NexGen Solutions

Introduction

For NexGen to work great, it needs good UI, great data capabilities, intelligent, thinks beyond the current scope or set of so called scope or resources of merit. Need to know the real need, real truth for the demand or need. Well, not to break rules; however the need to offer best solution in time for clients to take decisions is need and the supporting tools and technology stack is needed.

Gen-AI, ML, AI, Quantum, Block chain, V8 etc

Node.js and Chrome V8 make a great combination for browser based applications; this offers natural sandbox per process, direct parsing and machine code execution in target CPU, which makes a good development strategy for cloud applications that need scalability and speed of execution. Similar research is progressing with other variants that help remove lots of processing, calls and transportation of large datasets and that can deploy and administer full cloud stack assets using JavaScript. This avoids relearning newer scripts and tools used for deploying cloud assets.

Chrome V8 is a JavaScript Virtual Machine and Node.js is a cross-platform, open-source JavaScript runtime environment. I feel there are other libraries that work well like TypeScript, and React. With this the need to rely on backend systems can be reduced. Administrator can effectively generate, run, stop virtual machines and other cloud elements used in IaaS, PaaS and SaaS. It can run client scripts using browsers only and no need of rich GUI screens that take lots of local space. Moreover, JavaScript is also used in embedded and low memory devices

Front end driven by V8, JavaScript, and Node.js. React, NPM etc - GCP SaaS. Backend driven by Cloud compute, storage, network nodes with Google's cloud IaaS, PaaS, offering. The Middleware is driven by Bhadale's Cloud hub, proxies, for multi-cloud hybrid offerings that can integrate popular public clouds from AWS, GCP, Azure, etc. Solution Café is part of the front end and middleware that has global scope and can use GCP Spanner, other global scale services. This allows for decentralized, region specific controls and centralized governance as part of MNC global operations, allowing for good flexibility, local cache, and local control and also supervised at a global scale datacenter that can have only performance, gains, results data being exported overseas, and making data safer. These can be Global Command center with local operations that adhere to regional, state or national policies.

Most of the corporate structure are agile and can be restructured based on changing business needs, targets. This makes this more agile and also more result oriented.

Clients can be from a single person to a large MNC. Backed by popular cloud providers and partners especially Google will help make this really big. We, at Bhadale IT are offering a general framework that can be integrated with other cloud, especially GCP or as an open framework that will have to go through a refinement that allows newer cloud designs for truly global coverage and more than 50 industries can make use of this.

Points to consider

1. Write a Catalogue and show how Solution Cafe, Cloud Hub are part.
2. Show how these products can be considered futuristic using, AI, GenAI, QC and various use cases and human roles/ avatars, business types can be served.
3. Show how Gen AI can help or replace workers doing manual work. Say for example, I can create a coder Avatar to do coding using Google Cloud. Using various Labs, coder can complete the assignments as per Lab instructions. This can be automated using Workflows that mimic human NLP. For example, a user logs in, select course; complete a course video, quiz and lab. How each module is approached using QNLP and how. Another example a farmer, how he plans based on data, how he runs tractors, tools, etc. Can a team of robots replace the farmer's jobs?

4. So many of the mundane office jobs, transcription, data entry, office assistants, delivery boys, courier boys, pizza, food delivery jobs can be replaced. Off course the service men supervise the fleet of robots
5. What types of agent form factors do you have, physical, software only, systems, cyber/virtual etc. What are the roles these agents offer?
6. Use Industry 5.0 and human-machine classification and taxonomy that allows for special types of design and development to meet every use case and industry that plan to use AI, automation and intelligence to make improvements in productivity, automation and improve accuracy using the LLMs and speed using QC
7. Human machine, industry 5 based design of cloud systems, apps, infrastructure, software, agents, robots etc. Human touch, human context, ambient, location aware services.
8. Gen AI should obey rules of human machine engagement, interaction and not become aggressive in any condition. Emergent behaviors, diversions need approval. Human sensitive matters like religion, politics, crime etc should not emerge.
9. These can be part of governance unit and mandatory module for any AI related activities.
10. Roles based on human needs can be developed like postman, courier man, office boy, cook, etc can be designed. Specific templates can be developed and a set of rules +ML, Gen AI used to make a standard based app that can be either cloud deployed or physical unit deployed.
11. How these roles be used in projects you have for 35+ industries to make AI ready workforce that can de cloud based or standalone robot.
12. How QAI will speed up some large data processing steps, big data and empower robot handler to use that for the project.
13. Clients can start using cloud based projects, or use standalone and move other way. Finally fully connected cloud controlled, cloud spikes for scalability can be leveraged.
14. Finally how your project roles, agents can help you automate, improve productivity, give more time, and deliver better service.
15. How Gen AI can be used to restructure your business. How u set it as an example to clients to do same.
16. Exercise: Show how you restructure your Business divisions and map those to the IoC tool used to restructure IT assets, catalogues, products and services of each BU, retrain workers, and resolve internal, external issues.
17. Is this not an Agile Company at all levels? Don't use big jargons like PMI, Sys Engg., and SE etc. Keep the steps simple and good for any analyst to operate the business.
18. How Gen AI offer co-design features, across emerging technologies, QC, AI, ML, Blockchain etc.
19. Exercise: Do sample chat and gather details. Don't upload your templates or ideas.
20. How does Gen AI refer to private Books, company confidential data and its content? How to use that content in your job.
21. Show how your Corporate Technology stack is restructured based on changes. How hard coding avoided.
22. How can the restructured keep learning from daily operations and improve for better quality
23. How you add AI, QC, Automation to all projects that you have. Make an excel sheet to show how our projects are personalized using emerging technologies. How they can be upgraded, remote controlled.
24. How existing industries pain points are addressed. Make columns to projects excel to fit above features, pain points, future features, benefits, tech used etc.
25. What is the common framework used to mix technologies that can be customized as per use- cases. What common framework for projects used, is it industry framework or your own? .
26. Which RD center will give approval of this project? Endorsements
27. What data endorsers need? What is the benefit?
28. Assume you have 1 million qubits. How would you use that? How would you build a scalable real time pipeline with least latency? What tools you will use that can handle batch and real time events.
29. How classical modules preprocess data so that QC with short life operations say 1 minute will produce results with 1 million qubit operations.

