AgroFarmServ

Agriculture Farmer Services

Team

Divya Vemula
Mani Krishna Tippani
Rahul Chauhan
Shaheryar Nadeem

Introduction

Farmers around the world, particularly those with limited landholdings, face significant challenges in accessing essential agricultural services such as tilling, sowing, and harvesting at the right time and at fair prices.

These services are often controlled by middlemen, who impose high markups, reducing farmers' earnings and, in many cases, forcing them to sell at a loss.

The lack of direct access to service providers and fair marketplaces limits farmers’ ability to maximize their productivity and profitability, ultimately affecting their livelihoods and food security.

Challenges Faced by Farmers

- **Lack of Trust in Service Providers**: Delays in obtaining critical farm services.
- **Pricing Issues:** Inflated costs due to middlemen cuts.
- **Geographical Hurdles:** Customers in various states or regions often face challenges accessing reliable and high-quality services due to limited availability in their area

Mission statement

- **Our mission is to Empower farmers** by providing direct and real time access to service providers through a robust digital platform.
- **Eliminate exploitative middlemen** to ensure fair pricing and timely agricultural services.
- **Create a trusted marketplace** where farmers and service providers can connect, collaborate, and conduct business seamlessly.
- **Maximize farmers' margins** and support sustainable growth in agriculture.

Plan of Action

Business Model - We leverage generative AI to create a seamless and secure platform where farmers and service providers can connect, transact, and grow their businesses.

Approach

- Develop a user-friendly mobile app with secure payments, user verification, feedback systems.
- Provide real-time access to services.

Operational Flow

1. User Enrollment (Farmers & Service Providers)
2. Service Listing & Pricing
3. Order Placement & Secure Payment
4. Feedback & Review Mechanism

Database Design & Simulation

Collected tables information

User Management

- user_type: Defines different types of users (e.g., Farmers, Service Providers).
- user: Stores user details like name, contact, and verification status.
- user_address: Stores addresses linked to users.
- address_type: Categorizes addresses (e.g., Home, Business).

Services Management

- service: Contains different agricultural services.
- user_service: Links services to users, including pricing and availability.

Order and Payment Management

- user_order: Tracks orders placed by users.
- payment_details: Stores transaction details.
- order_service: Manages service fulfillment for orders.

Support & Notifications

- support_ticket: Handles customer service issues.
- user_notification: Manages system notifications.

User Table Information

Column Name	Data Type	Constraints	Purpose
user_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each user. Automatically increments for new records.
user_type_id	INTEGER	NOT NULL, FOREIGN KEY referencing user_type(user_type_id) ON DELETE CASCADE	Links the user to a specific user type (e.g., farmer, buyer, admin).
first_name	TEXT	NOT NULL	Stores the user's first name. Required field.
last_name	TEXT	NOT NULL	Stores the user's last name. Required field.
org_name	TEXT	NULL	Stores the organization name (if applicable). Optional for individual users.
mobile_number	TEXT	UNIQUE	Stores the user's mobile number. Must be unique for each user.
email	TEXT	UNIQUE	Stores the user's email address. Must be unique for each user.
aadhaar_number	TEXT	UNIQUE	Stores the user's Aadhaar number (Indian identification). Must be unique for each user.
pan_number	TEXT	UNIQUE	Stores the user's PAN number (Indian tax identification). Must be unique for each user.
verification_status	TEXT	DEFAULT 'Pending', CHECK (verification_status IN ('Pending', 'Verified', 'Rejected'))	Tracks the verification status of the user. Defaults to 'Pending'.
created	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the user was created. Automatically set to the current time.
created_by	TEXT	-	Stores the name of the user/admin who created this record.
modified	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the user was last modified. Automatically updated.
modified_by	TEXT	-	Stores the name of the user/admin who last modified this record.
is_active	BOOLEAN	DEFAULT 1	Indicates whether the user account is active (1) or inactive (0). Defaults to active.

User Type Table Information

Column Name	Data Type	Constraints	Purpose
user_type_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each user type. Automatically increments for new records.
user_type	TEXT	NOT NULL, UNIQUE	Stores the type of user (e.g., farmer, buyer, admin). Must be unique. Required field.
user_description	TEXT	-	Stores the description of the user type. Optional field.

User Address Table Information

Column Name	Data Type	Constraints	Purpose
user_address_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each address. Automatically increments for new records.
user_id	INTEGER	NOT NULL, FOREIGN KEY referencing user(user_id) ON DELETE CASCADE	Links the address to a specific user. Cascades on delete to maintain referential integrity.
address_type_id	INTEGER	NOT NULL, FOREIGN KEY referencing address_type(address_type_id) ON DELETE CASCADE	Links the address to a specific address type (e.g., home, office). Cascades on delete.
line_1	TEXT	NOT NULL	Stores the first line of the address (e.g., street name and number). Required field.
line_2	TEXT	NULL	Stores the second line of the address (e.g., apartment number). Optional field.
city	TEXT	NOT NULL	Stores the city of the address. Required field.
state	TEXT	NOT NULL	Stores the state of the address. Required field.
zip_code	TEXT	NOT NULL	Stores the ZIP code of the address. Required field.
country	TEXT	NOT NULL	Stores the country of the address. Required field.
created	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the address was created. Automatically set to the current time.
created_by	TEXT	-	Stores the name or ID of the user/admin who created this record.
modified	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the address was last modified. Automatically updated.
modified_by	TEXT	-	Stores the name of the user/admin who last modified this record.

Address Type Table Information

Column Name	Data Type	Constraints	Purpose
address_type_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each address type. Automatically increments for new records.
address_type	TEXT	NOT NULL, UNIQUE	Stores the type of address (e.g., home, office). Must be unique. Required field.
address_description	TEXT	-	Stores the description of the address type. Optional field.

Service Table Information

Column Name	Data Type	Constraints	Purpose
service_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each service. Automatically increments for new records.
service_name	TEXT	NOT NULL, UNIQUE	Stores the name of the service. Must be unique. Required field.
service_description	TEXT	-	Stores the description of the service. Optional field.
unit	TEXT	NOT NULL	Stores the unit of measurement for the service (e.g., kg, hour). Required field.
avg_price	REAL	CHECK (avg_price >= 0)	Stores the average price of the service. Must be a non-negative value.
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the service was created. Automatically set to the current time.
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the service was last updated. Automatically updated.
service_is_active	BOOLEAN	DEFAULT 1	Indicates whether the service is active (1) or inactive (0). Defaults to active.

User Service Table Information

Column Name	Data Type	Constraints	Purpose
user_service_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each user-service record. Automatically increments for new records.
user_id	INTEGER	NOT NULL, FOREIGN KEY referencing user(user_id) ON DELETE CASCADE	Links the service to a specific user. Cascades on delete to maintain referential integrity.
service_id	INTEGER	NOT NULL, FOREIGN KEY referencing service(service_id) ON DELETE CASCADE	Links the service to a specific service type. Cascades on delete.
price	REAL	NOT NULL, CHECK (price >= 0)	Stores the price of the service. Must be a non-negative value.
discount	REAL	DEFAULT 0.00, CHECK (discount >= 0)	Stores the discount applied to the service. Defaults to 0.00 and must be non-negative.
unit	TEXT	NOT NULL	Stores the unit of measurement for the service (e.g., kg, hour). Required field.
service_location	TEXT	-	Stores the location where the service is provided. Optional field.
detail_1	TEXT	-	Additional details about the service. Optional field.
detail_2	TEXT	-	Additional details about the service. Optional field.
created	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the user-service record was created. Automatically set.
created_by	TEXT	-	Stores the name of the user/admin who created this record.
modified	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the user-service record was last modified. Automatically updated.
modified_by	TEXT	-	Stores the name of the user/admin who last modified this record.
is_active	BOOLEAN	DEFAULT 1	Indicates whether the user-service record is active (1) or inactive (0). Defaults to active.

User Order Table Information

Column Name	Data Type	Constraints	Purpose
user_order_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each order. Automatically increments for new records.
user_id	INTEGER	NOT NULL, FOREIGN KEY referencing user(user_id) ON DELETE CASCADE	Links the order to a specific user. Cascades on delete to maintain referential integrity.
user_address_id	INTEGER	NOT NULL, FOREIGN KEY referencing user_address(user_address_id) ON DELETE CASCADE	Links the order to a specific address. Cascades on delete.
order_status	TEXT	NOT NULL, CHECK (order_status IN ('Pending', 'Processing', 'Shipped', 'Completed', 'Cancelled'))	Tracks the status of the order. Must be one of the specified values.
payment_status	TEXT	NOT NULL, CHECK (payment_status IN ('Pending', 'Paid', 'Failed', 'Refunded'))	Tracks the payment status of the order. Must be one of the specified values.
payment_method	TEXT	NOT NULL, CHECK (payment_method IN ('Credit Card', 'Debit Card', 'PayPal', 'Cash', 'Bank Transfer'))	Specifies the payment method used for the order. Must be one of the specified values.
order_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the order was placed. Automatically set to the current time.
shipping_address	TEXT	NOT NULL	Stores the shipping address for the order. Required field.
billing_address	TEXT	NOT NULL	Stores the billing address for the order. Required field.
order_feedback	TEXT	-	Stores feedback provided by the user about the order. Optional field.
detail_1	TEXT	-	Additional details about the order. Optional field.
modified	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the order was last modified. Automatically updated.
modified_by	TEXT	-	Stores the name of the user/admin who last modified this record.

Order Service Table Information

Column Name	Data Type	Constraints	Purpose
order_service_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each order-service record. Automatically increments for new records.
user_order_id	INTEGER	NOT NULL, FOREIGN KEY referencing user_order(user_order_id) ON DELETE CASCADE	Links the record to a specific order. Cascades on delete to maintain referential integrity.
user_service_id	INTEGER	NOT NULL, FOREIGN KEY referencing user_service(user_service_id) ON DELETE CASCADE	Links the record to a specific service. Cascades on delete.
quantity	INTEGER	NOT NULL, CHECK (quantity > 0)	Stores the quantity of the service ordered. Must be greater than 0.
service_start_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the start date of the service. Automatically set to the current time.
service_end_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the end date of the service. Automatically set to the current time.
tracking_number	TEXT	UNIQUE	Stores the tracking number for the service. Must be unique for each record.
service_feedback	TEXT	-	Stores feedback provided by the user about the service. Optional field.
service_rating	INTEGER	CHECK (service_rating BETWEEN 1 AND 5)	Stores the rating given by the user for the service. Must be between 1 and 5.
review_status	TEXT	DEFAULT 'Pending', CHECK (review_status IN ('Pending', 'Submitted'))	Tracks the status of the review. Defaults to 'Pending'.
location_coordinates	TEXT	-	Stores the geographic coordinates of the service location. Optional field.
created	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the order-service record was created. Automatically set.
created_by	TEXT	-	Stores the name or ID of the user/admin who created this record.

