Large Language Model with Retrieval Augmented Generation

Project Objective and Limitations

i Project Overview

The advent of modern automobile manufacturing has led to increased technical complexity, often resulting in mechanics opting to replace parts rather than diagnose and fix issues. This approach, while convenient for contemporary vehicles, poses a significant challenge for classic cars built 30 to 40 years ago, where replacement parts are scarce or non-existent.

To address this problem, this project aims to leverage Generative AI to create a "virtual mechanic." By utilizing a corpus gathered from a classic car forum, this tool will be capable of understanding unstructured questions and providing relevant answers. This solution aims to assist classic car enthusiasts and mechanics by offering expert guidance, thereby preserving the heritage and functionality of vintage automobiles.

ii Objectives

The primary objective of this project is the development of a Natural Language Processing (NLP) model as part of a portfolio of AI projects that can be showcased to potential employers. This will include an outline of the necessary workflow with a comparison and selection of architectures, libraries, and methods.

iii Use Case

With this code, a user will be able to ask questions in plain, unstructured English and receive answers that are a result of previous similar questions from the forum used to create the corpus. Users will see these answers in plain English. As a programmer, I will have control over the extent to which the answers are sourced from the supplemental corpus versus the pre-trained model. If successful, I will explore publishing the application for others to use.

iv Limitations and Challenges

To address budget constraints, a combination of open source and free resources will be used. Python will be the primary programming language. Google Colab will be used for the notebook with compute resources limited to CPUs.

1. Architectures and Frameworks

This document provides an overview of various architectures, models, and tools used in natural language processing tasks. Understanding the strengths and weaknesses of different approaches is crucial for designing effective NLP systems tailored to my specific use case and requirements.

1.1 Machine Learning NLP Architectures

1.1.1 Traditional Models

Solution:	Bag-of-Words (BoW)
Description:	Represents text data as a collection of unique words and their frequencies.
Example:	Term Frequency-Inverse Document Frequency (TF-IDF)
Pros:	Simple and efficient representation.
	Works well for tasks like sentiment analysis and document classification.
Cons:	Ignores word order and context.
	Doesn't capture semantic meanings well.

1.1.2 Statistical NLP Models

Solution:	Hidden Markov Models (HMM)
Description:	Sequential text models based on hidden state transitions
Example:	hmmlearn (Python library for HMMs)
Pros:	Captures sequential dependencies effectively.
	Suitable for tasks like part-of-speech tagging and named entity recognition.
Cons:	Requires labeled sequential data for training.
	May struggle with capturing complex semantic relationships.

Solution:	Conditional Random Fields (CRF)
Description:	Sequence labeling models
Example:	sklearn-crfsuite (Python library for CRFs based on scikit-learn)
Pros:	Effective for sequential labeling tasks.
	Incorporates feature dependencies between adjacent labels.
Cons:	Requires labeled sequential data for training.
	Less effective for capturing long-range dependencies

Solution:	Support Vector Machines (SVM)
Description:	A supervised learning model used for classification and regression analysis.
Example:	scikit-learn (Python library for machine learning)
Pros:	Effective in high-dimensional spaces.
	Versatile with different kernel functions for flexibility in decision boundaries.
Cons:	Memory-intensive for large datasets.
	May require careful selection of kernel functions and tuning parameters.

1.1.3 Deep Learning Models

Solution:	Word Embeddings
Description:	Represent words as dense vectors in a continuous vector space.
Examples:	Word2Vec, GloVe
Pros:	Captures semantic meanings and relationships between words.
	Provides dense vector representations suitable for downstream tasks.
Cons:	Requires large amounts of data for training.
	Struggles with out-of-vocabulary words.

Solution:	Sequence Models
Description:	Models that capture the sequential nature of text data.
Examples:	Hidden Markov Models (HMM), Conditional Random Fields (CRF)
Pros:	Captures sequential dependencies in data.
	Suitable for tasks like named entity recognition and part-of-speech tagging.
Cons:	Requires labeled sequential data for training.
	Can be computationally intensive.

Solution:	Recurrent Neural Networks (RNN)
Description:	Neural networks that process sequences by iterating through elements.
Examples:	Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)
Pros:	Effective for capturing sequential dependencies in data.
	Suitable for tasks like language modeling and machine translation.
Cons:	Vulnerable to vanishing and exploding gradient problems
	Computationally expensive to train.