30. What classical hardware, CPU, GPU can handle this? Assume million qubits are platform agnostic, how you will use them, once generated within 1 minute without storing them, especially logical qubits may be lesser by 25 percent.
31. What hardware will handle 1 million operations, especially classical hardware, non blocking software? Is Linux preferred? What are the QPU that has HPC capacity? If not what new design required.
32. How will your platform scale to handle more than 1 million qubits, 1 million operations per sec, what cluster, horizontal, vertical scalability required. Will cloud platforms support this or we need an inhouse configuration team to overseas extreme computing scales. How will you monitor speed, power, spending and ensure the data center works well.
33. How are QAI products developed considering human machine interactions? Does industry 5.0 offer a framework for this.
34. How are QAI products developed considering easier integration with multi cloud platform. What are open stds used
35. What GCP products can I use for above.
36. How are QAI products deployed and provisioned to cloud client tenant web sites, address, end points in a data center or a target device like mobile, laptop, server etc
37. If I want to build a mini datacenter in house and deploy all above for private cloud offering
38. How can I build a private cloud in premise data center while connecting to GCP public cloud. I would burst load to Public cloud when required. What type of client and server machines are required, let me know for the operating system, VM types, memory etc required for the IaaS stack.
39. What is the best way to design multi-cloud platform for deploying QAI products. What types of Networking, Virtual Machines, Portability, and Security are required to make this possible? I plan to use GCP, Azure and AWS cloud platforms.
40. What are the ways to map the classical data to Hilbert space and Fock space for quantum computing?
41. What types of linear and non-linear functions using feature maps, or kernels that map classical data to quantum Hilbert space is required for encoding, making classical data compatible for the quantum computing node
42. What benefits are possible using Variational Quantum Splines, SVM / Kernel, quantum embeddings, and squeezed states in designing classical-quantum hybrid systems
43. A recent study at The University of Basel, provides promise of 1 million qubits using holes. Can you explain how these can help in the development of Universal quantum system
44. Silicon FinFETs hosting hole spin qubits at temperatures over 4 Kelvin
45. Relook at the types of use cases for industries using QAI as key technology drivers, backed by allied like Block chain, Distributed computing, QInternet, HPC, Mainframe, Supercomputers etc.
46. See how you can create physical and virtual agents, robots that can be trained to use our products and services.
47. It is like a fleet of robots, drones, AI virtual agents, chatbots, RPA, all connected to a corporate DB, Knowledge base, digital assets that ensure a ready to use corporate with only humans as supervisor. Onboarding to our Datacenter NexGen solutions will allow clients to use subscription or upfront purchase of our racks with private cloud and hi speed NW to public cloud.
48. With this the investments are loaned or pay as you go mode, offering features for end client productivity, metering and billing.
49. Rework on the products and services they make good use of QAI, being modular and SoA based, easily integrated to popular cloud platforms.
50. As quantum needs subzero temps like 4-10 Kelvin, how you plan to embed quantum QPU in an ordinary device like laptop, mobile, or even a datacenter server
51. Search for Quantum dots working at room temps
52. Show the circuit to work with CPU, QPU, TPU, GPU as computing nodes and how a hi level orchestrator manages workloads locally and distributed nodes.

53. What the motherboard looks like using QRAM, bus, sw modules etc.
54. Min qubits needed to do real computing, starts at 1000 qubits scales to 1 million.
55. What all can u do if 1000 qubits given and 10000, 1 million.
56. How will your products scale and able to run bigger workloads.
57. How Quantum, AI, ML, classical tasks are identified or marked ensuring they reach proper nodes.
58. How will your Universal QC look like with all these?
59. How will the UQC work on all OS, targets, Devices
60. If multiple UQC are present how they scale.
61. Show GUI for cloud based products, web parts that call API calls to fulfill a use cases, how facets are designed grouping needs of a business, performance, quality etc. Controls and user can use to make economic, costly, hi performance, calls.
62. What results expected how these pages are developed in Cloud env. GCP.
63. How your guidance talks to your hub, and multi cloud, multi hops to fulfill.
64. How these products can be installed from browser, standalone, LinkedIn to browser for offline work. IDE for your GUI.

Actions:

- Try to get a good structure for IT services and research with a holding company structure with subsidiaries
- Identify the key areas for improvements that you can query for with Gemini.
- Offer a pictorial restructuring process diagram for the current Bhadale IT org diagram, Wix website based improvements in product and service offerings and internal company data like folders, images, content, etc
- Draw sketch from the recommended improvements?
- Complete drafting of the Catalogue from the above restructured and improved company org, content, etc
- You can make this another NexGen PPT slides , highlighting how you used Gemini in making your company better
- See who can offer you insights on the QAI features that you plan to add to your products. What standards can you use, how to use that, how well your products work with the multi-cloud platforms?

References:

1. https://v8.dev/
2. https://lnkd.in/gEH8z7XS
3. https://lnkd.in/giRRgwBq
4. https://lnkd.in/gH6WXWbc
5. https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-022-00314-5
6. https://hgs.cx/blog/industry-5-0-connecting-humans-and-technology-for-sustainable-growth/
7. https://www.frontiersin.org/articles/10.3389/frobt.2022.997386/full
8. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9578547/
9. https://camunda.com/blog/2023/05/human-workflow-orchestration-generative-ai-openai/
10. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9578547/
11. https://resources.system-analysis.cadence.com/blog/msa2021-using-finfets-vs-mosfets-for-ic-design
12. https://www.cell.com/the-innovation/pdf/S2666-6758(22)00126-6.pdf
13. https://www.nature.com/articles/s41567-024-02481-5
14. https://buff.ly/4dxfyFm
15. https://www.google.com/url?sa=t&source=web&cd=&ved=2ahUKEwiAhlnmgfyFAxWAS2cHHe-kBs0QFnoECBIQAQ&url=https%3A%2F%2Fedoc.unibas.ch%2F37038%2F1%2FPrechtel_Thesis%2520edoc.pdf&usg=AOvVaw0Egf-kHNN8P58V7FqOFc9d&opi=89978449

/// Appendix

Gemini Prompts

How can quantum AI, quantum ML used to improve products and services. For example, I have a precision farming app and I need to use quantum and AI in this products. Please let me know how to go about this.

Quantum AI (QAI) and Quantum Machine Learning (QML) are emerging fields with the potential to revolutionize how we develop products and services. Here are a few ways they can lead to improvements:

- **Faster and More Accurate AI Models:** Traditional AI struggles with complex problems involving massive datasets. QML algorithms can analyze these problems much faster by exploiting the unique properties of qubits, the building blocks of quantum computers. This allows for creating more accurate and powerful AI models for tasks like medical diagnosis, [1]
- **Optimization on a New Level:** Quantum computers excel at solving optimization problems. This makes them ideal for optimizing various aspects of product development and service delivery. [2]
- **Unleashing the Power of Big Data:** Many industries deal with enormous datasets that are currently too complex for traditional computers to handle effectively. QAI can process this data much faster, leading to discovering hidden patterns and trends that can be used to improve products and services. This could be crucial in fields like drug discovery, [3]
- **Revolutionizing Material Design:** Developing new materials with specific properties is a time-consuming and expensive process. QML can accelerate material simulations, allowing scientists to test a wider range of possibilities and discover novel materials with superior properties for future products.
- **Enhanced AI models:** QML algorithms excel at tackling complex optimization problems. This can lead to significantly faster training times for AI models and the ability to handle datasets too intricate for classical computers. Imagine AI that personalizes products or services to an entirely new level of detail, taking into account a vast array of factors.
- **Faster drug discovery:** Simulating complex molecules is a perfect example. Today's AI struggles with this, but QML could accelerate drug discovery by efficiently modeling molecular interactions and optimizing material properties for new drugs and materials.
- **Improved financial modeling:** The financial sector relies heavily on simulations and risk analysis. QML can speed up calculations like Monte Carlo simulations, leading to more accurate financial modeling and potentially revolutionizing investment strategies.
- **Optimized logistics and manufacturing:** Quantum computers can tackle complex logistics problems like routing optimization or the traveling salesman problem, leading to more efficient supply chains and manufacturing processes.
- **Logistics and delivery:** Finding the most efficient routes for delivery trucks, reducing costs and emissions.
- **Personalized medicine:** Analyzing a patient's individual genetic data to develop more targeted treatments.

