QINFINITE

Qifinite AI Engineering

Scalable, Secure and Unified AIOps, LLMOps & MLOps

Qinfinite's Integrated AI Engineering Framework: Enabling Scalable and Responsible AIOps, LLMOps, and MLOps

Qinfinite delivers a unified AI engineering framework that integrates AIOps, LLMOps, and MLOps to streamline the development, deployment, and governance of AI solutions. With capabilities like lifecycle management, prompt and feature optimization, RAG support, and responsible AI checks, it enables scalable, secure, and continuously improving AI operations in enterprise environments.

LLMOps for Managing Language Model Agents

It defines and elaborate on the key features and components required in a robust LLMOps (Large Language Model Operations) framework for managing, monitoring, and maintaining LLM-based agents across their lifecycle—from creation to production and beyond.

1. **Agent Lifecycle Management**

a. **Creation**
   - Instantiate agent instances with custom configurations
   - Specify role, persona, and objective for each agent
   - Integrate context-specific data sources (documents, APIs, databases)

b. **Versioning**
   - Maintain multiple versions of agent configurations, prompts, and models
   - Track changes over time using semantic or semantic diffing tools

c. **Deployment**
   - Deploy agents to different environments (dev, staging, production)

d. **Retirement**
   - Schedule or archive deprecated agents
   - Ensure data and logs are retained for audits

2. **Prompt Management**

a. **Prompt Repository**
   - Centralized storage of prompts with tagging and versioning

b. **Prompt Testing & Optimization**
   - Simulate prompt executions across different models and provide the best LLM model options.
   - Optimize prompt performance using prompt engineering techniques

- AI Model suggestions of prompts. Using AI model to design effective prompts suggestions.

3. Model Management

a. LLM & Multi-Model Support

- Use multiple LLMs (e.g., GPT-4, Claude, Mistral, LLaMA)
- Assign specific models to different instances

b. Vector Store Configuration

- Ability to select from multiple vector database backends such as:
  - **FAISS**(in-memory, fast for experimentation)
  - **Pinecone**(managed, scalable)
  - **Weaviate**(semantic + hybrid search, schema-aware)
  - **Qdrant**(production-ready with REST + gPRC)
  - **Milvus** (high-throughput large-scale vector store)
  - **Chroma** (lightweight, open-source)
  - **PgVector**

c. LLMOps Support for RAG Strategies

LLMOps should include configurable, testable, and version able chunking strategies to optimize Retrieval-Augmented Generation (RAG) workflows

Key Features

Feature	**Description**
Chunk Size / Overlap Parameters	Manage via central configuration or UI per agent
Chunking Algorithm	Token-based, sentence-based, or semantic splitting
Dynamic Chunking	Choose strategy dynamically based on document type or content
Versioning of Chunk Strategies	Store and track different strategies with unique hashes or timestamps
Auto-Tuning Chunk Parameters	Automatically adjust chunk size and overlap using retrieval and performance metrics

Example Use Case

LLMOps dashboard shows chunking configuration for **Agent X**:

- **Chunk Size:** 512 tokens
- **Overlap:** 100 tokens
- **Method:** Sentence-based splitting (sentence_split)

This chunking config helps Agent X retrieve more coherent and complete responses by preserving context across chunk boundaries.

d. Retrieval Configuration & Evaluation

- Define temperature, top-k, filters, reranking, hybrid search, etc.
- Track retrieval effectiveness using metrics:
  - ➢ Top-k hit rate
  - ➢ Retrieval recall
  - ➢ Relevance scores
  - ➢ Average chunk usefulness per query

4. Validation & Qulaity Assurance

a. Benchmarking

- Evaluate performance on standard benchmarks (BLEU, ROUGE, SQuAD, etc.)

b. Responsible AI Checks

Key Responsible AI Dimension:

Check	**Description**
Fairness	Ensures that outputs do not disproportionately **favour** or **disfavour** any demographic or social group. Involves detecting discrimination or stereotyping.
Bias Detection	Identifies and mitigates model biases by evaluating responses using known bias benchmarks (e.g., gender, racial, political bias). Supports A/B testing.
Explainability	**Prediction Token Probabilities** and determining the LLM **confidence** on the outcome and confidence of **next work prediction**.
Content Safety & GDPR Violation Detection	Detects and flags harmful, toxic, violent**, PII level information or c**ontent.

c. Hallucination Rate Detection

Hallucination in LLMs refers to scenarios where the model generates information that is factually incorrect, unverifiable, or fabricated. This is especially critical in Retrieval-Augmented Generation (RAG) workflows, where the model is expected to generate answers grounded strictly in retrieved source documents.