1.1.4 Transformers

Solution:	Transformer Models
Description:	Neural network architecture based entirely on self-attention mechanisms.
Examples:	BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), T5 (Text-To-Text Transfer Transformer)
Pros:	Captures long-range dependencies effectively.
	Parallelizable training process.
Cons:	Requires large amounts of computational resources.
	Limited interpretability compared to traditional models.

Solution:	Pre-trained Models
Description:	Models pre-trained on large corpora and fine-tuned for specific tasks.
Examples:	BERT, GPT, T5
Pros:	Leverage large amounts of unlabeled data for pre-training.
	Achieve state-of-the-art performance on various NLP tasks.
Cons:	Resource-intensive pre-training process.
	May require substantial computational resources for fine-tuning.

1.1.5 Additional Models and Techniques

Solution:	Retriever-Generator Models
Description:	Models combine retrieval and generation components for text generation tasks.
Examples:	RAG
Pros:	Incorporates both structured and unstructured information for generation.
	Produces more diverse and contextually relevant responses.
Cons:	Requires efficient retrieval mechanisms.
	Increased complexity in model architecture.

Solution:	Knowledge-Enhanced Retrieval-Augmented Generation (KERAG)
Description:	A variant of RAG that incorporates knowledge graphs.
Examples:	Graph-BERT
Pros:	Integrates structured knowledge for improved understanding and generation.
	Enables more coherent and contextually relevant responses.
Cons:	Requires high-quality and curated knowledge graphs.
	Increased computational complexity compared to standard RAG.

Solution:	Elastic Search
Description:	Distributed search and analytics engine for indexing and searching big data.
Examples:	Elasticsearch, Apache Solr
Pros:	Scalable and distributed architecture.
	Supports full-text search and complex query structures.
Cons:	
	Requires infrastructure for deployment and maintenance.
	Indexing and search performance may degrade with large datasets.

Architecture Options Score Card

Model/Architecture	Key Strength	CPU Compatibility	Ease of Use	Performance & Accuracy	Scalability	Integration	Total
RAG	Context Understanding	1	1	2	2	2	8
BoW	Simplicity	2	2	0	2	2	8
Pre-trained Models	Accuracy	1	1	1	2	2	7
Word Embeddings	Semantic Understanding	1	1	2	1	2	7
Elastic Search	Scalability	2	1	1	2	1	7
Transformer Models	State-of-the-Art	0	0	2	1	2	5
KERAG	Knowledge Integration	0	0	2	1	2	5
HMM	Sequence Modeling	1	1	1	1	1	5
CRF)	Sequential Labeling	1	1	1	1	1	5
SVM	Versatile	1	1	1	1	1	5
Sequence Models	Order Preservation	1	1	1	1	1	5
RNN	Sequential Dependencies	0	1	2	0	1	4

0: Does not meet 1: Partially meets 2: Fully meets

Conclusion

The RAG model is most appropriate for this effort given its heavy use of domain specific information (that may be missing from a stand-alone pre-trained model). Its contextual accuracy has a higher weighted value for this use case than Ease of Use. Having said that, it is a computationally heavy architecture, and it is unclear if free cloud CPU resources will be sufficient.

Example of RAG Model Implementation

1. **Query**: "What are the benefits of using a RAG model?"
2. **Retriever**:
   - Searches a corpus for relevant documents or passages related to "benefits of using a RAG model".
   - Retrieves top-k documents or passages that discuss the advantages of RAG models.
3. **Generator**:
   - Takes the retrieved documents and generates a response: "A RAG model combines the strengths of information retrieval and generative modeling. It retrieves relevant documents to provide context and generates accurate and contextually appropriate responses. This makes it highly effective for tasks requiring detailed and specific information."

For this project, the Retriever will be the corpus scraped from the online forum processed with Word Embeddings, and the Generator will be from a pretrained model.