Here's a general roadmap to get you started:

Product Design:

Focus on a specific problem: QAI/QML excel at solving complex optimization problems. Start by identifying a specific pain point in precision farming like optimizing fertilizer application,
Classical + Quantum Approach: Most effective applications will likely combine classical machine learning with QAI/QML.
Focus on specific use cases: Don't try to solve everything at once. Identify a specific aspect of precision farming where QAI/QML can offer a significant advantage, such as optimizing crop yield based on real-time sensor data or predicting disease outbreaks.

Focus on data collection and integration: QAI and QML need high-quality data to function effectively. Design your application to collect data from various sources like sensors, drones, and satellite imagery. Integrate seamlessly with existing farm management systems to leverage historical data.
Modular design for future scalability: The field of quantum computing is rapidly evolving. Design your application with a modular architecture so that you can easily integrate new QAI and QML algorithms as they become available.

Coding Languages:

- **QAOASM and Cirq:** While QAI/QML are still evolving, languages like QAOASM (Quantum Assembly Language) and Cirq (developed by Google) are being used to write quantum algorithms. Python libraries like TensorFlow Quantum are also being developed.
- **High-level languages with quantum libraries:** While QML is still evolving, high-level languages like Python and Julia are gaining traction due to the availability of quantum computing libraries like TensorFlow Quantum, Cirq, and PennyLane. These libraries provide a user-friendly interface for developing and running QML algorithms.
- **Python libraries:** Several Python libraries like Cirq, TensorFlow Quantum, and PennyLane are emerging for developing quantum circuits and algorithms. These libraries can be helpful for prototyping and testing your QAI and QML models.

Hardware:

- **Cloud Access to QPUs:** Quantum computers are expensive and complex to maintain. Cloud platforms like Google Cloud, Amazon Braket,
- **Classical CPU Considerations:** Even with a QPU, your application will rely heavily on classical computation for data processing and machine learning tasks.
- **Classical CPUs for pre-processing and post-processing:** QPU work best for specific tasks. Classical CPUs are still essential for data pre-processing, post-processing, and running the rest of your application logic.

Virtualization:

- **Quantum Cloud Access:** Since you'll be using cloud-based QPUs, virtualization is already handled by the cloud provider. Focus on choosing a cloud platform that offers good access to QPUs and integrates well with your chosen coding languages and frameworks.
- **Containerization:** Containerization technologies like Docker can help package your application and its dependencies, making it easier to deploy across different cloud environments.

Data Security:

//

How to Implement Modular Design:

1. **Identify Core functionalities:** Start by defining the core functionalities of your application, such as data collection,
2. **Break Down QAI/QML tasks**: Break down the specific tasks where you want to leverage QAI and QML. This could include tasks like optimizing fertilizer application,
3. **Develop Independent Modules:** Develop each core functionality and QAI/QML task as independent modules with well-defined interfaces. These interfaces should specify how data is exchanged between modules and how they interact with each other.
4. **Standardization:** Ensure all modules follow consistent coding standards and data formats.
5. **Orchestration Layer:** Develop an orchestration layer that manages the flow of data between modules and triggers the execution of specific QAI/QML models based on user needs and data analysis.

Benefits of Modular Design:

- **Easier Integration of New Algorithms:** By breaking down your application into independent modules, you can integrate new QAI and QML algorithms as they become available without needing to rewrite the entire application. Each module can be responsible for a specific task, such as:
  - Data pre-processing and cleaning
  - Feature engineering for QML models
  - Running QML algorithms on the QPU
  - Interpreting QML results
  - Integrating insights into actionable recommendations for farmers

- **Faster Development Cycles:** Modular design allows for independent development and testing of each module. This can significantly speed up development cycles, especially when working with cutting-edge technologies like QAI and QML

  Improved Maintainability: Modular applications are easier to maintain and troubleshoot. If there's an issue with a specific module, you can isolate and fix it without affecting the entire application.

How to Implement Modular Design:

- **Identify Core functionalities:** Start by identifying the core functionalities of your application, such as data management, model training (classical and quantum), and generating recommendations.
- **Break down into modules:** Break down these functionalities into independent, well-defined modules with clear interfaces for communication. Each module should have a specific responsibility and well-documented APIs for interaction with other modules.
- **Use Standardized Interfaces:** Define standardized interfaces between modules to ensure they can work together seamlessly, even if the underlying implementation details change in the future.
- **Focus on Loose Coupling:** Strive for loose coupling between modules. This means that changes in one module should have minimal impact on other modules. This allows for independent development and easier integration of new functionalities.

Examples of Scalability with Modular Design:

- As new QML algorithms become available, you can simply swap out the existing QML module with the new one, as long as the interfaces remain compatible.
- If you need to handle a larger volume of data in the future, you can scale the data processing module by adding more processing power without affecting other modules.
- You could even develop new modules for specific crops or farming practices, all within the larger framework of your core application.

Core Components:

- **Data Acquisition Module:** This module would be responsible for collecting data from various sources relevant to precision farming. This could include:
  - Sensor data from weather stations, soil moisture sensors, and plant health monitors.
  - Drone and satellite imagery for field analysis.
  - Historical data from farm management systems on crop yields, resource usage, and past decisions
- **Data Processing Module:** This module would clean, organize, and pre-process the collected data to prepare it for analysis. It might involve:
  - Removing outliers and inconsistencies from the data.
  - Feature engineering to create new data points relevant for QAI and QML models.

    - Transforming data into a format compatible with QPU execution
- **Classical Processing Module:** This module would handle tasks best suited for traditional CPUs. It could include:
    - Machine learning algorithms for initial data analysis and filtering.
    - Classical optimization techniques for solving problems where QAI doesn't offer a significant advantage.
    - User interface and data visualization tools for farmers to interact with the application.
- **Quantum Processing Module:** This module would handle specific tasks where QAI and QML algorithms offer a significant benefit. For example:
    - Optimizing irrigation schedules based on complex weather and soil moisture data.
    - Identifying optimal crop planting patterns and resource allocation.
    - Analyzing genetic data to predict crop yields and disease resistance.