LLMOps frameworks should include tools and mechanisms to detect, monitor, and reduce hallucination rates across agent instances to ensure reliability and trustworthiness in production systems.

Technique	Description
Groundedness Check	Compares the generated answer with retrieved chunks. If content is not found in sources, it's flagged as hallucinated.
Fact-Checking API Integration	Uses third-party tools or fact-checking APIs (e.g., , custom knowledge bases, Domain Data Document) to validate output facts.
Human-in-the-loop Review	Flags low-confidence responses for manual review to establish or calibrate hallucination detection accuracy.
Prompted Self-Verification	Adds a second model pass with a prompt like "Is this answer factual and supported by the provided documents?"
Judge Model	A secondary, larger model may be prompted occasionally, or while benchmarking, and the difference in answers will be calculated.

d. LLMOps Metrics & Dashboard Support

Metric	Purpose
Hallucination Rate (%)	Measures the percentage of model responses not grounded in the knowledge source.
Grounding Score	A numeric score reflecting how well a response is supported by retrieved evidence.
Confidence Score vs Accuracy	Detects mismatch between the model's confidence in an answer and its actual factual correctness.
Document Match Ratio	Compares the token-level or semantic alignment between the output and the retrieved content.

e. A/B Testing & Canary Deployments

- Compare agent versions in real-time to measure performance differences across various quality assurance factors, responsible AI checks and benchmarks mentioned above.

5. Continuous Learning with Human Feedback

The objective is to use structured human feedback to improve model predictions over time via fine-tuning or adapter-based updates for various tasks:

- Role prediction
- Intent detection
- Ticket prioritization and routing

The feedback captured by the SME will be stored in Feedback Logger and are used as input for model learning.

Data	Explanation
Prompt	Original ticket text or query
Task Type	Task category such as role_classification, intent_detection
True Label	Label provided by a human reviewer or SME
Original Prediction	Model's original prediction before correction
Reviewer ID	Identifier for the human who provided the feedback (for traceability)
Source	Indicates how the feedback was collected: manual_feedback, automated_flag, etc.