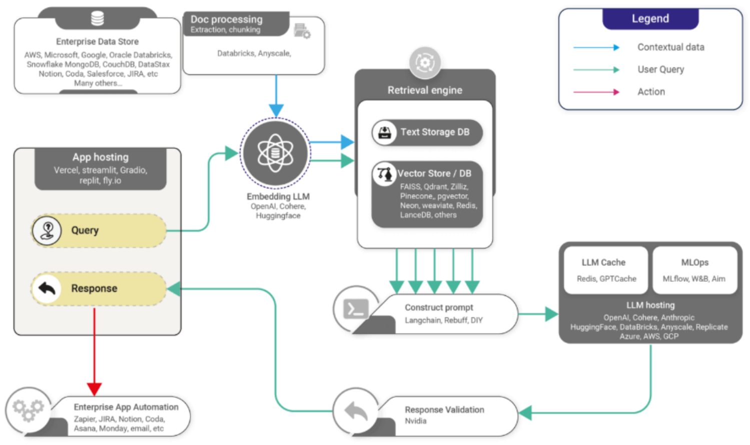
Enterprise Data Store
AWS, Microsoft, Google, Oracle Databricks, Snowflake MongoDB, CouchDB, DataStax Notion, Coda, Salesforce, JIRA, etc Many others...
Doc processing
Extraction, chunking
Databricks, Anyscale,
Legend
Contextual data
User Query
Action
Retrieval engine
Text Storage DB
Vector Store / DB
FAISS, Qdrant, Zilliz, Pinecone,, pgvector, Neon, weaviate, Redis, LanceDB, others
App hosting
Vercel, streamlit, Gradio, replit, fly.io
Embedding LLM
OpenAI, Cohere, Huggingface
Query
Response
LLM Cache
Redis, GPTCache
MLOps
MLflow, W&B, Aim
LLM hosting
OpenAI, Cohere, Anthropic HuggingFace, DataBricks, Anyscale, Replicate Azure, AWS, GCP
Construct prompt
Langchain, Rebuff, DIY
Enterprise App Automation
Zapier, JIRA, Notion, Coda, Asana, Monday, email, etc
Response Validation
Nvidia

Figure 1RAG Diagram from Bijit Ghosh @ Medium.com

1.2 Generator options

Vendor:	OpenAI
Package:	GPT (GPT-2)
Description:	Generative Pre-trained Transformer for generating text.
Pros:	Highly capable of generating coherent and contextually relevant text.
	Free to access and use.
Cons:	Requires significant computational resources for fine-tuning.
	GPT-2 is less powerful than newer models.

Vendor:	Anthropic
Package:	Claude (Claude 3)
Description:	AI assistant designed for safety and ethical considerations.
Pros:	Enhanced safety features and focus on ethical AI use.
	Designed for robust handling of varying text lengths.
Cons:	Primarily available for research access, which may limit commercial use.
	Conditional access, potentially limiting deployment flexibility.

Vendor:	Meta
Package:	DistilBART (sshleifer/distilbart-cnn-12-6)
Description:	A distilled version of BART optimized for efficiency.
Pros:	Optimized for CPU usage, making it efficient for resource usage.
	Open-source and free, allowing for flexible use and customization.
Cons:	Less powerful than the full BART model due to distillation.
	May require additional integration efforts compared to more commercial models.

Vendor:	Google
Package:	T5 (t5-small)
Description:	Text-to-Text Transfer Transformer for various NLP tasks.
Pros:	Highly flexible and powerful for a wide range of text-to-text tasks.
	Open-source and free to use and fine-tune.
Cons:	May require additional preprocessing steps for certain tasks.
	The small version is less powerful compared to larger T5 models.

Vendor:	Amazon
Package:	AWS Comprehend
Description:	Managed NLP service for text analysis and insights.
Pros:	Fully managed service, reducing the burden of infrastructure management.
	Tight integration with AWS ecosystem, offering scalability and ease of use.
Cons:	API-dependent, limiting control over the underlying models.
	Cloud performance may not be as optimized as specific CPU performance.

Generator Option Score Card

Vendor	Package	Key Strength	CPU Compatibility	Ease of Use	Performance & Accuracy	Integration & Flexibility	Scalability	Total
Meta	DistilBART	CPU Efficiency	2	2	1	2	2	9
Google	t5-small	Flexibility	2	1	2	2	1	8
Amazon	Comprehend	Managed Service	1	2	1	2	1	7
OpenAI	GPT-2	Coherent Text Generation	0	2	2	2	0	6
Anthropic	Claude 3	Ethical AI	1	1	2	1	1	6

0: Does not meet 1: Partially meets 2: Fully meets

1.3 Frameworks and Tools

Vendor:	Google
Package:	TensorFlow
Description:	Open-source ML framework for building and deploying models.
Pros:	Comprehensive ecosystem with deep learning support.
	Scalable on both CPUs and GPUs.
Cons:	Steeper learning curve than some other frameworks.
	Limited support for dynamic computation graphs.