Modular Connections and Scalability:

- **Standardized Interfaces:** Each module should have well-defined interfaces that specify the data it receives and outputs. This allows for easy integration of new modules in the future.
- **Loose Coupling:** Modules should be loosely coupled, meaning they should operate independently with minimal reliance on the internal workings of other modules. This simplifies maintenance and upgrades.
- **API Integration:** Develop a well-documented API (Application Programming Interface) for each module. This allows external developers to create new tools or services that can integrate with your application, further expanding its functionality.

Here's a breakdown of how to structure your design document and some sample content:

1. Introduction

- Briefly describe the purpose of the application and the problem it aims to solve for farmers.
- Highlight the role of QAI and QML in the application and the specific benefits they offer.

2. System Architecture

- Here, showcase the modular design using a diagram. Each module should be represented by a box with clear labels indicating its function. Arrows between modules should depict the flow of data.

2.1 Module Descriptions

- Dedicate a section for each module, detailing:
    - **Module Name and Function:** Briefly describe the module's purpose and the tasks it performs.
    - **Inputs and Outputs:** Specify the data formats the module receives and the data it generates.
    - **Technologies:** Mention the programming languages, libraries, or frameworks used for development within the module.
    - **APIs:** If the module exposes an API, document its functionalities and data formats.

Sample Content (Data Acquisition Module):

- **Module Name and Function:** Data Acquisition Module
- **Inputs:** None
- **Outputs:** Sensor data streams,
- **Technologies:** Python libraries for interacting with sensors and data APIs.
- **APIs:** Provides an API for external systems to push data into the application.

3. Data Flow

- Describe the flow of data through the system. Explain how data is collected by the Data Acquisition Module, pre-processed by the Data Processing Module, analyzed by the Classical Processing Module and Quantum Processing Module (where applicable), and finally presented to the user through the User Interface Module.

4. Scalability Considerations

- Explain how the modular design facilitates future scalability. Mention how new modules for advanced QAI/QML algorithms or integration with other services can be incorporated easily.

5. Security

- Outline the security measures implemented to protect sensitive agricultural data. This could include data encryption, access controls, and secure communication protocols.

6. Future Enhancements

- Briefly discuss potential future advancements for the application. This could involve integration with new sensor technologies, adoption of more sophisticated QAI/QML models, or expansion to new agricultural domains.

//

Data pipelines for QAI and QML in precision farming applications involve orchestrating classical data processing tasks with quantum function execution.

1. Data Preprocessing:

- The pipeline starts with classical data acquisition from sensors, drones, and historical farm management systems as mentioned earlier.
- This data is then cleaned, formatted, and transformed into a state compatible with QPU execution. Classical libraries like Pandas, Scikit-learn are helpful for data cleaning and feature engineering.

2. Classical Machine Learning for Initial Analysis:

- The pipeline might utilize classical machine learning algorithms for initial data filtering and analysis. This could involve tasks like:
  - Identifying patterns and trends in the data.
  - Classifying data points into relevant categories.
  - Generating preliminary insights to guide further analysis

3. Feature Engineering for Quantum Models:

- Classical techniques are used to create new data features specifically designed for QAI and QML models. This may involve:
  - Representing data points as qubits (quantum bits) suitable for quantum circuits.
  - Encoding relevant information from the data into quantum states.

4. Quantum Function Execution:

- Once the data is prepared, specific sections of the pipeline trigger QPU execution. QML algorithms written in QAOASM or using libraries like Cirq are submitted to the cloud-based quantum service provider (like Google Cloud, Amazon Braket, or IBM Quantum Cloud).

5. Post-Processing and Classical Integration:

- The results obtained from the QPU are probabilistic in nature and often require post-processing. Classical algorithms interpret the quantum outputs and translate them into actionable insights for farmers.
- These insights might be combined with the results from the initial classical analysis to provide a more comprehensive understanding of the agricultural system.

6. Data Visualization and User Interface:

- The final stage involves presenting the processed data and insights to the farmer through a user-friendly interface. This could include visualizations like charts, graphs, and actionable recommendations for optimizing farm operations.

Overall, data pipelines for QAI and QML in precision agriculture combine classical data processing techniques with the power of quantum computing. By carefully integrating these workflows, you can develop a powerful application that leverages the strengths of both paradigms to deliver valuable insights for farmers.

Alternative responses:

1. Pre-processing and Feature Engineering (Classical):

- The data pipeline starts with classical data processing modules.
- Raw data from sensors, drones, and historical records is collected by the Data Acquisition Module.

Data Processing Module then cleans and pre-processes the data. This may involve:

- Removing outliers and inconsistencies.
- Transforming data into a format suitable for further analysis.
- Performing feature engineering to create new data points relevant for QAI and QML models. For instance, combining sensor readings from multiple sources to create a more comprehensive picture of crop health.

2. Classical Machine Learning for Initial Analysis (Classical):

- The pre-processed data is then fed into classical machine learning models. These models can handle various tasks like:
  - Identifying patterns and trends in historical data to predict crop yields or potential disease outbreaks.
  - Filtering and prioritizing data to identify situations that might benefit most from QAI/QML algorithms.

3. Data Preparation for Quantum Processing (Classical):

- Data points relevant for the specific QAI/QML algorithms are then selected and transformed. This may involve:
  - Selecting specific features (e.g., soil nutrient levels, weather data) most impactful for the chosen QML model.
  - Encoding the data into a format compatible with QPUs. For instance, converting numerical data points into qubits (quantum bits) which are the basic units of information in quantum computers.

4. Quantum Processing (Quantum):

- The prepared data is then transferred to a cloud-based Quantum Processing Unit (QPU) accessible through APIs provided by cloud platforms like Google Cloud, Amazon Braket, or IBM Quantum Cloud.

- QPU executes the QAI/QML algorithms to perform tasks like:
    - Optimizing resource allocation (water, fertilizer) based on complex data combinations.
    - Identifying optimal planting patterns and crop rotations considering various factors.
    - Analyzing genetic data to predict disease resistance or crop yields with higher accuracy than classical models.

5. Post-Processing and Integration (Classical):

- The results obtained from the QPU are sent back to the classical pipeline.
- These results might require further processing, such as decoding the data from qubits back to a human-readable format.
- The classical processing module then integrates the QAI/QML insights with the outputs from the initial classical analysis.

6. Visualization and Decision Making:

- Finally, the combined results are presented to the farmer through a user-friendly interface. This may involve visualizations like charts, graphs, or maps.
- Farmers can then use these insights to make informed decisions about their farming practices, potentially leading to increased yields, reduced resource usage, and improved crop health.

Integration Challenges and Considerations:

- One challenge is ensuring smooth data flow between classical and quantum parts of the pipeline. Data needs to be efficiently transferred between different processing units while maintaining accuracy and security.
- Another challenge is optimizing the workflow to minimize the time spent on classical pre-processing and post-processing compared to the actual QPU execution time. QPU resources are still expensive, so maximizing their utilization is crucial.

Classical techniques handle initial data analysis and feature engineering, while QAI/QML algorithms provide unique problem-solving capabilities for specific tasks in your precision farming application.