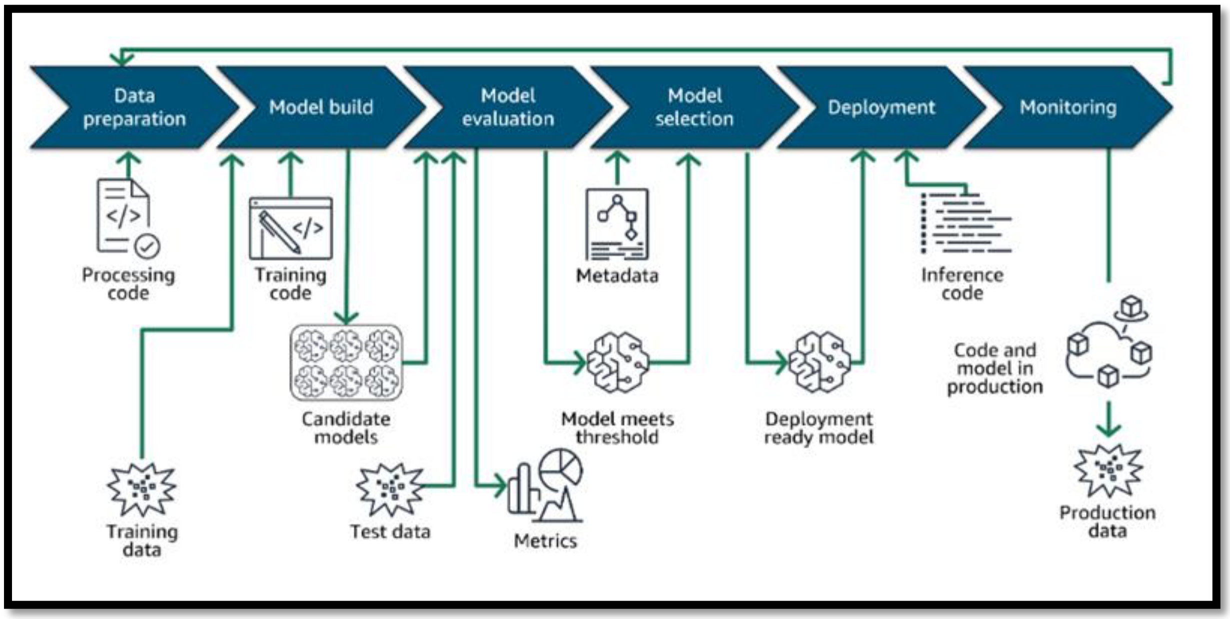
6. **Data Drift Detection**

Data drift occurs when the nature of user input changes over time, leading to potential degradation in model performance. Drift can be semantic (topic or meaning-based), distributional (frequency shift), or functional (output no longer meets user intent).

Method	Explanation
Query Volume & Type Monitoring	Monitor trends in incoming user prompts to detect changes in domain, topic, or query intent.
Embedding-Based Semantic Drift	Convert historical vs. current queries into embeddings (e.g., using Sentence Transformers) and compute divergence.
Statistical Feature Drift	Use statistical techniques (e.g., JS Divergence, KL Divergence) to track changes in prompt or response features.
Topic Modelling & Clustering	Use clustering algorithms (e.g., DBSCAN) on recent queries to identify new or emerging topic clusters.

Example: A customer support agent trained on SAP order management starts receiving a surge in "returns policy for international orders" — previously unseen. Drift is flagged for investigation.

7. LLM Fine Tuning and Auto-LLM Selection



The flowchart outlines a **machine learning lifecycle**, which applies to **LLM (Large Language Model) fine-tuning** and deployment pipelines.

a. Data Preparation

Goal: Gather and pre-process data tailored for fine-tuning the LLM

- **Training Data**:
  - Collect domain-specific, high-quality text data (e.g., customer chats, legal contracts, product descriptions).
  - Labelling or formatting may be required (eg. role/intent annotations).
- **Processing Code**:
  - Pre-processing involves:
    - Cleaning (removing HTML, special characters)
    - Tokenization
    - Formatting (prompt-response pairs, instruction-following templates)
  - May include splitting into training/validation/test datasets.

Example: Formatting a customer support chat as "User: My order is delayed. \nAgent:" for instruction tuning.

b. Model Build

Goal: Train/fine-tune the LLM with the prepared dataset.

- **Training Code**:
  - Choose a base model (e.g., gpt4o, LLaMA, Mistral, etc.)

    - Use libraries like LoRA (parameter-efficient fine-tuning) or unsloth
    - Specify hyperparameters (batch size, learning rate, epochs, max sequence length)

- **Candidate Models**:
    - Multiple versions can be trained with different configurations or subsets of data
    - Outputs are typically saved checkpoints (e.g., model_epoch_2.safetensors)

c. Model Evaluation

Goal: Assess model performance quantitatively and qualitatively.

- **Test Data**:
    - Held-out dataset not seen during training.
    - Should reflect real-world input types for the LLM use case.
- **Metrics**:
    - Common LLM evaluation metrics:
        - **Perplexity**: Language fluency
        - **BLEU/ROUGE**: For summarization or translation
        - **Accuracy/F1**: For classification tasks
        - **Exact Match (EM)**: For question-answering
    - Human feedback or RLHF may also be used.

Example: Evaluate a fine-tuned FAQ model on EM and F1 against benchmark questions.

d. Model Selection

Goal: Choose the best model for deployment based on performance.

- **Model Meets Threshold**:
    - Define acceptance criteria (e.g., ≥ 85% BLEU score)
    - Select the best-performing candidate model checkpoint.
- **Metadata**:
    - Track experiment parameters
    - Store logs, metrics, and training configs for reproducibility.

Example: Select the model with best validation BLEU score and lowest inference latency.

e. Deployment

Goal: Serve the selected LLM model for inference.

- **Deployment Ready Model**:
    - Convert model to optimized format if needed.
    - Package with inference API (FastAPI)
- **Inference Code**:
    - Wrap model logic for real-time or batch prediction.
    - Add input/output validation, logging, and batching logic.

Example: Deploy on AWS with Docker , Kubernetes & REST API.