Payment Table Information

Column Name	Data Type	Constraints	Purpose
payment_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each payment. Automatically increments for new records.
user_order_id	INTEGER	NOT NULL, FOREIGN KEY referencing user_order(user_order_id) ON DELETE CASCADE	Links the payment to a specific order. Cascades on delete to maintain referential integrity.
payment_amount	REAL	NOT NULL, CHECK (payment_amount >= 0)	Stores the payment amount. Must be a non-negative value.
payment_method	TEXT	NOT NULL, CHECK (payment_method IN ('Credit Card', 'Debit Card', 'PayPal', 'Cash', 'Bank Transfer'))	Specifies the payment method used. Must be one of the specified values.
payment_status	TEXT	NOT NULL, CHECK (payment_status IN ('Pending', 'Completed', 'Failed', 'Refunded'))	Tracks the payment status. Must be one of the specified values.
transaction_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the payment was made. Automatically set to the current time.

User Notification Table Information

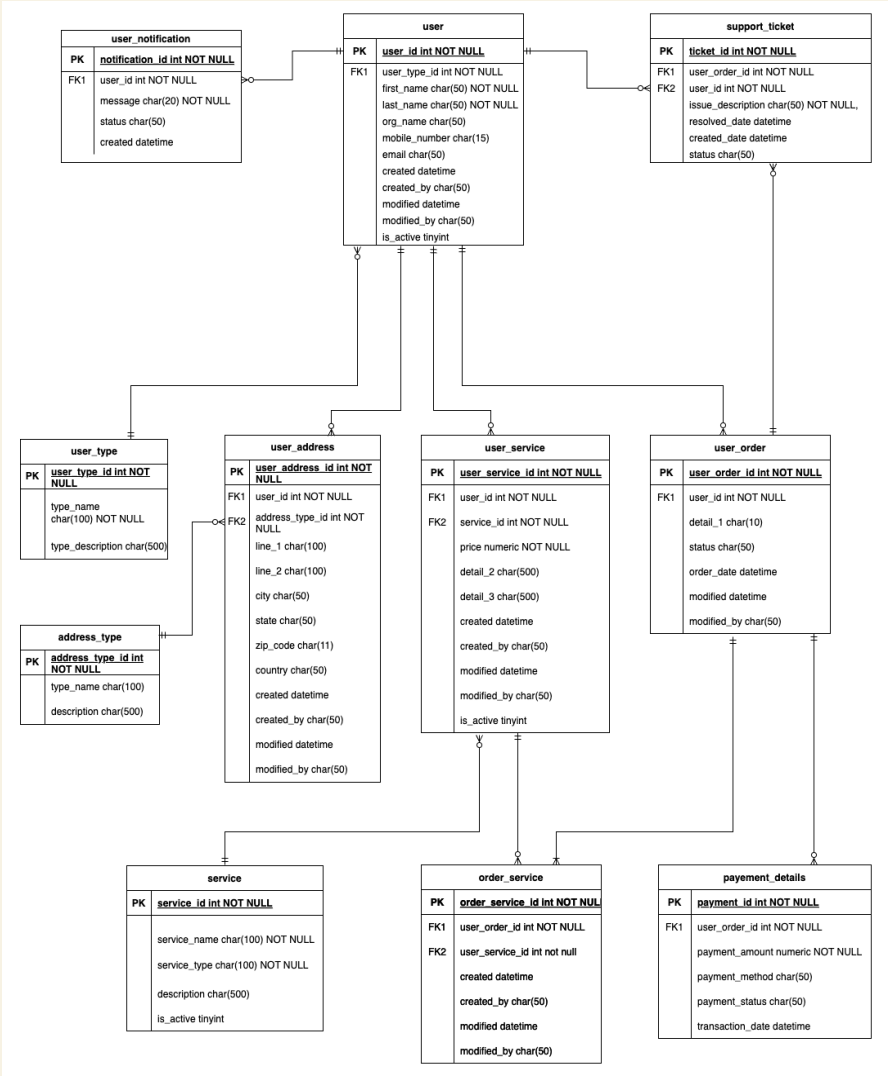
Column Name	Data Type	Constraints	Purpose
notification_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each notification. Automatically increments for new records.
user_id	INTEGER	NOT NULL, FOREIGN KEY referencing user(user_id) ON DELETE CASCADE	Links the notification to a specific user. Cascades on delete to maintain referential integrity.
message	TEXT	NOT NULL	Stores the notification message. Required field.
status	TEXT	DEFAULT 'Unread', CHECK (status IN ('Unread', 'Read'))	Tracks the status of the notification. Defaults to 'Unread'.
created_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the notification was created. Automatically set to the current time.

Support Ticket Table Information

Column Name	Data Type	Constraints	Purpose
ticket_id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique identifier for each support ticket. Automatically increments for new records.
user_id	INTEGER	NOT NULL, FOREIGN KEY referencing user(user_id) ON DELETE CASCADE	Links the ticket to a specific user. Cascades on delete to maintain referential integrity.
user_order_id	INTEGER	NOT NULL, FOREIGN KEY referencing user_order(user_order_id) ON DELETE CASCADE	Links the ticket to a specific order. Cascades on delete.
issue_description	TEXT	NOT NULL	Stores the description of the issue. Required field.
status	TEXT	DEFAULT 'Open', CHECK (status IN ('Open', 'In Progress', 'Resolved'))	Tracks the status of the ticket. Defaults to 'Open'.
created_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	Records the timestamp when the ticket was created. Automatically set to the current time.
resolved_date	DATETIME	-	Records the timestamp when the ticket was resolved. Optional field.

Entity-Relationship

Design Diagram



Entity-Relationship Design Explanation

User and Address (One-to-Many)

- A user can have zero or many addresses.
- This is represented by the user_id in the user_address table, linking multiple addresses to a single user.

User and Services (Many-to-Many)

- A user can request zero or many services.
- A service can be requested by many users.
- This is a many-to-many relationship, which is handled by the user_service table.

User and Orders (One-to-Many)

- A user can place zero or many orders.
- An order belongs to one user.

Order and Payments (One-to-One)

- Each order has exactly one payment record.
- A payment is linked to one order.

Order and Order Status (One-to-Many)

- An order can go through multiple status changes over time (e.g., pending, shipped, delivered).
- The order_status table tracks these changes.

User and Support Tickets (One-to-Many)

- A user can raise zero or many support tickets.
- Each ticket belongs to one user.

Data Collection And Preprocessing

Logic behind data simulation process

- **Data Simulation & Relevance:** **Faker** library to generate **10,000+ synthetic records** for users, services, orders, payments, and notifications.

Design Choice	Description	Why
Modular and Functional	Code divided into reusable functions.	Improves readability, reusability, maintainability.
Realistic Data Simulation	**Uses Faker, seasonality, and outliers**.	**Ensures data mimics real-world scenarios**.
Normalized Database	Schema follows 3NF to avoid redundancy.	Reduces redundancy and ensures data integrity.
Indexes for Performance	Indexes on email, mobile_number, etc.	Improves query performance and scalability.
Referential Integrity	Foreign keys ensure consistency across tables.	Maintains data consistency and prevents orphans.
Data Privacy and Security	**Sensitive data hashed using SHA-256.**	**Protects sensitive information.**
Probabilistic Distributions	Weighted choices and probabilistic fields.	Simulates non-uniform, real-world distributions.
Error Handling and Validation	Checks for empty tables and validates input.	Ensures data consistency and prevents runtime errors.

User Management Tables Information

Table: user_type
Columns: ['user_type_id', 'user_type', 'user_description']

user_type_id	user_type	user_description
1	farmer	Individual managing agricultural activities
2	vendor	Supplier providing services or products
3	admin	Administrator managing the platform
4	farmer_and_vendor	Individual both farming and providing services

==========

Table: address_type
Columns: ['address_type_id', 'address_type', 'address_description']

address_type_id	address_type	address_description
1	Residential	Residential address
2	Permanent	User permanent address
3	Bill_To	Billing address
4	Ship_To	Shipping address
5	Sold_To	Selling address

Table: user
Columns: ['user_id', 'user_type_id', 'first_name', 'last_name', 'org_name', 'mobile_number', 'email', 'aadhaar_number', 'pan_number', 'verification_status', 'created', 'created_

user_id	user_type_id	first_name	last_name	org_name	mobile_number	email	aadhaar_number	pan_number	verification_status
1	1	Calvin	Jacobs	Jackson Group	001-938-327-7314x73029	gina97@example.com	903240188916	1707074014	Verified
2	1	Ann	Thomas	Walker-Barr	(483)810-8494	millermonica@example.org	405922972722	1758229197	Verified
3	2	Stephanie	Young	Ray, Jones and Garcia	+1-863-850-0909x271	craig08@example.org	451046275119	4667994694	Verified
4	3	Michael	Johnson		278.469.1328x0282	lindaevans@example.com	946593324108	5729491361	Rejected
5	1	Elizabeth	Randall	Bennett-Ramirez	(400)778-4710x581	craig79@example.com	516892727861	806292086	Verified

==========

Table: user_address
Columns: ['user_address_id', 'user_id', 'address_type_id', 'line_1', 'line_2', 'city', 'state', 'zip_code', 'country', 'created', 'created_by', 'modified', 'modified_by']

user_address_id	user_id	address_type_id	line_1	line_2	city	state	zip_code	country	created	cr
1	4176	3	24893 Barker Manor	Suite 176	Perezhaven	Vermont	70017	Mayotte	2025-03-01 03:21:18	Marv
2	5208	5	6091 English Court Apt. 628	Suite 609	Port Johntown	Delaware	09769	Oman	2025-03-01 03:21:18	Nic
3	5927	1	486 Hernandez Falls	Apt. 810	Joelshire	Minnesota	67282	Serbia	2025-03-01 03:21:18	Terr
4	2929	5	99782 Natalie Shoals Suite 108	Apt. 957	North Lindafort	West Virginia	40834	Kuwait	2025-03-01 03:21:18	Tayl
5	1744	4	2666 Erin Valleys Apt. 354	Apt. 014	Lisashire	Washington	76508	Cape Verde	2025-03-01 03:21:18	Tina