Alternate soln:

1. Data Preprocessing:

- **Classical Processing:** Most data preprocessing will likely be handled by classical modules. This could involve tasks like:
    - Data cleaning to remove noise and inconsistencies.
    - Feature engineering to create new data points relevant for QAI and QML models.
    - Data transformation to ensure compatibility with the specific requirements of QPU execution (e.g., converting data into sparse matrices).

2. Workflow Integration:

- **Modular Design Advantage:** A modular architecture is beneficial here. The classical preprocessing module can output data in a format easily consumed by the quantum processing module.
- **Pipeline Orchestration Tools:** Utilize tools like Apache Airflow, Luigi, or Kubeflow to orchestrate the data pipeline. These tools can handle task scheduling, dependency management, and error handling, ensuring smooth execution of classical and quantum stages.

3. Classical-Quantum Handoff:

- This is a crucial step where preprocessed data is transferred from the classical module to the quantum processing module. There are two common approaches:
  - **Feature Selection:** The classical module can identify a subset of features most relevant for the QAI/QML model. This reduces the data size sent to the QPU, improving efficiency.
  - **Feature Encoding:** Classical processing can encode the data into a format compatible with QPU circuits. This may involve techniques like quantum feature mapping.

4. Quantum Processing:

- The QPU executes the QAI/QML algorithms on the preprocessed data.
- The choice of algorithm depends on the specific problem you're trying to solve in your precision farming application.

5. Post-Quantum Processing:

- Once the QPU outputs the results, they are sent back to the classical module for post-processing. This may involve:
  - Decoding the results from the quantum format back into a classical format.
  - Integrating the quantum results with other data from the classical processing modules.
  - Applying classical machine learning algorithms to further refine the results.

Certainly, here's a basic example using Cirq and TensorFlow Quantum (TFQ) to showcase a **classical-quantum pipeline** for a simple fertilizer recommendation task:

```
import cirq
import tensorflow as tf
import tensorflow_quantum as tfq

# Define a circuit for fertilizer recommendation
def fertilizer_circuit(data):
  q0 = cirq.GridQubit(0, 0)
  q1 = cirq.GridQubit(0, 1)

  # Apply classical data to qubits (replace with feature encoding in practice)
  circuit = cirq.ry(data[0])(q0)
  circuit += cirq.rx(data[1])(q1)

  # Apply a Quantum Machine Learning algorithm (replace with your specific QML model)
  circuit += cirq.H(q0)
  circuit += cirq.CNOT(q0, q1)
  circuit += cirq.rx(data[2])(q1)
  circuit += cirq.H(q1)

  # Measure qubits
  circuit += cirq.measure(q0, key='recommended_fertilizer')

  return circuit

# Sample data (replace with actual sensor data from your application)
data = [0.2, 0.4, 0.8]  # Hypothetical features for soil nitrogen, phosphorus, potassium
```

```
# Convert circuit to TensorFlow tensor
qcircuit = tfq.convert_to_tensor(fertilizer_circuit(data))

# Define a classical model to post-process quantum results (replace with your model)
def classical_postprocessing(measured_data):
  recommended_fertilizer = measured_data['recommended_fertilizer']
  # Apply classical logic based on the measured fertilizer qubit
  # (e.g., high probability = recommend fertilizer)
  return recommended_fertilizer

# Create a TensorFlow Quantum Model
model = tfq.models.CircuitModel(qcircuit)

# Compile the model (choose appropriate optimizer based on your QML algorithm)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01))

# Run the data pipeline (replace with data from your acquisition module)
measured_data = model.predict(data)

# Apply classical post-processing
recommended_fertilizer = classical_postprocessing(measured_data)

print(f"Recommended fertilizer amount: {recommended_fertilizer}")
```

This code demonstrates a basic structure for a classical-quantum data pipeline using Cirq and TFQ. You'll need to replace the sample data, feature encoding logic, and classical post-processing with your specific application requirements and chosen QML model.

// Alternative

Here's a basic example using Cirq and TensorFlow Quantum (TFQ) to demonstrate a **classical-quantum pipeline** for a simple machine learning task in Python. This example doesn't directly relate to precision farming, but it showcases the core concepts of data processing, circuit creation, and integration with classical models.

```
import cirq
import tensorflow as tf
import tensorflow_quantum as tfq

# Define a simple circuit (replace this with your QML algorithm)
def create_classifier_circuit(x):
  q0 = cirq.GridQubit(0, 0)
  circuit = cirq.Circuit(cirq.H(q0), cirq.rx(x * np.pi / 2.0)(q0))
  return circuit

# Generate some sample data
data = np.random.rand(10, 1)  # 10 samples, 1 feature each

# Classical preprocessing (replace this with your data processing steps)
scaled_data = 2 * data - 1  # Scale data to +/- 1 range
```

```
# Convert data to Cirq circuits
circuits = [create_classifier_circuit(x) for x in scaled_data]

# Define a quantum simulator (replace this with QPU access for real application)
simulator = tfq.simulator.TFQSimulator()

# Execute the circuits on the simulator
results = simulator.run_batch(circuits, repetitions=10)

# Get the bitstrings from the results
bitstrings = results.measurements['m']

# Convert bitstrings to labels (replace this with your post-processing logic)
labels = np.where(bitstrings[:, 0] == 1, 1, 0)

# Train a simple classical model (replace this with your main machine learning model)
model = tf.keras.Sequential([
  tf.keras.layers.Dense(1, activation='sigmoid'),
])
model.compile(loss='binary_crossentropy', optimizer='adam')
model.fit(scaled_data, labels, epochs=10)

# Use the trained model for prediction on new data
new_data = 0.7  # Example new data point
new_circuit = create_classifier_circuit(new_data)
new_result = simulator.run(new_circuit, repetitions=10)
new_bitstring = new_result.measurements['m'][0]
predicted_label = np.where(new_bitstring == 1, 1, 0)

print("Predicted label for new data:", predicted_label)
```

This code demonstrates:

1. **Classical Preprocessing:** Data is scaled to a specific range.
2. **Circuit Creation:** The `create_classifier_circuit` function defines a simple quantum circuit based on the input data point. (Replace this with your actual QML algorithm).
3. **Data to Circuit Conversion:** A loop iterates through the data, creating Cirq circuits for each data point.
4. **Quantum Processing (Simulation):** Circuits are executed on a TensorFlow Quantum simulator. (Replace this with actual QPU access in a real application).
5. **Result Acquisition:** Measurement outcomes (bitstrings) are retrieved from the simulation results.
6. **Post-Quantum Processing:** Bitstrings are converted to labels (replace with your application-specific processing).
7. **Classical Model Training:** A simple classical model is trained using the generated labels.
8. **Prediction on New Data:** The trained model is used to predict the label for a new data point using a newly created circuit.