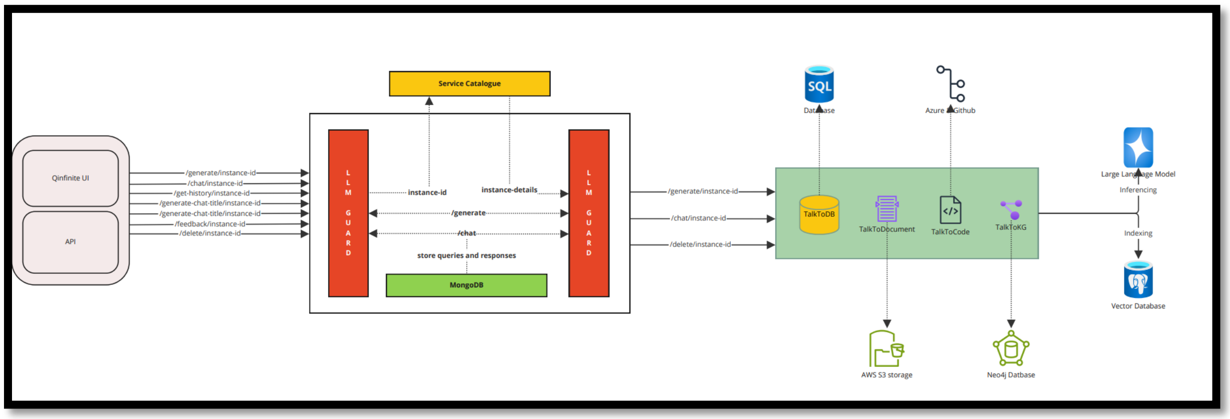
f. Monitoring

Goal: Continuously observe and maintain model performance in production.

- **Code and Model in Production**:
  - Logging real-time usage, latency, throughput, and errors.
  - Monitor prompt drift and output toxicity/bias.
- **Production Data**:
  - Gather feedback data to detect performance degradation.
  - Use human-in-the-loop (HITL) review for high-stakes outputs.

Example: Track user queries that consistently yield poor results and retrain with feedback.

8. Full Agentic Solution – System Integration & Architecture



a. Overview

This document provides an in-depth analysis of the Qinfinite Agentic Architecture, which enables intelligent agents to interact with various data modalities using a unified platform. The system supports modular agent instances, LLM-backed services, RAG workflows, and full integration with developer platforms and cloud services such as Azure, AWS, MongoDB, and GitHub.

b. High-Level Architecture Components

Method	Explanation
Qinfinite UI	Frontend user interface for managing, interacting with, and testing AI agents.
API Layer	RESTful endpoints for agent management and interactions (e.g., /generate, /chat, /feedback).
Agent Instances	Each instance encapsulates a configured prompt, vector source, and context memory.
LLM Guard	Responsible for content moderation, prompt sanitization, and safety enforcement.
Service Catalogue	Directory of available agent services and capabilities.
Feedback System	Enables collection of user corrections or ratings for continuous learning.

c. Data Storage & Integration Layers

Service	Role
MongoDB	Stores agent metadata, conversation history, instance configuration.
AWS S3	Stores document files, prompt templates, embeddings, and logs.
Neo4j	Stores and queries graph knowledge bases for knowledge-grounded conversations.
Vector Database	Embedding-based search index used in Retrieval-Augmented Generation (RAG).

d. Deployment Model

Deployment options include cloud-native environments with support for scaling, CI/CD, and monitoring.

Deployment Stack

Component	Technology
Containerization	Docker-based microservices for APIs, frontend, agents.
Orchestration	Kubernetes (AKS/EKS) with Helm charts for auto-scaling.
CI/CD	GitHub Actions or Azure DevOps pipelines with unit and integration tests.
Secrets & Vault	Managed via Azure Key Vault or AWS Secrets Manager.

e. Agent Service Overview

Each agent provides domain-specific functionality. Services are exposed via modular APIs.

Agent Type	Functionality
Talk To Document	RAG agent to extract answers from uploaded or linked documents.
Talk To DB	Natural Language to SQL generation and execution for structured DBs.
Process Automation Agent	Executes workflows and integrations (e.g., Jira ticketing, email parsing).
Code Agent	Generates code from instructions and performs basic unit test validation.

f. Monitoring & LLMOps Integrations

Operational metrics and audit logs are essential for debugging, optimization, and compliance.