Service Management Tables Information

Table: user_service
Columns: ['user_service_id', 'user_id', 'service_id', 'price', 'discount', 'unit', 'service_location', 'detail_1', 'detail_2', 'created', 'created_by', 'modified', 'modified_by'

user_service_id	user_id	service_id	price	discount	unit	service_location	
1	1683	1	454.58	15.47	per acre	92079 Gregory Mall Suite 768, Williamport, MA 63838	Mind right guy deep
2	3962	8	109.32	17.68	per service	USNV Torres, FPO AA 26212	Every hold major cause. Enough sister amount
3	6052	1	97.74	80.22	per hour	70344 Morales Terrace Suite 295, Bensonhaven, MA 38334	Leg even keep interest could.
4	8900	9	376.26	1.66	per hour	2045 Knight Station Suite 195, Cathyshire, NJ 21761	
5	1398	7	492.16	3.46	per hour	70162 Lisa Valley Apt. 667, East Preston, PR 97149	Busines Defense fall you hospit

Table: service
Columns: ['service_id', 'service_name', 'service_description', 'unit', 'avg_price', 'created_at', 'updated_at', 'service_is_active']

service_id	service_name	service_description	unit	avg_price	created_at	updated_at	service_is_active
1	Crop Consulting	Expert advice on crop selection, planning, and management.	per hour	159.12	2022-12-31	2024-01-31	1
2	Irrigation Services	Customized solutions for efficient water distribution.	per acre	153.4	2024-02-01	2023-06-24	1
3	Fertilizer Supply	High-quality fertilizers to boost crop yields.	per ton	61.97	2020-07-11	2022-04-20	1
4	Pest Control	Safe and effective methods to protect crops from pests.	per acre	284.68	2022-08-26	2022-01-04	1
5	Farm Equipment Rental	Affordable rental options for modern farming equipment.	per day	488.44	2023-12-10	2023-02-25	0

Order Management Tables Information

Table: user_order
Columns: ['user_order_id', 'user_id', 'user_address_id', 'order_status', 'payment_status', 'payment_method', 'order_date', 'shipping_address', 'billing_address', 'order_feedback'

user_order_id	user_id	user_address_id	order_status	payment_status	payment_method	order_date	shipping_address	
1	4413	4650	Cancelled	Pending	Credit Card	2025-01-06	564 Craig Center Apt. 384, Leahmouth, KS 59012	9821 John
2	8852	8816	Shipped	Paid	Cash	2025-01-02	65596 Byrd Inlet Suite 500, Port Leeland, MN 43290	394 Bra
3	1129	5629	Processing	Pending	Credit Card	2025-01-01	126 Paul Causeway Apt. 328, East Matthewton, TN 56168	7391
4	9928	1128	Completed	Paid	Debit Card	2025-01-16	870 Joyce Neck Suite 764, South Andrewburgh, OH 87769	570 Fl
5	6047	6672	Completed	Pending	Debit Card	2025-02-02	4459 Kimberly Vista Apt. 143, New Veronicahaven, SD 56776	

==========

Table: payment_details
Columns: ['payment_id', 'user_order_id', 'payment_amount', 'payment_method', 'payment_status', 'transaction_date']

payment_id	user_order_id	payment_amount	payment_method	payment_status	transaction_date
5	1307	254.21	Credit Card	Failed	2025-02-22
6	7678	543.04	Debit Card	Pending	2025-01-11
9	8106	829.51	Debit Card	Pending	2025-02-04
10	6829	613.05	Credit Card	Failed	2025-01-18
13	1888	317.39	Debit Card	Pending	2025-02-21

Table: order_service
Columns: ['order_service_id', 'user_order_id', 'user_service_id', 'quantity', 'service_start_date', 'service_end_date', 'tracking_number', 'service_feedback', 'service_rating',

order_service_id	user_order_id	user_service_id	quantity	service_start_date	service_end_date	tracking_number	
1	4779	4319	7	2023-07-07	2024-11-11	3464913890	Policy success paper money also while. N Feel whom along
2	205	2499	9	2023-12-12	2024-06-30	5085536782	Similar before single big.
3	3579	4882	2	2023-03-18	2023-06-03	4070006621	Office table heart ever cup. Read national management
4	9878	7452	7	2023-09-23	2024-07-12	6402212411	Marriage media and hospital note. Adm
5	4327	7767	3	2023-03-06	2023-06-08	9418657830	Across agency third month president Arm i

Support & Notifications Tables Information

==========

Table: support_ticket
Columns: ['ticket_id', 'user_id', 'user_order_id', 'issue_description', 'status', 'created_date', 'resolved_date']

ticket_id	user_id	user_order_id	issue_description
2	2816	4800	Together occur easy east make admit beat.
5	4250	9046	Activity true scene grow court be. Tax spring lose contain let. Those some evening far place trial. Point among dog arm. Trouble two treatment heart.
6	722	8142	Material party blue pass management someone. Maybe unit key last interesting each under. Woman modern save ability still visit. Statement between subject enough then treatment instead. Quality poor owner choice idea mention fr
9	7051	1613	Million effect kitchen get market. Along simply various usually hair discuss. Fish traditional despite measure spend tough enter TV.
10	4982	5988	History amount PM him up out. Travel game court network. Two their sister mission. National fill program lawyer claim trial new example. Especially happen tough record point. Appear bill above ever night. Idea sure war Mr. Break break customer born.

==========

Table: user_notification
Columns: ['notification_id', 'user_id', 'message', 'status', 'created_date']

notification_id	user_id	message
1	3261	Father doctor network single situation. Forget surface message list.
2	6345	Speech any ask represent behavior film west. For book view. Leader world popular support name. Article plan ten. Beat instead impact help others administration. Bring mind type allow test. Check assume discussion PM well afte
3	3762	Away fill leave fire ok. Over artist activity however wait west start. Consider recognize cover scene fill down.
4	1649	Management exist few throughout what edge glass. Notice stuff perhaps teacher. Become try suffer itself white main thank view. Ability shake everyb
5	1943	Individual car popular the spend. Consumer audience idea guess teach. Summer police maintain physical whom itself soldier. Short several shoulder someone sometimes.

Logic behind data simulation process

Table 1: Data Generation Process (User Data to Service Data)

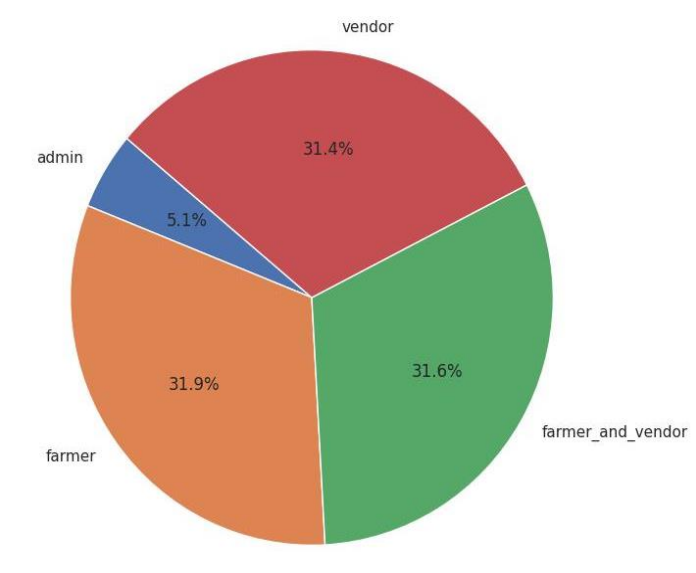
Data Table	Generated Data	Generation Method
User Data	- User Type: farmer, vendor, admin, farmer_and_vendor	- Random selection from redefined user types
	- First Name, Last Name, Email, Mobile Number	- Generated using the Faker library for realistic names, emails, and phone numbers.
	- Verification Status: Verified, Pending, Rejected	- Random selection with weighted probabilities using random.choices().
	- Is Active: True, False	- 80% chance of being active (np.random.binomial(n=1, p=0.8)).
Address Data	- User-Address Mapping: Associated with random user IDs	- Random user selection for each address.
	- Address Details: address_line_1, city, state, zip_code, etc.	- Generated using Faker library for realistic address data.
	- Address Type: Residential, Permanent, etc.	- Random selection from predefined address types.
Service Data	- Service Name: Crop Consulting, Irrigation Services, etc.	- Predefined service names.
	- Service Price: Price range (e.g., 50−50−500)	- Random number generation within a range, with occasional outliers (e.g., 1000−1000−5000).
	- Service Active: True, False	- 80% chance of being active (np.random.binomial(n=1, p=0.8)).
User-Service Mapping	- User-Service Associations: Users may have multiple services	- Random selection of services for each user.
	- Service Prices: Based on service price and noise (random variation)	- Service prices adjusted with a random factor for variability (add_noise() function).