Vendor:	Meta
Package:	PyTorch
Description:	Open-source deep learning framework by Meta AI Research.
Pros:	Pythonic and intuitive interface for model development.
	Dynamic computation graph for easier debugging and experimentation.
Cons:	Less optimized for production deployment than TensorFlow.
	Limited built-in support for distributed training.

Vendor:	Python Software Foundation
Package:	scikit-learn
Description:	Simple, efficient ML library built on NumPy, SciPy, and matplotlib.
Pros:	Easy-to-use API for common machine learning tasks.
	Comprehensive documentation and community support.
Cons:	Limited support for deep learning models and architectures.
	Less flexibility compared to deep learning frameworks.

Framework Score Card

Vendor	Package	Key Strength	CPU Compatibility	Ease of Use	Performance & Accuracy	Integration & Flexibility	Scalability	Total
Meta	PyTorch	Dynamic Computation Graph	2	2	2	2	1	9
Google	TensorFlow	Comprehensive Ecosystem	2	1	2	2	2	9
Python Software Foundation	scikit-learn	Simplicity	2	2	1	1	1	7

0: Does not meet 1: Partially meets 2: Fully meets

1.4 Embedding

Solution:	Universal Sentence Encoder
Provider:	Google
Libraries:	TensorFlow Hub
Pros:	Captures sentence-level embeddings, enhancing text understanding.
	Efficient and easy to integrate with TensorFlow models.
Cons:	May not capture fine-grained word-level nuances.
	Performance can vary depending on the complexity of the sentences.

Solution:	FastText
Provider:	Meta
Libraries:	Gensim, TensorFlow, PyTorch
Pros:	Handles out-of-vocabulary words as bags of character n-grams.
	Captures subword information, enhancing the representation of rare words.
Cons:	Increases computational complexity due to subword representations.
	Larger model size compared to Word2Vec and GloVe.

Solution:	Amazon SageMaker Embeddings
Provider:	AWS
Libraries:	Amazon SageMaker
Pros:	Provides pre-built models for embeddings, simplifying deployment.
	Integrates seamlessly with other AWS services for scalability.
Cons:	Requires familiarity with the AWS ecosystem.
	Costs can increase with extensive usage.

Solution:	GPT-3 Embeddings
Provider:	OpenAI
Libraries:	OpenAI API
Pros:	Generates high-quality, contextually relevant text embeddings.
	Handles long-range dependencies and contextual information.
Cons:	Requires significant computational resources.
	Access may require API usage and associated costs.

Solution:	Claude Embeddings
Provider:	Anthropic
Libraries:	Anthropic API
Pros:	Offers state-of-the-art embeddings with a focus on safety and ethics.
	Handles context and nuances effectively for complex tasks.
Cons:	Primarily available for research access, limiting commercial use.
	Access may require API usage and associated costs.

1.5 Tokenization

Tokenization is a crucial preprocessing step in NLP, segmenting text into manageable units for further analysis or model training. The choice of tokenization strategy affects both the complexity of the model and its ability to understand the text.

Solution:	Word-level Tokenization
Libraries:	NLTK, spaCy, TensorFlow/Keras Tokenizers, BPE, Hugging Face Tokenizers
Pros:	Preserves word integrity and meaning, crucial for comprehension tasks.
	Subword tokenization methods like BPE can efficiently handle unknown words.
Cons:	Can result in a large vocabulary, increasing memory and processing needs.
	May overlook nuances in character-level variations.

Solution:	Character-level Tokenization
Libraries:	Supported by deep learning frameworks like TensorFlow and Keras
Pros:	Captures morphological nuances at the character level, aiding rich languages.
	Simplifies vocabulary to unique characters, reducing model complexity.
Cons:	Leads to longer input sequences, increasing computational costs.
	Loses direct access to semantic information in words or phrases.

Solution:	Subword Tokenization
Libraries:	A blend of word-level and character-level tokenization methods
Pros:	Balances vocabulary size and semantic information preservation.
	Handles rare or unknown words by breaking them into recognizable subwords.
Cons:	Requires preprocessing to establish a subword vocabulary, adding complexity.
	Generated subwords may lack standalone meaning, complicating interpretation.