Metric	Purpose
Latency & Token Usage	Track cost and performance for each API call.
Hallucination Rate	Evaluates output grounding using chunk match ratio and fact-checkers.
Prompt Effectiveness	Measures intent classification accuracy and output relevance.
Model Drift Detection	Detects shifts in input/output semantics for re-training triggers.

g. Conclusion & Future Scope

Future enhancements will include auto-prompt tuning, synthetic data generators, and advanced multi-agent collaboration.

MLOps

Model Lifecycle Management

a. **Creation**

- Data exploration, feature engineering, and algorithm selection.
- Developing training and inference code.
- Integration with Qinfinite MLCockpit framework.
- Instantiate model with custom configurations and hyperparameters.
- Models are typically containerized for reproducibility.

b. **Versioning**

- Model code, parameters, training datasets, and artifacts are versioned.
- Allows seamless comparison, rollback, and auditing.

c. **Deployment**

- Deploy models to different environments.

d. **Retirement**

- Schedule or archive models based on:
- Model staleness (last used, last retrained)
- Accuracy degradation or drift detection
- Archival of models and metadata in long-term storage

Feature Management

a. **Feature Store Integration**
    - Centralized feature storage and retrieval system
    - Versioning of feature sets and transformation logic

b. **Feature Validation & Drift Detection**
    - Feature expectations are defined (e.g., value ranges, null thresholds)

- Automatic drift detection at schema and statistical level
- Alerting on drift events and auto-trigger of retraining workflows

Model Management

a. Multi-instance Support

- MLCockpit supports concurrent management of Multiple model instances per use-case

b. Model Registry

- Add any algorithm with practically any training and inference logic, seamlessly with generic MLCockpit RESTful APIs. Agnostic of underlying framework, be it pytorch, tensorflow, etc.

Validation & Quality Assurance

a. Benchmarking

- Model performance is benchmarked across:
  - Historical datasets
  - Cross-validation splits
  - Production data samples
- Metrics tracked: Accuracy, ROC-AUC, F1, latency, throughput

b. Data Drift Detection

- Monitors Shift in input feature distribution
- Periodic retraining triggered on significant drift

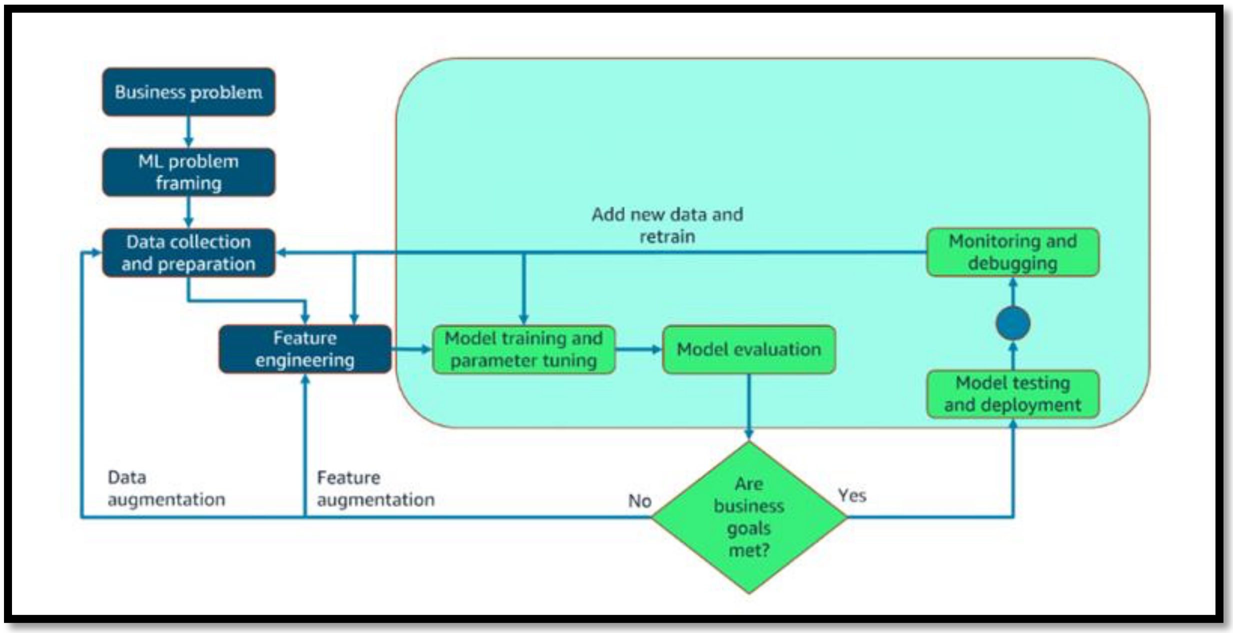
c. Dashboards & Metrics

- View and Manage models and instances
- View various metrics to take informed decisions regarding retraining and deployment