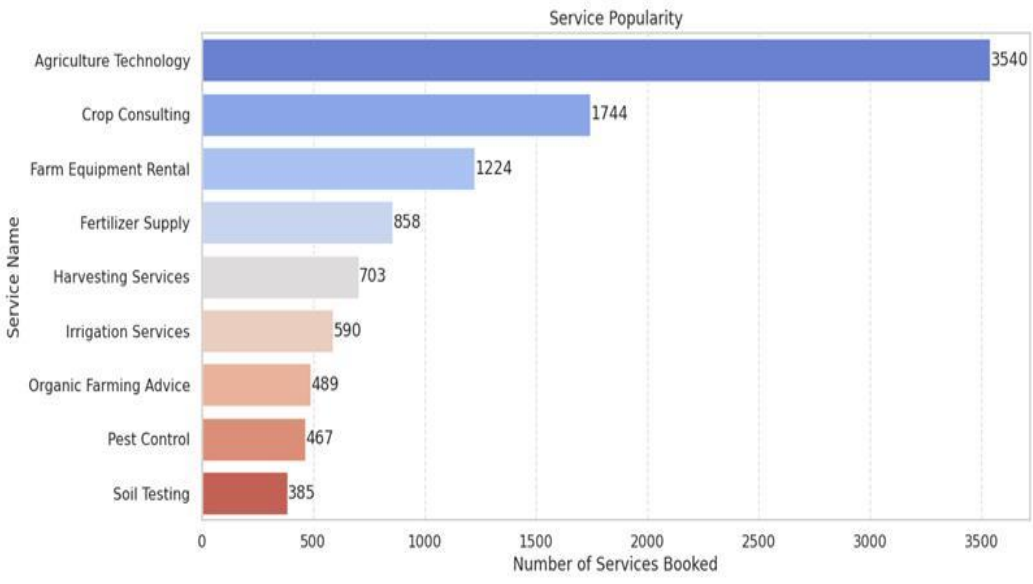
Table 2: Data Generation Process (Order, Support Ticket Data to Key Methods)

Data Table	Generated Data	Generation Method
Order Data	- Order Details: User ID, Service ID, Order Date	- Random selection of user-service pairs with random order date within a date range.
	- Order Status: Pending, Processing, Shipped, Completed, Cancelled	- Random assignment with weighted probabilities (weighted_choice() function).
	- Payment Status: Pending, Paid, Failed, Refunded	- Random assignment with weighted probabilities (weighted_choice() function).
	- Payment Method: Credit Card, Debit Card, PayPal, Cash, Bank Transfer	- Random assignment with weighted probabilities (weighted_choice() function).
	- Order Value: Adjusted for seasonality	- Base value adjusted based on the month (apply_seasonality() function).
Support Ticket Data	- Issue Type: Service-related issues, billing issues	- Random issue type assignment.
	- Ticket Status: Open, In Progress, Resolved	- Random assignment with weighted probabilities (weighted_choice() function).
	- Resolved Date: Random date if status is Resolved	- Random date generation using Faker.
Notification Data	- Notification Type: Service update, Payment reminder, etc.	- Predefined notification types.
	- Status: Read, Unread	- Random selection with weighted probabilities (weighted_choice() function).
Key Data Generation Methods	- Randomness: Data like user type, service price, issue type, etc. use randomness to simulate real-world uncertainty.	- Achieved using random.choices() and random number generation.
	- Predefined Options and Weights: Fields like user type, address type are selected from predefined options with associated probabilities.	- Weighted random selection using random.choices().
	- Noise & Outliers: Noise and outliers are added in certain fields (e.g., price, ratings) to make the data more realistic.	- Added by introducing random noise (add_noise()) or extreme values (generate_price_outlier(), generate_rating_outlier()).
	- Random Date Generation: Dates for orders, tickets, and services are randomly generated within a specified range.	- Random date generation using Faker (fake.date_this_year(), fake.date_between()).

Data Analysis & Insights

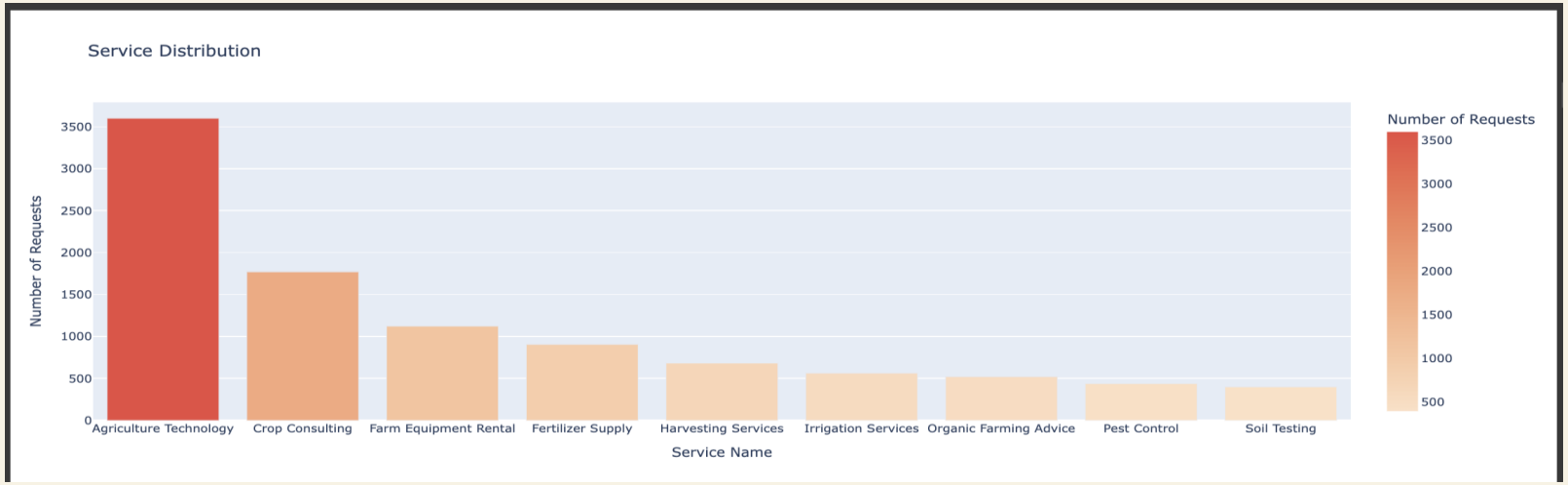
User Segmentations And Services





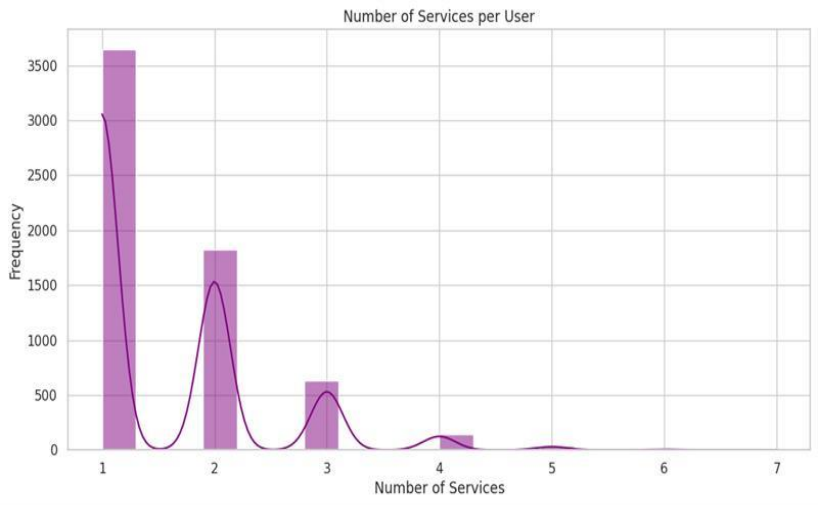
This graph helps to visualize the popularity of various agricultural services and their usage across different user segments, such as vendors, farmers, and combined farmer-and-vendor groups. It helps identify which services are in high demand and how different user groups engage with these services, providing insights for strategic decision-making and resource allocation.

Service Distribution



- **Agriculture Technology** has the highest service requests (3500+), highlighting strong reliance on agricultural technology solutions.
- **Crop Consulting, Farm Equipment Rental, and Fertilizer Supply** see high demand, reflecting the need for expert guidance and essential farming inputs.
- **Services like Pest Control and Soil Testing** have the lowest requests, indicating potential gaps in awareness or accessibility.
- **Investing** in agricultural technology, expanding advisory services, and bundling related offerings can drive service adoption and improve farming efficiency.

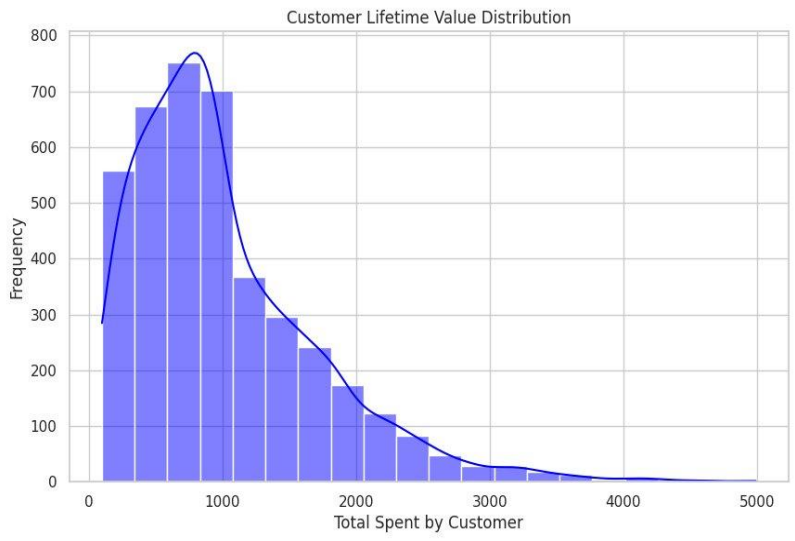
Analysis on Services offered

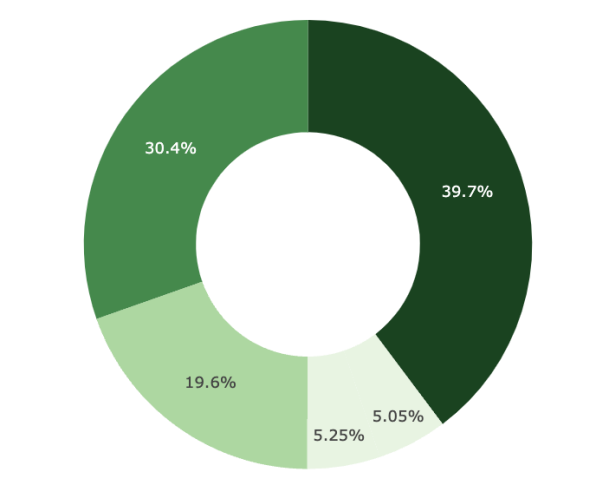


Key Insights:

- Most farmer orders include one or two services, while fewer orders involve three or more services at a time.
- Agriculture Technology dominates service requests, with high demand for Farm Equipment Rental and Fertilizer Supply, reflecting farmers' reliance on essential resources.