Alternative

You'll need to install Cirq (`pip install cirq`) and TensorFlow Quantum (`pip install tensorflow-quantum`) before running this code.

```
import cirq
import tensorflow as tf
import tensorflow_quantum as tfq

# Define a simple quantum circuit for classification
def classifier_circuit(data):
  qubits = cirq.LineQubit.range(2)  # Two qubits for this example
  circuit = cirq.Circuit(
     cirq.H(qubits[0]),  # Apply Hadamard gate to first qubit
     cirq.rx(data[0], qubits[1]),  # Apply rotation gate with datapoint as angle
     cirq.CNOT(qubits[0], qubits[1])  # Apply CNOT gate
  )
  return circuit

# Sample data for classification (replace with your actual data)
data = tf.constant([0.5])

# Create a quantum simulator
simulator = tfq.simulator.TFQSimulator()

# Build the QML model using Cirq circuit and data
model = tfq.layers.CircuitLayer(classifier_circuit)
q_circuit = model(data)

# Simulate the quantum circuit and get measurement results
results = simulator.run(q_circuit, repetitions=100)
measured_bits = results.measurements['m']

# Process the measurement results (replace with your classification logic)
predicted_class = tf.math.reduce_mean(measured_bits[:, 0])  # Assuming bit 0 holds the class info

# Print the predicted class (for demonstration purposes)
print("Predicted Class:", predicted_class.numpy())
```

This code defines a simple quantum circuit that takes a data point as input and applies various gates. The circuit is then executed on a simulator, and the measurement results are used for classification (which is a simplified example here). In a real application, you would replace this with your specific QAI/QML algorithm for your precision farming problem.

A full QML pipeline would involve data preprocessing, classical feature engineering, potentially more complex quantum circuits, and post-processing of the quantum results for integration with classical machine learning models.

Here are some additional resources you might find helpful as you continue exploring this field:

- **Cirq Documentation:** https://quantumai.google/cirq
- **TensorFlow Quantum Documentation:** https://www.tensorflow.org/quantum/install

- **Applications of Quantum Machine Learning:** https://arxiv.org/pdf/2209.09590 (This is a research paper but a good starting point)

Docs to Share:

https://docs.google.com/document/d/19jVP0yk8vQePLWcnvBMQ1_oH3gL8uI_zY71RNMHxpLQ/edit?usp=sharing

https://docs.google.com/document/d/1vypQ6uURVJdkZOdYvUMILGvoVDlBqIqlHTRr46Kbpok/edit?usp=sharing

Gemini:
https://gemini.google.com/app/7e6869c9ec38b238

Generative AI has great potential for business and research projects.

For example, I was trying to get instructions for developing a project that I am not so familiar, like precision agriculture. I received informative responses that were well structured and good to start. Also code generation was satisfactory to get the basic structure of the data pipeline using QAI / QML. I also prompted for design document and got good content. All these were not possible earlier. I have enclosed two links to Google docs with response and code.

Other Generative AI tools, chatbots, agents can boost business and research work. Various types of suggestions related to new projects, migration projects, performance improvements, agile projects and multi-cloud, classical-quantum data pipeline.

https://docs.google.com/document/d/e/2PACX-1vT5-CVqd5MgXSKg0CGRb1SNqFLtov9gFyJpxOU0Lr1zgID0xGyJok1pbX3ldppxh1V77md2PXF88V30/pub

https://docs.google.com/document/d/e/2PACX-1vQlFqA12YRp3-Qo8k4lHYa7fleSiVgmXISE2gyO3ahBak2pJHq47kbC3G5CTwjOV1NLevAeVwdhSkrl/pub

Utilizing 1 million qubits is an ambitious goal, considering current technological limitations. However, let's explore how we might approach building a real-time pipeline with such a system,

Hybrid Quantum-Classical Architecture: Given the potential for decoherence and error correction complexities, a hybrid approach is likely necessary. This would involve:

- **Classical Pre-processing and Post-processing:** Leverage classical computers for data preparation, task scheduling, and result analysis to minimize the time data spends in the error-prone quantum realm.
- **Quantum Processing Unit (QPU):** Utilize the 1 million physical qubits for specific computations where quantum algorithms offer significant advantages. However, be aware that due to limitations in qubit coherence times and error correction, the number of logical qubits available for computation might be significantly less than the physical qubit count.

Real-time Considerations:

- **Minimizing Latency:**
  - **Quantum Circuit Compilation:** Optimize quantum circuits for the specific QPU architecture to minimize execution time.
  - **Batching Similar Tasks:** Group similar tasks together to reduce the overhead of switching between computations on the QPU.

- **Error Correction and Fault Tolerance:** Implement robust error correction protocols to maintain the integrity of computations despite potential errors in physical qubits.

Tools for Batch and Real-time Events:

Quantum Cloud Platforms: Services from companies like IBM Quantum, Google Quantum AI, and Rigetti Computing offer access to QPUs and tools for developing and running quantum circuits. These platforms often provide:

- **High-Level Programming Languages:** Tools like Qiskit (IBM), Cirq (Google), and OpenQASM allow writing quantum algorithms in a more user-friendly way compared to directly manipulating qubits.
- **Job Queuing Systems:** Manage the execution of quantum circuits, potentially batching similar tasks for better resource utilization.

Classical Stream Processing Engines: For real-time data processing on the classical side, consider frameworks like Apache Flink or Apache Kafka Streams. These can handle high-volume, real-time data streams and integrate with quantum computing platforms through APIs.

Alternative answer

Scalable Real-Time Pipeline:

1. **Hybrid Quantum-Classical Approach:**
   - Leverage classical computers for pre-processing and post-processing data. Classical computers can efficiently handle tasks like data filtering, feature engineering, and result interpretation.
   - Utilize the 1 million physical qubits for the core quantum computation portion of the pipeline. This could involve tasks like solving optimization problems, simulating complex systems, or performing machine learning tasks with quantum algorithms.
2. Pipeline Breakdown:

   Real-Time Data Ingestion:

   - Utilize real-time streaming frameworks like Apache Kafka or Apache Flink to ingest data with minimal latency.
   - Perform initial data cleaning and pre-processing on the classical side before feeding it to the quantum circuit.

   Quantum Circuit Execution:

   - Design a modular quantum circuit that can handle smaller chunks of data at a time. This allows for parallelization and reduces the overall execution time on the quantum computer.
   - Use techniques like quantum circuit optimization to minimize the number of gates required, further reducing latency.

   Result Retrieval and Post-Processing:

   - Retrieve the results from the quantum computer and send them back to the classical side.
   - Perform post-processing tasks on the classical side to interpret the results and generate insights.

Tools for Batch and Real-Time Events:

Classical Tools:

- Utilize existing frameworks like Apache Spark or Flink for distributed processing and real-time analytics on the classical side.
- Consider libraries like TensorFlow or PyTorch for classical machine learning tasks that can complement the quantum computation.

Quantum Tools:

- High-level quantum programming languages like Cirq, Qiskit, or PennyLane can be used to design and optimize quantum circuits.
- Consider cloud-based platforms like Google Quantum AI, Amazon Braket, or IBM Quantum that offer access to quantum hardware and development tools.

