})
```

Once the Local Helper receives the uploaded file and script request, it **runs the Python model locally** using a subprocess, ensuring fast and system-compatible file processing.

6. Challenges and How We Overcame Them

Challenge	How We Solved It
Cross-Origin Resource Sharing (CORS) issues	Configured FastAPI CORS middleware to allow requests from the React frontend.
Tkinter and Flask blocking each other	Used Python `threading` to run Flask server in a background thread while keeping the GUI responsive.
Cross-platform file path issues (Windows vs macOS)	Standardized path management using `os.path` to ensure compatibility.
Secure token management for Google Drive API	Used session-based OAuth 2.0 management and automatic token refresh.
Packaging Local Helper into executable	Used PyInstaller to successfully build standalone `.exe` (Windows) files for simple deployment.

7. Future Work

Building on the initial success, Lightcast Consulting Cloud can be further enhanced by:

- Adding **drag-and-drop** functionality for file uploads in the frontend.
- Allowing **folder pinning** for commonly accessed datasets.
- Implementing **progress bars** and **file status indicators**.
- Extending support for **processing embedded Excel files in Word documents**.
- Exploring **full cloud model execution** for lightweight scripts to reduce local dependencies in the future.

8. Conclusion

The Lightcast Consulting Cloud project began the process of modernizing Lightcast's workflow into a **secure, scalable, and hybrid cloud-local platform**.
Through the use of modern frameworks such as **React**, **FastAPI**, and **Tkinter**, and by deploying the **frontend and backend to AWS**, we built a system that is **robust today** and **flexible for tomorrow**.

By combining **Google Drive authentication**, **local processing efficiency**, and **cloud storage scalability**, the project delivered not only on the original requirements but opened new doors for future innovation at Lightcast.
Positive client feedback, successful deployment demonstrations, and production readiness of the platform show that the solution met or exceeded all key project objectives.

9. Contributors

This project was successfully completed by the following team members:

- **Ian King**
- **Caleb Mouat**
- **Shashwot Niraula**
- **Andrew Plum**
- **Bibek Sharma**

10. GitHub Repository

For source code, technical documentation, and project files, please visit our GitHub repository.