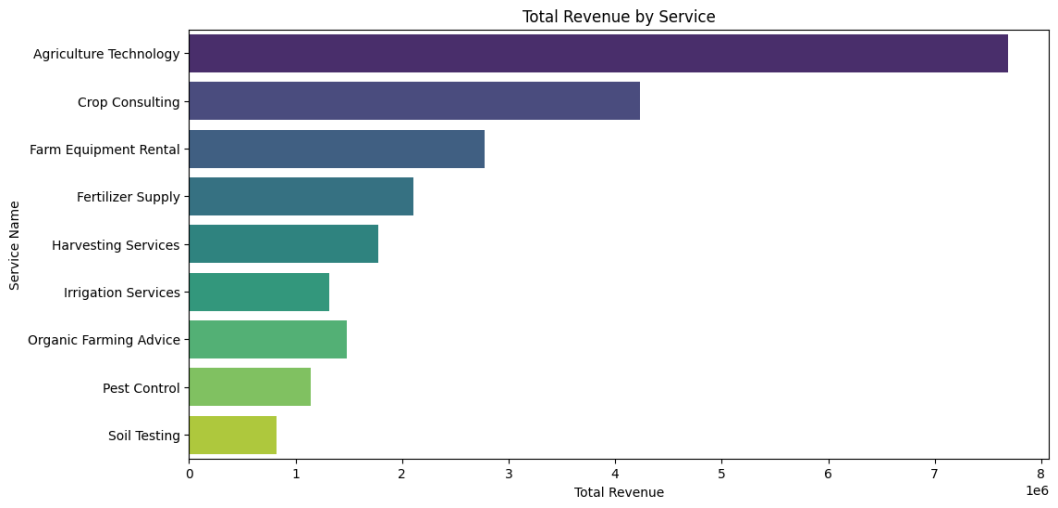
Analysis on Payment and Transactions

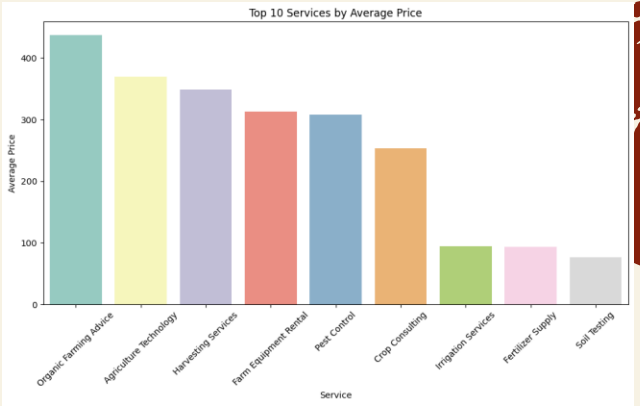




- Most customers are clustered in lower spending ranges, indicating many make smaller purchases.
- A smaller group of high spenders extends the distribution to the right.
- This right-skewed shape implies a handful of customers contribute disproportionately higher revenue.
- Credit Cards (39.7%) and Debit Cards (30.4%) are the preferred payment methods, while Cash (19.6%) remains relevant, indicating limited access to digital banking in some areas.

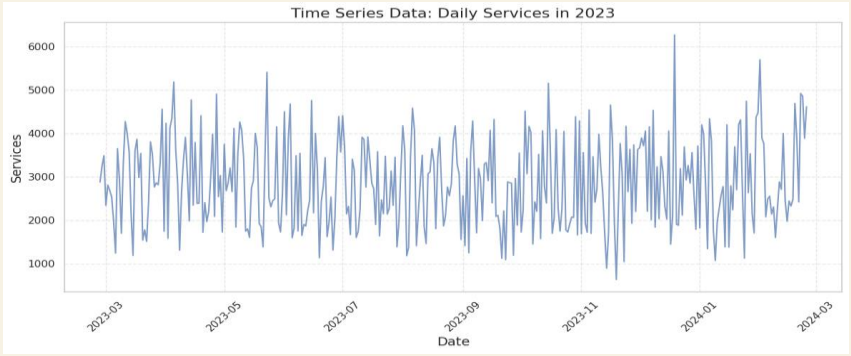
Total Revenue by Service

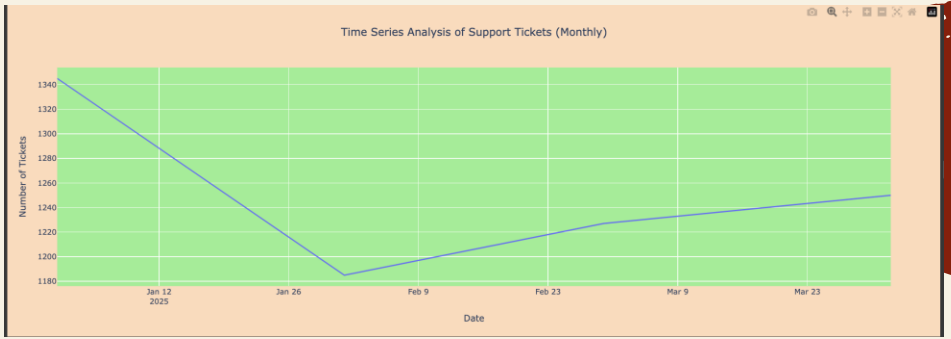


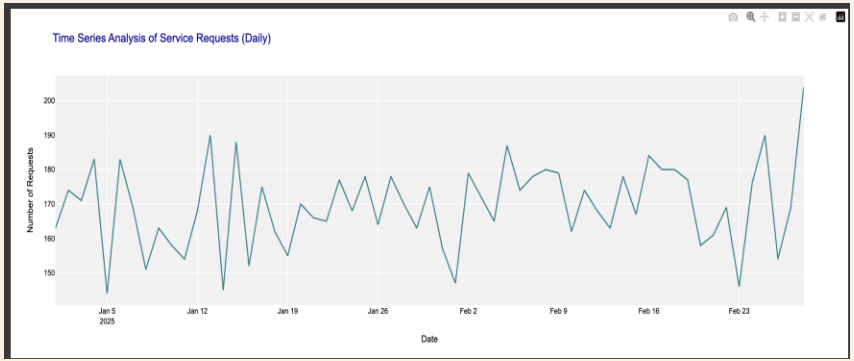


- **Agriculture Technology** generates the **highest revenue**, significantly more than any other service.
- **Crop Consulting** follows as the second **highest revenue**-generating service.
- **Farm Equipment Rental** also contributes significantly, ranking third.
- **Other services** like Fertilizer Supply, Harvesting Services, and Irrigation Services contribute **moderately to revenue.**
- Organic Farming Advice, Pest Control, and Soil Testing generate the **least revenue** among the listed services.
- This suggests that technology and consulting services dominate revenue generation, while support services like soil testing and pest control contribute less to comparison.

Time Series Data – User Engagement







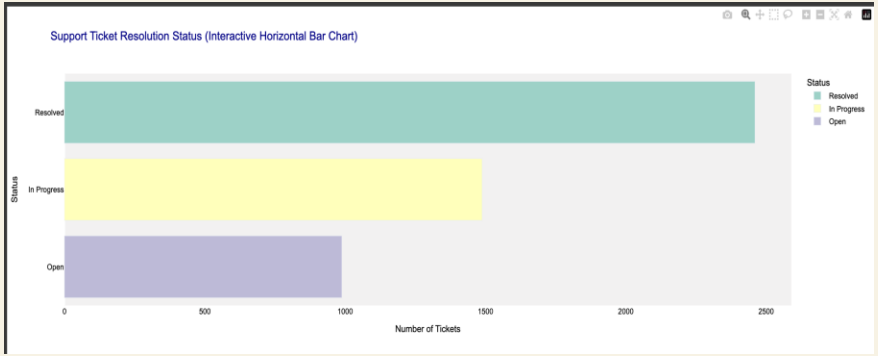
- Daily usage fluctuates reflecting periods of high and low demand.
- Occasional peaks correspond to seasonal farming activities or promotions.
- The trend indicates consistent engagement with some notable spikes in service usage.

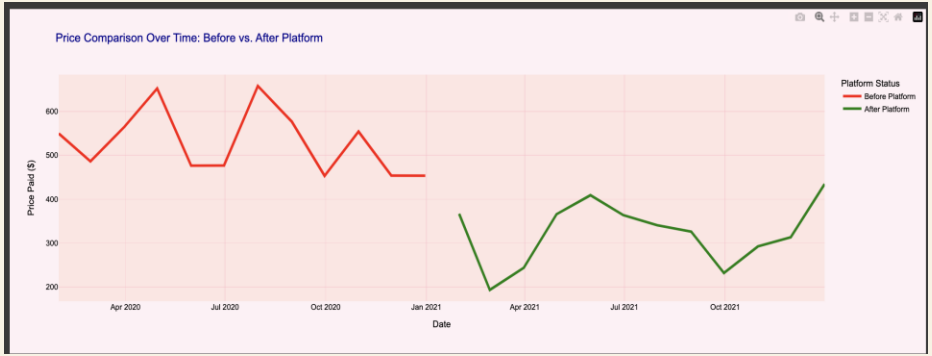
Analysis
Pain Points And Solutions

Reducing Costs for Farmers Through Direct Connectivity

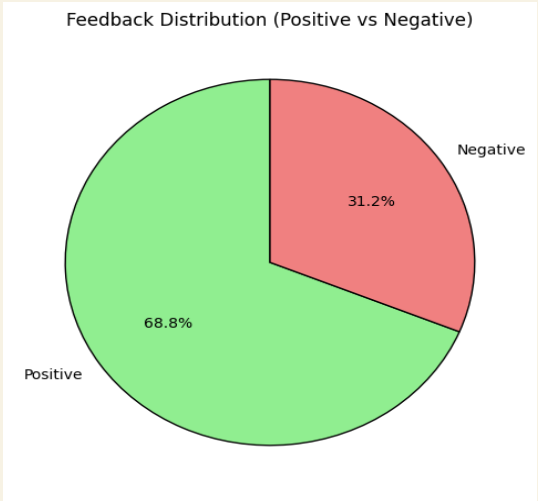
- **Eliminating Middlemen:** Previously, farmers paid higher prices due to intermediaries inflating costs.
- **Direct Cost Savings:** Our platform connects farmers directly with service providers, significantly reducing expenses and ensuring fair pricing.