Solution:	Model-Specific Tokenization
Libraries:	Hugging Face's transformers library provides access to pre-built tokenizers
Pros:	Ensures tokenization consistency with the model's original training data.
	Reduces the need for extra preprocessing steps and custom tokenization.
Cons:	Limited flexibility to change tokenization beyond the model's method.
	May not be efficient for tasks outside the model's specific design.

Tokenization and embedding must be considered together because tokenization directly impacts the quality of embeddings. The choice of tokenization method determines how text is segmented, which in turn affects how embeddings capture context and meaning. Inconsistent tokenization can lead to poor embeddings and reduced model performance. Properly aligned tokenization and embedding processes ensure that the text's structure and semantics are preserved, enhancing overall model effectiveness.

Tokenization and Embedding Score Card

Vendor	Embedder	Compatible Tokenizer	CPU Compatibility	Ease of Use	Performance & Accuracy	Integration & Flexibility	Scalability	Total
Google	Universal Sentence Encoder	TensorFlow Text	2	2	2	2	2	10
Meta	FastText	Gensim Tokenizer or NLTK	2	2	2	2	1	9
OpenAI	GPT-2 Embeddings	OpenAI's GPT-2 Tokenizer	2	2	2	2	1	9
Amazon	Amazon SageMaker Embeddings	Amazon SageMaker’s preprocessing tools	1	2	2	2	1	8
Anthropic	Claude Embeddings	Anthropic API’s built-in tokenization	1	2	2	2	0	7

0: Does not meet 1: Partially meets 2: Fully meets

1.6 Vendor Leader Board

Vendor	CPU Compatibility	Ease of Use	Performance & Accuracy	Integration & Flexibility	Scalability	Total
Meta	6	6	4	6	5	27
Google	6	4	6	6	4	26
Amazon	3	6	4	6	3	22
OpenAI	2	6	6	6	1	21
Anthropic	3	4	6	4	2	19

Conclusion

The results are interesting and paint a clearer picture of how the strengths of each option compare. While close, Google seems to have an edge on Performance and Accuracy but suffers a bit on Ease of Use and Scalability. Having said that, the use of Google would support my efforts to gain Google Could Certification—a highly desirable skill in the job market. With that in mind, I'll be moving forward with a Google dominant stack.

1.7 Develop Corpus

Data Ethics
The data collected here is a collection of posts from widely available public forum. However, should this project move into public distribution, additional step will be necessary to endure PII is obfuscated or removed. In addition, this document shall serve as full disclosure of the projects goals and data gathering process.

Data Collection
The project leverages user-generated content from a domain-specific online forum as the training corpus. This data is largely unstructured, with minimal metadata available. The following tools were considered to gather the source text for the corpus:

Web Scraping

Tools:	Beautiful Soup, online SaaS products
Pros:	Direct Access to Targeted Data: Enables precise extraction of user-generated content from specific sections or threads within the forum.
	Efficiency in Data Collection: Automated scripts can gather large volumes of data in a short amount of time, making it suitable for assembling significant datasets for NLP.
Cons:	Potential for Incomplete Data: May miss embedded content or dynamically loaded data, depending on the website's structure.
	Ethical and Legal Considerations: Scraping data from forums may raise concerns about user privacy and must adhere to the terms of service of the website.
	Very Platform Dependent: Forum specific solutions result in forum specific data schemas that must be reverse engineered for successful text extraction.

Forum-specific APIs

Tools:	Python (\`requests\` library for API calls and \`json\` library for handling responses)
Pros:	Structured and Reliable Data Retrieval: APIs provide structured data, making it easier to process and integrate into your project.
	Efficient and Direct Access: Directly accessing the forum's data through its API is efficient, bypassing the need for HTML parsing.
	Compliance and Ethical Data Use: Utilizing APIs respects the forum's data policies and ensures access is in line with user agreements.
Cons:	Rate Limiting: APIs often have limitations on the number of requests that can be made in a certain timeframe, which could slow down data collection.
	API Changes: Dependence on the forum's API structure means that changes or deprecation could disrupt your data collection pipeline.
	Access Restrictions: Some data or functionalities might be restricted or require authentication, posing additional challenges for comprehensive data collection.