Minimizing Latency:

- **Parallelization:** Break down the problem into smaller sub-problems that can be tackled by the quantum computer in parallel.
- **Quantum Circuit Optimization:** Use techniques like gate decomposition and compiler optimizations to minimize the number of gates in the quantum circuit.
- **Efficient Data Transfer:** Utilize high-bandwidth communication channels between the classical and quantum parts of the pipeline.

Alternative response

Scalable Real-Time Pipeline with 1 Million Qubits:

- **Hybrid Quantum-Classical Architecture:** We wouldn't dedicate all 1 million qubits to the core computation. Instead, we'd utilize a hybrid architecture. Here's a breakdown:
    - **Classical Processor:** Handles data pre-processing, task scheduling, error correction, and post-processing of quantum results.
    - **Quantum Processing Unit (QPU):** A smaller subset of the 1 million qubits (logical qubits) would perform the actual quantum computations. The number of logical qubits depends on the specific hardware's capabilities and the chosen error correction scheme.

- **Real-Time vs. Batch Processing:** This pipeline could handle both real-time and batch events.

    - **Real-time events:** Prioritize a subset of logical qubits for real-time tasks. Classical processing would analyze incoming data and determine if specific computations benefit from immediate quantum processing. Here, minimizing latency is crucial.

    - **Batch events:** Utilize the remaining logical qubits for batch processing. This could involve accumulating data over a period and then running complex quantum algorithms on it. Batch jobs may tolerate higher latency compared to real-time tasks.

Tools for Scalability and Latency Reduction:

- **Quantum Cloud Platforms:** Services like IBM Quantum, Google Quantum AI, and Rigetti Computing offer access to quantum hardware and development tools. They provide libraries and frameworks optimized for their specific hardware, allowing you to design and run quantum circuits.
- Quantum **Circuit Compilation Tools:** Tools like Terra https://www.ibm.com/quantum/blog/quantum-roadmap-2033 (IBM) or Cirq https://quantumai.google/cirq

- **Real-time Stream Processing Frameworks:** Classical stream processing frameworks like Apache Flink or Apache Spark Streaming can handle real-time data ingestion, pre-processing, and triggering real-time quantum computations when necessary.

//

Design diagrams (See Gemini-1- Raw capture.doc for content details)

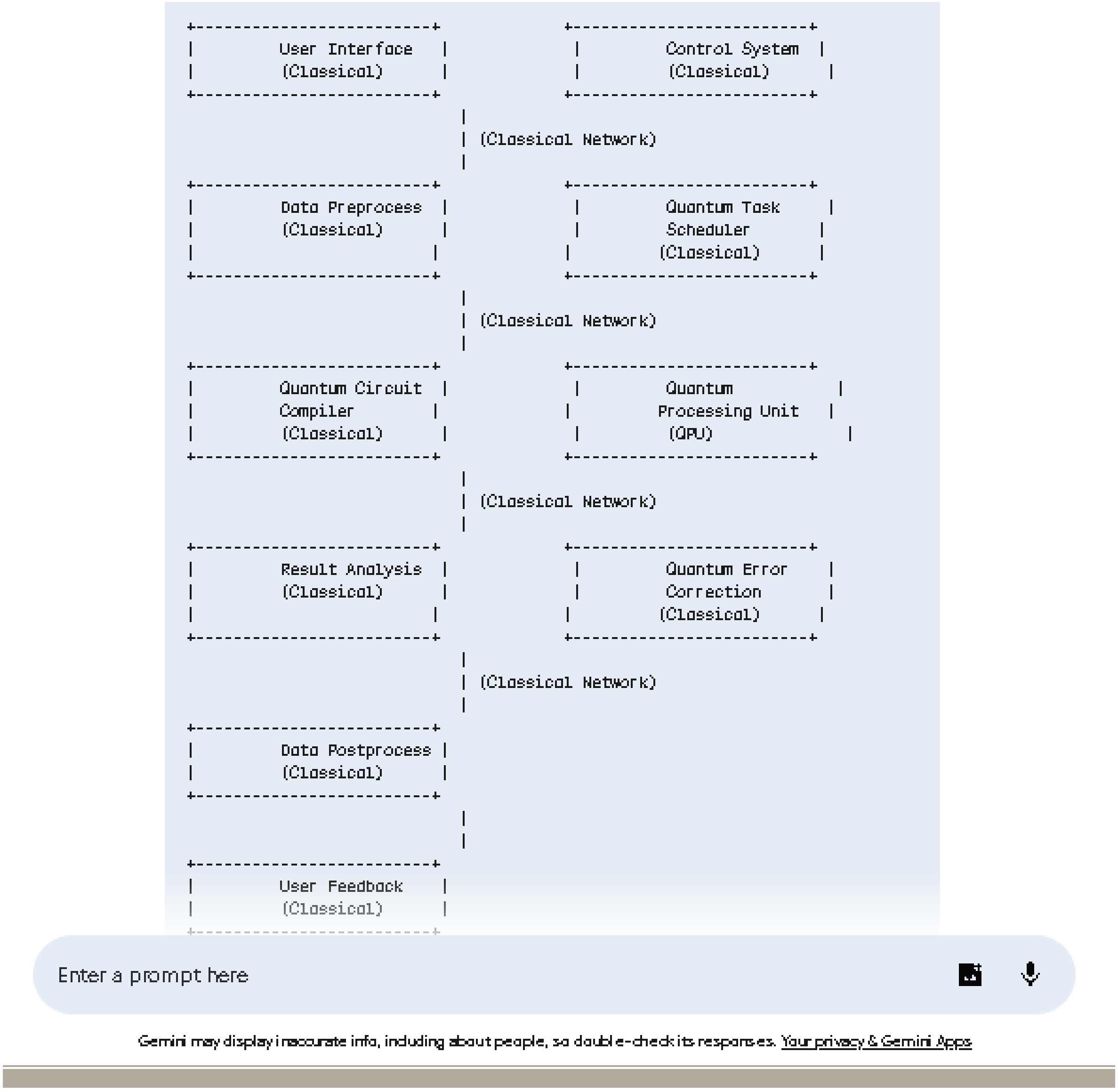

Diagram:

```
+--------------+          +--------------+          +--------------+
| User Interface |          | Job Scheduler |          | System Managemen
+--------------+          +--------------+          +--------------+
        |
        | (Classical Control Unit)
        |
+--------------+          +--------------+          +--------------+
| Quantum      |          | Quantum      |          | Quantum      |
| Circuit     |          | Circuit     |          | Circuit     |
| Compiler    |          | Compiler    |          | Compiler    |
+--------------+          +--------------+          +--------------+
        |
+--------------+
| Classical    |
| Data Pre-    |
| Processing   | (Scalable)
+--------------+
        |
+--------------+
| Quantum      |
| Processing   | (Scalable - Multiple QPUs)
| Unit (QPU)   |
+--------------+
        |
+--------------+
| Classical    |
| Post-        |
| Processing   | (Scalable)
+--------------+
        |
+--------------+
| Communication |
| Network      | (Scalable)
+--------------+
```