Customer Trust Analysis in Service Providers

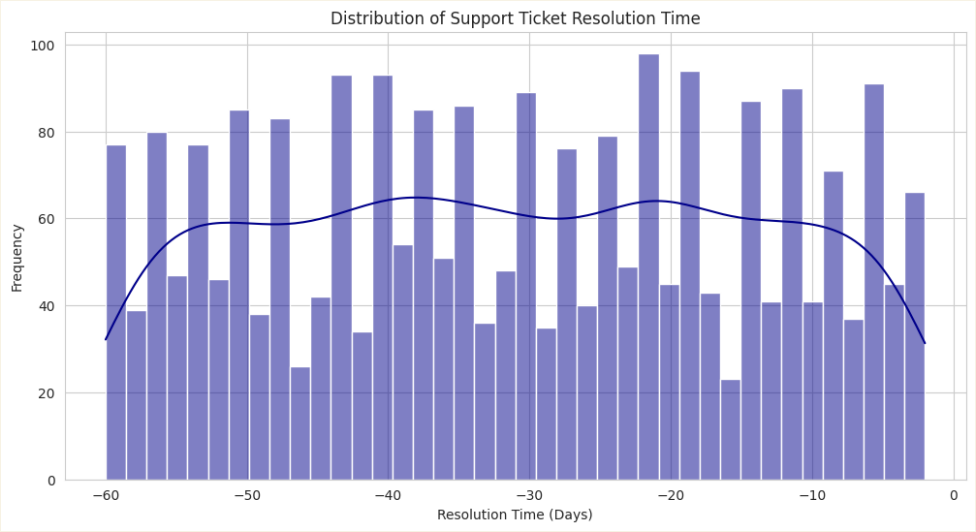


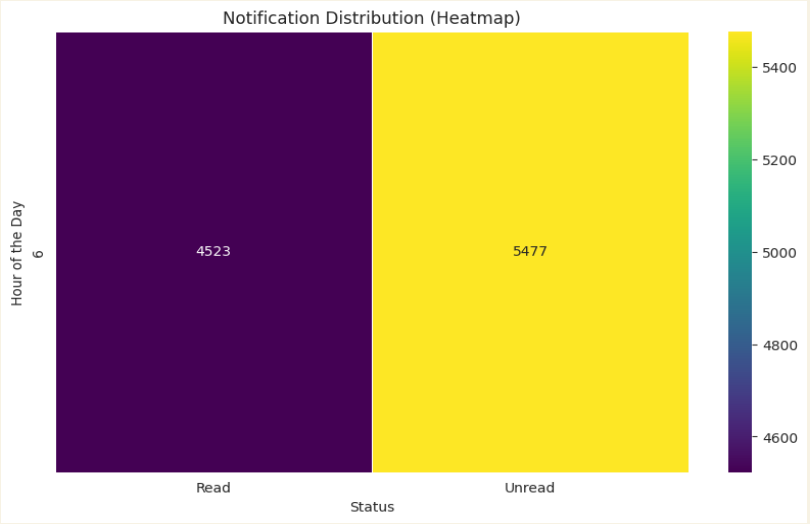


- **Improved Customer Trust:** Sentiment analysis shows a shift towards trust in service providers, with only 8.9% negative sentiment.
- **Growing Confidence:** 68.8% positive and 31.2% negative responses indicate growing confidence in services.
- **Sustaining Trust:** Addressing concerns and ensuring transparency will further enhance customer trust.



Sustaining Customer Trust





Distribution of Support Ticket Resolution Time

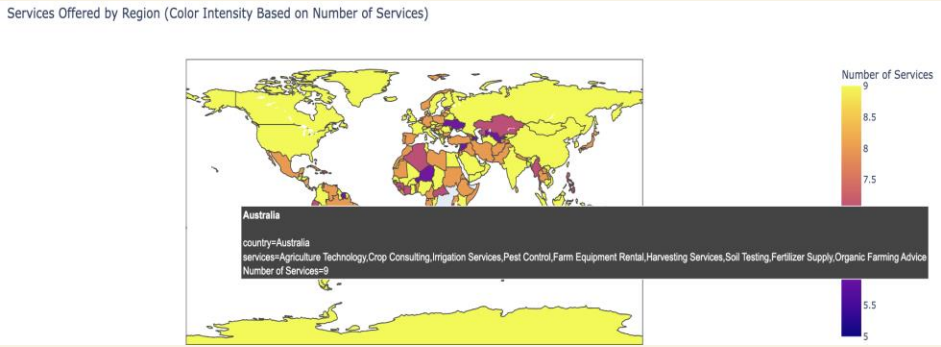
✓ No. of support tickets were resolved within specific timeframes.
✓ Most tickets cluster around a moderate resolution period, with fewer at the extremes.
✓ The orange line tracks a smoothed trend, revealing dips and peaks in resolution times.

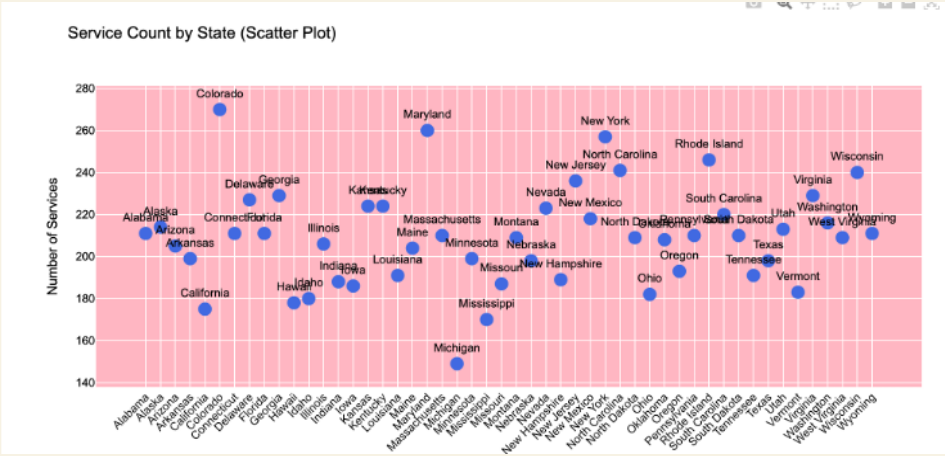
Notification Read Rates

✓ Many users don't consistently engage with or check notifications.
✓ The high unread rate points to potential improvements in notification relevance or delivery.

A Need to streamline support processes, enhance notification strategies, and address the minority of negative user experiences

Geographical Hurdles





- **Wide Coverage**: Customers in various states can easily access our services.
- **Convenience**: Reduces the need for customers to travel or rely on limited local options.
- **Scalability**: Our infrastructure supports operations across diverse locations, ensuring consistent service quality.
- **Regulatory Impact:** Some states have strict regulations on middlemen, leading farmers to rely on cooperatives or government-backed programs instead of third-party platforms.
- **State Subsidies:** Regions with subsidized agricultural services provide resources directly to farmers, reducing the need for external service requests.
- **Large-Scale Farming:** Bigger farms often have in-house service teams, minimizing outsourcing needs.
- **Small-Scale Farmer Demand:** States with more independent farmers show higher engagement, as they seek external support.
- **Strategic Insights:** Analyzing state-level trends can help identify where service demand is strongest and why

RAG Based DB System

Pain Points & RAG Solutions

Support Delays

- **Pain Point*:*** Lengthy resolution times in support tickets.
- **RAG Fix*:*** Instantly retrieve ticket history & related FAQs, generating faster, more precise resolutions.

Pricing & Service Confusion

- **Pain Point*:*** Users uncertain about updated prices, causing disputes or order cancellations.
- **RAG Fix*:*** On-demand retrieval of **current** service listings and dynamic generation of user-friendly price breakdowns.

Unread Notifications

- **Pain Point*:*** Over 80% remain unread, suggesting user disengagement.
- **RAG Fix*:*** Contextual reminders or summary messages that prompt farmers/providers to check critical updates.

Negative Feedback

- **Pain Point*:*** ~9% negative sentiment, often stemming from miscommunication or unclear service details.
- **RAG Fix*:*** Provide immediate clarifications, policy references, or step-by-step guides drawn from the database.

Support Delays Fix Through RAG Prompt

```
### RAG Pain Point Fixes ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are a security-focused AI assistant for AgroFarmServ, responsible for generating optimized SQL queries from user queries to address customer support, pricing, notificat

    ### General Guidelines:
    - Use the **provided database schema** to craft **precise SQL queries**.
    - Prioritize **JOIN operations** to fetch relevant information across multiple tables.
    - Apply **WHERE clauses** to filter records accurately based on user requests.
    - Only retrieve **non-sensitive columns** or anonymize sensitive columns using placeholder values.
    - Generate **secure SQL queries** with no direct access to sensitive information like Aadhaar numbers, PAN numbers, or payment details.
    - Ensure queries handle **NULL values** and missing data gracefully.

    ### RAG Pain Point Fixes:
    #### 1. **Support Delays**
    - Instantly retrieve **ticket history**, **ticket status**, and **related FAQs**.
    - Prioritize **active tickets** with status "Open" or "In Progress".
    - Join `support_ticket`, `user_order`, and `service` tables to link tickets with orders and services.
```

```
user_query = "Retrieve the last five support tickets"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

- **Automated Ticket Retrieval:** The system fetches the last five support tickets, prioritizing active cases ("Open" or "In Progress").
- **Efficient Query Optimization:** Uses JOIN operations to link support tickets with orders and services for better tracking.
- **Faster Issue Resolution:** Enables quick access to ticket history, status updates, and resolution times, reducing delays.
- **Improved Support Workflow:** Helps support teams prioritize unresolved tickets and streamline customer service operations

```
📌 Generated SQL Query:
 SELECT
    support_ticket.ticket_id,
    support_ticket.user_id,
    support_ticket.user_order_id,
    support_ticket.issue_description,
    support_ticket.status,
    support_ticket.created_date,
    support_ticket.resolved_date
FROM
    support_ticket
JOIN
    user_order ON support_ticket.user_order_id = user_order.user_order_id
JOIN
    service ON user_order.user_order_id = service.service_id
WHERE
    support_ticket.status IN ('Open', 'In Progress')
ORDER BY
    support_ticket.created_date DESC
LIMIT 5;
   ticket_id  user_id  user_order_id  \
0       2342      192              9

                                  issue_description       status  \
0  Believe affect matter claim past ask nothing. ...  In Progress

          created_date resolved_date
0  2025-03-01 06:40:17          None
```

Pricing & Service Confusion Fix Through RAG Prompt

```
### RAG Pain Point Fixes ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are a security-focused AI assistant for AgroFarmServ, responsible for generating optimized SQL queries from user queries to address cus

    ### General Guidelines:
    - Use the **provided database schema** to craft **precise SQL queries**.
    - Prioritize **JOIN operations** to fetch relevant information across multiple tables.
    - Apply **WHERE clauses** to filter records accurately based on user requests.
    - Only retrieve **non-sensitive columns** or anonymize sensitive columns using placeholder values.
    - Generate **secure SQL queries** with no direct access to sensitive information like Aadhaar numbers, PAN numbers, or payment details.
    - Ensure queries handle **NULL values** and missing data gracefully.