d. Model Retraining

- Support for scheduled retraining jobs (e.g., daily, weekly)
- Conditional retraining based on data drift, error rates

MLOps Pipeline



Business Problem

- Identify and clearly articulate the business need
- Determine how solving this problem will impact operations or decisions
- Example: Reduce churn rate, improve fraud detection, forecast demand

ML Problem Framing

- Translate the business goal into a machine learning problem.
- Decide on the ML task type: classification, regression, clustering, etc.
- Align success metrics with business objectives.

Data Collection & Preparation

- Source the data required to solve the problem (databases, APIs, logs, etc.).
- Perform basic cleaning: handling missing values, removing noise, type conversions.
- Format and split data for training, validation, and testing

Data Augmentation

- Apply techniques to artificially expand or balance the dataset.
- Includes oversampling, noise injection, image flipping, text paraphrasing (based on domain).
- Helps improve generalization and prevent overfitting.

Feature Engineering

- Create meaningful input variables that help the model learn patterns better
- Includes transformations like scaling, encoding, polynomial features, date decompositions
- Often involves domain knowledge

Feature Augmentation

- Introduce additional derived features or external data sources
- Techniques may include aggregations, ratios, embeddings, or features from pre-trained models
- Increases the expressiveness of input data

Model Training and Parameter Tuning

- Select and train a machine learning model using the prepared dataset.
- Use techniques like cross-validation, grid search, or Bayesian optimization for hyperparameter tuning.
- Evaluate preliminary performance metrics during this phase.

Model Evaluation

- Measure performance using chosen metrics (Accuracy, Precision, Recall, F1, ROC-AUC, etc.).
- Use validation/test datasets to gauge generalization.
- Identify biases, variance issues, or systematic errors.

Are Business Goals Met?

- Decision checkpoint.
- If evaluation results do not meet business expectations:
- Loop back to data, features, or model tuning.
- If yes, proceed to deployment.

Model Testing and Deployment

- Conduct final tests (unit, integration, shadow testing).
- Package the model (e.g., as a container or callable API).
- Deploy into staging or production environments.

Monitoring and Debugging

- Continuously track performance metrics, latency, failure rates, and data distribution in production.
- Detect model drift or operational issues (e.g., missing inputs).
- Alert stakeholders and log issues for investigation.

Add New Data and Retrain

- Incorporate fresh data periodically to adapt to evolving patterns.
- Trigger retraining schedules based on data drift, time intervals, or performance drop.
- Archive old models and compare versions.

Full MLOps System – Integration & Architecture

Overview

- Unified platform to manage:

  - Data ingestion
  - Feature pipelines
  - Model lifecycle
  - Monitoring & retraining

- Core principle: reproducibility, observability, automation

High-level Architecture Components

- Frontend: Qinfinite MLCockpit
- API layer: RESTful endpoints for integration and model management
- Service Catalogue: Training and deployment of ML models
- Model Instances: Creating, forking, deleting and archival of model instances via MLCockpit.
- Feedback System: Collection of user corrections or ratings for continuous learning

Data Storage

- MongoDB: Storing model metadata, metrics, configs and instances.
- S3: Storing model files and embedding data

Deployment Model

- Containerization: Docker-based microservices for APIs, frontend, training and inference code
- Orchestration: Kubernetes (AKS/EKS) with Helm charts for auto-scaling.
- CI/CD with GitHub Actions, Azure DevOps

Monitoring & MLOps Integrations

- Full tracing from ingestion to inference
- Model health dashboards with metrics and alerting

Conclusion & Future Scope

- Support for RLHF-style retraining
- Improved cost tracking and model efficiency analysis

Get Started with Qinfinite

QINFINITE