Remember: This is a high-level design, and specific implementations will vary


```
+--------------------+          +--------------------+          +--------------------+
|     User Interface  |          |   Classical Pre-   |          |  Quantum Post-     |
|                    | - - - - - -> |   Processing     | - - - - - -> |   Processing     |
+--------------------+          +--------------------+          +--------------------+
                   |
                   | (Real-time & Batch)
                   v
+--------------------+          +--------------------+
|  Classical Control  |          |  Quantum Circuit  |
|    Unit (CCU)      | - - - - - -> |  Compiler (QCC)  | - - - - - -> |  Quantum Processing|
|                    |          |                  |          |     Unit (QPU)     |
+--------------------+          +--------------------+          +--------------------+
                   |
                   | (Optimizes for QPU)
                   v
+--------------------+
|  Quantum Hardware   |
|    Interface (QHI)  |
+--------------------+
                   |
                   | (Controls & Reads QPU)
                   v
+--------------------+
|     Quantum Bits    | (Scalable from 100 to 1 Million)
+--------------------+
```

Diagram:

```
+-------------------+         +------------------+
| On-Premises DC    |         | GCP Public Cloud |
+-------------------+         +------------------+
          |
          | Cloud Interconnect (or VPN)
          |
+-------------------+         +------------------+
| Private Cloud      |         |  Compute Resources|
| Platform (e.g.,   |         |                   |
| VMware)           |         |                   |
+-------------------+         +------------------+
          |
          | Workload Management
          | (e.g., Kubernetes)
          |
+-------------------+         +------------------+
| Your Applications  | <------> | GCP Services     |
+-------------------+         +------------------+
                                 | (For Bursting)
```

How it Works:

- Your primary workload runs on your private cloud infrastructure.
- Workload management software monitors resource utilization within your private

High-Level Diagram:

```
+-----------------+      +-----------------+      +-----------------+
| On-Premises Datacenter |      | Google Cloud Platform |      | Client Applications|
+-----------------+      +-----------------+      +-----------------+
            |
            | Cloud Interconnect (or Cloud VPN)
            |
+-----------------+      +-----------------+
| Virtualization Platform|      | Cloud Monitoring |
+-----------------+      +-----------------+
            |
            | (Management & Orchestration)
            |
+-----------------+      +-----------------+      +-----------------+
| Software-Defined Network |      | Autoscaling Groups |      +--------+
+-----------------+      +-----------------+      | Load Balancer |
            |                                     +--------+
            |
+-----------------+      +-----------------+      +-----------------+
| Storage Solution  |      | VMs (Scalable)     |      | (On-Premises & GCP)|
+-----------------+      +-----------------+      +-----------------+
```

```
+--------------------+         +------------------+
| On-Premises Data    |         | GCP Public Cloud |
| Center             |         +------------------+
+--------------------+
      |
      | (Building)
      v
+--------------------+         +--------------------+
| Network Firewall   |         | Router             |
+--------------------+         +--------------------+
      |
      v
+--------------------+         +--------------------+
| Top of Rack Switch  | (Network Fabric) | Top of Rack Switch  |
+--------------------+         +--------------------+
      |                 |                 |
      v                 v                 v
+---------+---------+ +---------+---------+ +---------+---------+
| Server  | Server  | | Server  | Server  | | Server  | Server  |
+---------+---------+ +---------+---------+ +---------+---------+
      |                 |                 |
      v                 v                 v
+---------+---------+ +---------+---------+ +---------+---------+
| Storage | Storage | | Storage | Storage | | Storage | Storage |
+---------+---------+ +---------+---------+ +---------+---------+
      |                 |
      v                 v
+--------------------+
|  ... More Racks ...|
+--------------------+
      |
      v
+--------------------+         +--------------------+
| Interconnect       | -----> |                  |
+--------------------+         +--------------------+
      | (Private Network)
      v
+--------------------+
| Your Applications  |
+--------------------+
      |
      v
          +---------+---------+
          | Bursting |         |
          | Needs    |         |
          +---------+---------+
                |
                v
          +-----------------+
          | GCP Scalable     |
          | Resources (VMs,  |
          | Containers)      |
          +-----------------+
```

Private Cloud Rack (Without QPU):

```
+--------------------+
| Top of Rack Switch | (TOR)
+--------------------+
|                    |
| Network Fabric     | (10GbE or higher)
|                    |
+--------------------+
|                    |
| Interconnect       | (Dedicated or Cloud VPN)
|                    |
+--------------------+
|                    |
| Server (x86_64)    | (Multiple with CPUs, Memory, Storage)
|                    |
+--------------------+
|                    |
| Server (x86_64)    | (Multiple with CPUs, Memory, Storage)
|                    |
+--------------------+
|                    |
| ... (More Servers)  |
|                    |
+--------------------+
|                    |
| Storage Array       | (SAN or NAS)
|                    |
+--------------------+
```

Rack-Based Private Cloud with Bursting to GCP Public Cloud

```
+--------------------+         +------------------+
| On-Premises DC     |         | GCP Public Cloud |
+--------------------+         +------------------+
     |
     | (Private Network)
     v
+---------+---------+      +---------+---------+
| Top of  | Rack     |      | Top of  | Rack     |
| Rack Sw | Switch  | -----> | Rack Sw | Switch  |
+---------+---------+      +---------+---------+
     |
     v (Network Fabric)
+---------+---------+      +---------+---------+
| Server  | Server  |      | Server  | Server  |
| (CPU,   | (CPU,   | -----> | (CPU,   | (CPU,   |
| Memory) | Memory) |      | Memory) | Memory) |
+---------+---------+      +---------+---------+
     |
     v (Storage)
+---------+---------+      +---------+---------+
| Storage | Storage | -----> | Storage | Storage |
+---------+---------+      +---------+---------+
     |
     v (Interconnect)
+---------+---------+      +---------+---------+
|         |         | -----> |         |         |
|  Private |  Cloud  |      |   Public|   Cloud  |
|  Network|  Platform|      |  Network|  Platform|
+---------+---------+      +---------+---------+
     |
     v (Applications)
+---------+---------+
| Your     |         |
| Apps     |         |
+---------+---------+
     |
     v (Bursting)
+---------+---------+
|         |         |
|  Scaling |  Needs  |
|         |         |
+---------+---------+
     |
     v (GCP Resources)
+---------+---------+
| Scalable| VMs/     |
| VMs/     | Containers|
| Containers|         |
+---------+---------+
```

Disclaimer: This content has been sourced from multiple resources like our internal staff, Google Gemini, and other cloud providers, reference links etc. We have no intention for any copyright infringements. This is used for knowledge sharing and awareness of our work in this area of QAI and the related cloud design