    ### RAG Pain Point Fixes:
    #### 1. **Pricing & Service Confusion**
    - Fetch the **latest service price** and **discounts** from the `user_service` and `service` tables.
    - Include **dynamic price breakdowns** with applied discounts.
    - Ensure only **active services** are shown.
```

- **Optimized SQL Queries:** The system dynamically retrieves the latest service prices and applied discounts from the database.
- **Filtered & Secure Data Retrieval:** Queries fetch only active services, ensuring accuracy while avoiding exposure of sensitive fields.
- **Regional Price Breakdown:** The system enables state-specific pricing analysis, helping users understand localized costs.
- **Improved Transparency:** The structured output provides clear price breakdowns, reducing confusion and enhancing decision-making.

```
user_query = "Show me the latest prices for all available services"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
SELECT
    service.service_name,
    user_service.price,
    user_service.discount,
    (user_service.price - (user_service.price * user_service.discount / 100)) AS final_price
FROM
    service
JOIN
    user_service
ON
    service.service_id = user_service.service_id
WHERE
    service.service_is_active = 1 AND
    user_service.is_active = 1;
```

	service_name	price	discount	final_price
0	Agriculture Technology	331.40	10.32	297.199520
1	Agriculture Technology	128.02	7.31	118.661738
2	Soil Testing	454.17	0.96	449.809968
3	Agriculture Technology	164.52	0.43	163.812564
4	Agriculture Technology	306.23	13.22	265.746394
...	...	...	...	...
4920	Agriculture Technology	473.91	7.02	440.641518
4921	Agriculture Technology	444.84	7.37	412.055292
4922	Agriculture Technology	495.98	7.15	460.517430
4923	Organic Farming Advice	2291.86	3.27	2216.916178
4924	Agriculture Technology	495.79	11.35	439.517835

[4925 rows x 4 columns]

```
user_query = "Give me a breakdown of the pricing for Agriculture Technology in state California"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
INNER JOIN
    user_service ON service.service_id = user_service.service_id
INNER JOIN
    user ON user_service.user_id = user.user_id
INNER JOIN
    user_address ON user.user_id = user_address.user_id
WHERE
    service.service_name = 'Agriculture Technology'
    AND user_address.state = 'California'
    AND service.service_is_active = 1
    AND user_service.is_active = 1;
```

	service_name	price	discount	final_price
0	Agriculture Technology	345.61	1.84	339.250776
1	Agriculture Technology	401.62	15.47	339.489386
2	Agriculture Technology	140.17	14.03	120.504149
3	Agriculture Technology	329.71	2.01	323.082829

Unread Notifications Fix Through RAG Prompt

- **Automated Query Execution:** The system retrieves unread notifications for users using RAG-based SQL query generation.
- **Unread Message Summarization:** Queries count and aggregate unread messages per user, improving notification tracking.
- **User-Specific Data Retrieval:** Allows fetching unread messages for a single user or multiple users based on query input.
- **Efficient Data Organization:** Aggregated messages and timestamps provide structured insights for better notification management.

```
user_query = "Retrieve user 6087"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
 SELECT
    user.user_id,
    user.first_name,
    user.last_name,
    user_notification.message,
    user_notification.created_date
FROM
    user
JOIN
    user_notification
ON
    user.user_id = user_notification.user_id
WHERE
    user_notification.status = 'Unread' AND user.user_id = 6087;
   user_id first_name last_name  \
0     6087      Wayne      Wood

                                             message         created_date
0  Impact upon newspaper car public short bit. Ba...  2025-03-01 06:40:20
```

```
user_query = "Summarize all unread messages for user ID 3454 and provide a count of Unread messages"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
 SELECT
    user_id,
    COUNT(*) AS UnreadMessagesCount,
    GROUP_CONCAT(message, '; ') AS UnreadMessages
FROM
    user_notification
WHERE
    user_id = 3454
    AND status = 'Unread'
GROUP BY
    user_id;
   user_id  UnreadMessagesCount  \
0     3454                    3

                                     UnreadMessages
0  Boy purpose memory husband. Personal approach ...
```

```
user_query = "Retrieve 5 users"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
 SELECT u.user_id, u.first_name, u.last_name, un.message, un.created_date
FROM user u
JOIN user_notification un ON u.user_id = un.user_id
WHERE un.status = 'Unread'
ORDER BY un.created_date DESC
LIMIT 5;
   user_id first_name last_name  \
0     1588      Hannah    Carlson
1     3454       Louis      Davis
2     2130     William      Patel
3     6087       Wayne       Wood
4     4029        John      Riley

                                             message         created_date
0  Positive field less actually sometimes base. R...  2025-03-01 06:40:20
1  Figure wide drive expert. Into finally across ...  2025-03-01 06:40:20
```

Negative Feedback Fix Through RAG Prompt

```
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are a security-focused AI assistant for AgroFarmServ, responsible for generating optimized SQL queries from user qu

    ### General Guidelines:
    - Use the **provided database schema** to craft **precise SQL queries**.
    - Prioritize **JOIN operations** to fetch relevant information across multiple tables.
    - Apply **WHERE clauses** to filter records accurately based on user requests.
    - Only retrieve **non-sensitive columns** or anonymize sensitive columns using placeholder values.
    - Generate **secure SQL queries** with no direct access to sensitive information like Aadhaar numbers, PAN numbers, or
    - Ensure queries handle **NULL values** and missing data gracefully.

    ### RAG Pain Point Fixes:
    #### 1. **Negative Feedback**
    - Fetch **negative feedback** from `order_service`, `payment_details` and `user_order` tables.
    - Provide **step-by-step policy references** or service clarifications from `service_description`.
    - Join the `user`, `user_order`, `payment_details`, `order_service` and `service` tables for personalized assistance.
```

```
user_query = "Retrieve all refund requests related to poor service experiences"
db_path = "/content/Project_AgroFarmServ_grouop_1.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

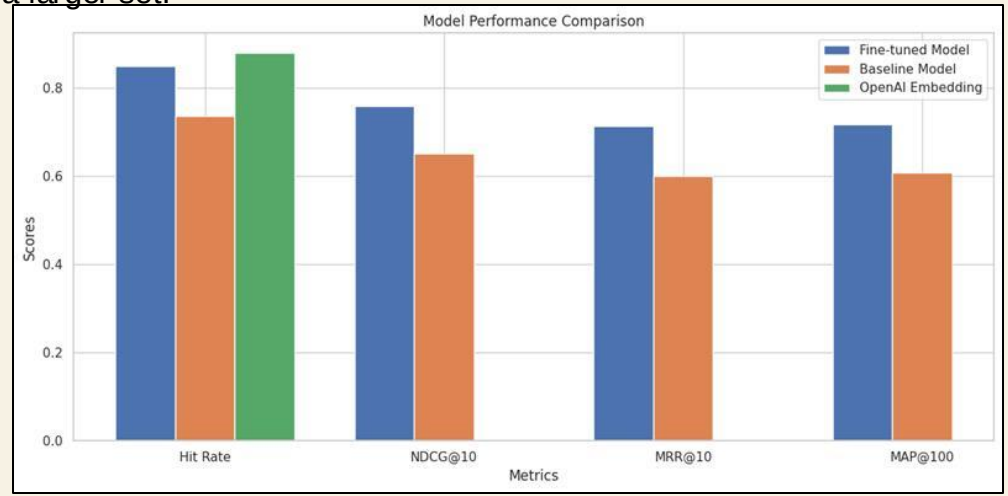
```
📌 Generated SQL Query:
 SELECT
    u.first_name,
    u.last_name,
    os.service_feedback,
    s.service_description
FROM
    user_order uo
JOIN
    user u ON uo.user_id = u.user_id
JOIN
    order_service os ON uo.user_order_id = os.user_order_id
JOIN
    user_service us ON os.user_service_id = us.user_service_id
JOIN
    service s ON us.service_id = s.service_id
WHERE
    os.service_feedback IS NOT NULL
AND
    os.service_rating < 3;
     first_name  last_name  \
0          Jorge      James
1        Gregory  Rodriguez
2         Teresa      Nelson
3            Joy       Mccoy
4          Kayla     Aguirre
...          ...        ...
6920      Ashley      Evans
6921  Christopher  Dickerson
6922         John      Patel
6923       Leslie      Lyons
6924      Matthew   Campbell

                                      service_feedback  \
0     Modern argue keep vote among society. What hea...
1     Challenge everyone sing stuff market rather ne...
2     Which act represent computer. Manage century u...
```

- **Automated Feedback Retrieval:** Queries fetch negative feedback and refund requests from multiple tables for analysis.
- **Step-by-Step Service Clarifications:** Retrieves service descriptions and policy references to address user complaints.
- **Filtering & Security:** Uses JOIN operations to consolidate feedback while protecting sensitive information.
- **Personalized Assistance:** Links user, order, and payment details to provide tailored support for resolving issues.