2. Preprocessing Text
3. Clustering and Summarization
4. Format Text for Training
5. Indexing
6. Query Processing and Search
7. Retrieve and Rank
8. Answer Generation
9. Evaluation and Tuning
10. Deployment

11. Appendix

Reflection and Learning

Challenges Faced

Lessons Learned

Future Work and Improvements

Potential Enhancements

Areas for Further Research

Workflow for NLP RAG Model

1 Architectures and Models
2 Corpus Development
3 Clean and Preprocess Text
4 Clustering and Summarization
5 Format Text for Training
6 Embedding and Indexing
7 Query Processing and Search
8 Retrieve and Rank
9 Answer Generation
10 Evaluation and Tuning
11 Deploying the Demo

Complete Stack Options

Google

Architecture	TensorFlow
Library	TensorFlow
Framework	TensorFlow Extended (TFX)
Tokenizer	SentencePiece
Embedding	Universal Sentence Encoder
Summarization	T5
Storage	Google Cloud

AWS

Architecture	SageMaker, Bedrock
Library	Comprehend
Framework	SageMaker
Tokenizer	Comprehend
Embedding	SageMaker, Bedrock
Summarization	SageMaker, Bedrock
Storage	Amazon S3

Meta

Architecture	PyTorch
Library	PyTorch
Framework	PyTorch
Tokenizer	Bart
Embedding	Word2Vec
Summarization	DistilBART
Storage	Proprietary

OpenAI

Architecture	GPT
Library	OpenAI API
Framework	OpenAI Codex
Tokenizer	GPT-3 Tokenizer
Embedding	GPT-3 Embeddings
Summarization	GPT-3
Storage	Proprietary

Anthropic

Architecture	Claude
Library	Anthropic API
Framework	Custom framework
Tokenizer	Claude Tokenizer
Embedding	Claude Embeddings
Summarization	Claude 3
Storage	Proprietary

Score Card Template

Provider	Specific Option	Strength	CPU Performance	Free	Ease of Integration	API Independence	Use Case appropriate	Total Score
Meta								
Google								
OpenAI								
Anthropic								
Amazon								

Complexity and Ease of Use

1.8 Integrated Combinations

Choosing a vertically integrated library and framework combination ensures seamless integration and optimized performance, simplifying development and maintenance. This approach also provides consistent APIs and comprehensive ecosystem support, enhancing productivity and efficiency.

Stack:	PyTorch and Hugging Face Transformers
Pros:	PyTorch offers dynamic computation graphs for RAG model development.
	Hugging Face's library provides access to pre-trained models and tokenizers.
Cons:	Combination might require a steep learning curve for new users.
Stack:	TensorFlow and T5
Pros:	TensorFlow offers robust tools for model development and deployment.
	T5 is versatile for text-to-text tasks, adaptable for RAG purposes.
Cons:	TensorFlow's static graphs can be less intuitive than PyTorch's dynamic ones.
	T5's format might require additional preprocessing steps.
Stack:	scikit-learn and NLTK
Pros:	NLTK provides tools for text preprocessing and feature extraction.
	scikit-learn offers a suite of traditional machine learning algorithms.
Cons:	NLTK and scikit-learn may not scale as well as deep learning frameworks.
	This combination is less suited for deep learning tasks.

1.9 Data Storage and Database

Efficient data storage and management are pivotal for the project, focusing on accommodating extensive unstructured data from various sources. The project explores two main classes of storage solutions: Cloud Storage and Local Storage, each offering unique benefits and challenges.

Solution:	Cloud Storage
Libraries:	Snowflake, AWS, Google Cloud
Pros:	Scalability: Scales to meet growing data demands without physical management.
	Accessibility: Provides global access, facilitating collaboration and remote work.
Cons:	
Cost:	Costs can increase significantly with data volume and throughput.
	Internet Dependence: Requires consistent internet access, which can be limiting.

Solution:	Local Storage
Libraries:	MySQL, MongoDB
Pros:	
Control:	Complete control over the data storage environment and configurations.
Cost:	No ongoing costs related to storage size or access, aside from initial setup.
Cons:	Scalability: Physical limits to scalability; expanding requires additional hardware.
	Maintenance: Requires resources for maintenance, backups, and security.