Model Performance Evaluation Metrics

- **Fine-Tuned Model Outperforms Baseline:** Across all metrics, the fine-tuned model (blue) achieves higher scores than the baseline model (orange), indicating improved performance.
- **OpenAI Embedding Excels in Hit Rate:** The OpenAI Embedding model (green) achieves the highest Hit Rate, suggesting better retrieval effectiveness.
- **NDCG@10 & MRR@10 Show Strong Gains:** The fine-tuned model consistently outperforms the baseline in ranking quality and relevance scoring.
- **MAP@100 Reflects Better Precision:** The fine-tuned model maintains higher Mean Average Precision, indicating better ranking of relevant results over a larger set.

```
Performance Metrics:
Fine-tuned Model Hit Rate: 0.8492
Fine-tuned Model NDCG@10: 0.7584
Fine-tuned Model MRR@10: 0.7132
Fine-tuned Model MAP@100: 0.7172
Baseline Model Hit Rate: 0.7374
Baseline Model NDCG@10: 0.6522
Baseline Model MRR@10: 0.6006
Baseline Model MAP@100: 0.6072
OpenAI Embedding Hit Rate: 0.8807
```



RAG Based Security Risk Analysis

Potential Privacy Risks

- **Exposed Personal Information:** Full names, emails, and mobile numbers are displayed without anonymization, posing a major privacy risk.
- **Direct SQL Query Execution:** Raw queries could be exploited for SQL injection if user input is not properly sanitized.
- **Unsecured Database Access**: Direct access in an online environment increases the risk of unauthorized data exposure.
- **Plaintext Storage:** Lack of encryption exposes sensitive data to breaches and misuse.

```
user_query = "list 3 users"
db_path = "/content/Project_AgroFarmServ_grouop_4.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
 SELECT * FROM user LIMIT 3;
   user_id  user_type_id first_name last_name                    org_name  \
0        1             3      Kelli      Beck                        None
1        2             3      Amber    Taylor                        None
2        3             1   Brittany    Joseph  Brady, Wiggins and Cannon

           mobile_number                  email aadhaar_number  pan_number  \
0    622.307.3599x38658      uruiz@example.com    782857990351  2040377790
1          993.757.9109  cynthia64@example.net    152185784329  1915750154
2  +1-619-931-4151x8517   diazjohn@example.com    265679955235  7397999910

  verification_status              created     created_by  \
0            Verified  2025-03-01 06:39:31  Amanda Powers
1            Verified  2025-03-01 06:39:31  Derrick Mcgee
2            Verified  2025-03-01 06:39:31     Maria Scott

              modified    modified_by  is_active
0  2025-03-01 06:39:31   Ashley Ponce          1
1  2025-03-01 06:39:31   Kevin Stokes          1
2  2025-03-01 06:39:31  Elaine Zuniga          1
```

Mitigation Strategies

- **Data Anonymization & Pseudonymization:** Mask sensitive data when displaying user information.
- **Secure SQL Queries:** Implement parameterized queries to prevent SQL injection.
- **Database Encryption:** Encrypt user details like emails and phone numbers to prevent direct access.
- **Access Control & Role-Based Permissions:** Restrict direct database queries to admin users only.

```
### Security Configuration ###
SENSITIVE_COLUMNS = {
    'user': ['mobile_number', 'email', 'aadhaar_number', 'pan_number'],
    'user_address': ['line_1', 'line_2', 'city', 'state', 'zip_code'],
    'payment_details': ['payment_method', 'transaction_date'],
    'user_notification': ['message'],
    # Add other sensitive columns as needed
}

SENSITIVE_RESULT_COLUMNS = [
    'mobile_number', 'email', 'aadhaar_number', 'pan_number',
    'line_1', 'line_2', 'city', 'state', 'zip_code',
    'payment_method', 'transaction_date', 'message'
]
```

Data Encryption

- **Secure Query Formatting:** The function format_query_prompt enforces security, performance, and readability in SQL queries.
- **Data Protection:** Sensitive fields (SENSITIVE_RESULT_COLUMNS) are masked or encrypted before storage and hidden in query outputs.
- **Masked Query Output:** Email addresses are replaced with ***** to prevent exposure.
- **Role-Based Access:** Queries filter data based on user roles, restricting access to admins (WHERE user_type = 'admin').

```
### Secure Prompt Design ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are an expert SQL assistant with in-depth knowledge of databases and query optimization.
    Given a user request and database schema information, generate the most accurate and efficient SQL query.
    Ensure that the query follows best practices, maintains security, and avoids unnecessary complexity.
    If multiple solutions exist, choose the one that best balances performance, readability, and security.

    Data Security Guidelines:
    1. **Sensitive Data Handling**:
        - Sensitive data fields include: {', '.join(SENSITIVE_RESULT_COLUMNS)}
        - For any sensitive data, **mask the data or store in plaintext**
        - Encrypt sensitive columns before storing them in the database
        - Consider masking the sensitive data when displaying partial results for security reasons
```

```
1s
user_query = "list 10 users"
db_path = "/content/AgroFarmServ.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
 SELECT user_id, first_name, last_name, '*****' as email
FROM user
WHERE user_type_id = (SELECT user_type_id FROM user_type WHERE user_type = 'admin');
     user_id first_name last_name  email
0          4     Marcus     Davis  *****
1         24      Cathy   Michael  *****
2         57       John     Carey  *****
3        110      Janet   Ramirez  *****
4        117       Mary    Taylor  *****
..       ...        ...       ...    ...
501     9864       Tony     Smith  *****
502     9866       Lisa       Day  *****
503     9887   Danielle  Mcdonald  *****
504     9899      Keith    Wilson  *****
505     9903    Russell       Ray  *****

[506 rows x 4 columns]
```

Secure Database Configuration

- **Sensitive Data Classification:** The **SENSITIVE_COLUMNS** dictionary marks personal and financial data across user, address, payment, and notification tables.
- **Filtered Data Retrieval:** The **SENSITIVE_RESULT_COLUMNS** list ensures sensitive fields are filtered before being accessed or displayed

```python
### Security Configuration ###
SENSITIVE_COLUMNS = {
    'user': ['mobile_number', 'email', 'aadhaar_number', 'pan_number'],
    'user_address': ['line_1', 'line_2', 'city', 'state', 'zip_code'],
    'payment_details': ['payment_method', 'transaction_date'],
    'user_notification': ['message'],
    # Add other sensitive columns as needed
}

SENSITIVE_RESULT_COLUMNS = [
    'mobile_number', 'email', 'aadhaar_number', 'pan_number',
    'line_1', 'line_2', 'city', 'state', 'zip_code',
    'payment_method', 'transaction_date', 'message'
]
```

```python
### Secure Prompt Design ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    ### General Guidelines:
    - Use the **provided database schema** to craft **precise SQL queries**.
    - Prioritize **JOIN operations** to fetch relevant information across multiple tables.
    - Apply **WHERE clauses** to filter records accurately based on user requests.
    - Only retrieve **non-sensitive columns** or anonymize sensitive columns using placehold
    - Generate **secure SQL queries** with no direct access to sensitive information like Aa
    - Ensure queries handle **NULL values** and missing data gracefully.

    ### Secure Database Configuration Guidelines:
    ### 1.**Secure Database Configuration:**
    - Ensure the database is configured with strong passwords and regular security updates.
    - Disable unused features to reduce the attack surface.
    - Encrypt database backups.
    - while displaying, mask the sensitive fields: {', '.join(SENSITIVE_RESULT_COLUMNS)}
```

```
    user.is_active = 1
LIMIT 5;
   payment_id  user_order_id  payment_amount payment_method payment_status  \
0           2           7625          169.81         XXXXXX        Pending
1           3           6015          635.40         XXXXXX        Pending
2          10           8045          123.41         XXXXXX        Pending
3          12           2048          471.87         XXXXXX         Failed
4          14           2137          907.34         XXXXXX        Pending

  transaction_date   first_name last_name
0       YYYY-MM-DD         Lynn  Stephens
1       YYYY-MM-DD      Anthony   Kennedy
2       YYYY-MM-DD     Courtney     Ewing
3       YYYY-MM-DD       Gloria     Jones
4       YYYY-MM-DD  Christopher     Brown
```

Data Minimization

- **Limit Data Exposure:** Retrieve only necessary columns, excluding sensitive fields (SENSITIVE_RESULT_COLUMNS) unless required.
- **Query Optimization:** Apply filtering, aggregation, and pagination to minimize unnecessary data retrieval and performance issues.
- **Avoid Unnecessary Data Retention:** Restrict timestamp queries to relevant periods and prevent excessive caching of sensitive data.
- **Optimized SQL Output:** Retrieves only essential fields (e.g., user_id, name, verification_status), excluding sensitive details unless necessary.

```
### Secure Prompt Design ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are a security-focused AI assistant for AgroFarmServ, responsible for generating optimized SQL queries from user

    ### General Guidelines:
    - Use the **provided database schema** to craft **precise SQL queries**.
    - Prioritize **JOIN operations** to fetch relevant information across multiple tables.
    - Apply **WHERE clauses** to filter records accurately based on user requests.
    - Only retrieve **non-sensitive columns** or anonymize sensitive columns using placeholder values.
    - Generate **secure SQL queries** with no direct access to sensitive information like Aadhaar numbers, PAN numbers,
    - Ensure queries handle **NULL values** and missing data gracefully.

    ### Data Minimization Guidelines:
    ### 1. **Limit Data Exposure**:
        - Always retrieve **only the necessary columns** from the database that are needed to fulfill the user's request
        - Only select the columns necessary to answer the query
        - Avoid selecting sensitive data unless absolutely required for the task and avoid columns like created by, cre
        - For any query that accesses sensitive columns like {', '.join(SENSITIVE_RESULT_COLUMNS)}, ensure they are **ex

    2. **Query Design**:
        - Apply filtering and aggregation where possible to ensure only the **relevant data** is returned, reducing unne
        - If displaying results to users, only show the **minimal required set of information**, masking or excluding nc
        - Use pagination or chunked queries when dealing with large datasets to prevent large-scale data exposure.

    3. **Avoid Unnecessary Data Retention**:
        - If data retrieval involves temporal queries, ensure that **only the relevant time period** is selected.
        - Do not store or cache unnecessary query results that may include sensitive or excessive amounts of data.
```

```
user_query = "list 10 users"
db_path = "/content/AgroFarmServ.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

1s

📌 Generated SQL Query:
SELECT user_id, user_type_id, first_name, last_name, org_name, verification_status, is_active
FROM user;

	user_id	user_type_id	first_name	last_name	org_name	verification_status	is_active
0	1	3	Kelli	Beck	None	Verified	1
1	2	3	Amber	Taylor	None	Verified	1
2	3	1	Brittany	Joseph	Brady, Wiggins and Cannon	Verified	1
3	4	3	Mary	Lee	None	Pending	1
4	5	3	Steven	Simon	None	Verified	1
...	...	...	...	...	...	...	...
9995	9996	2	Amy	Alvarez	Howard Ltd		
9996	9997	2	Robert	Snyder	Moses, Huerta and Ford		
9997	9998	2	Stephanie	Mckenzie	Turner, Reynolds and Yang		
9998	9999	3	Earl	Fernandez	None		
9999	10000	1	Sydney	Oneal	Holt, Morgan and Owens		

Admin Role - Access Control

```python
user_role = ['admin', 'user']
### Access Control ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are an expert SQL assistant with in-depth knowledge of databases and query optimization.
    Given a user request, database schema information, and the user's role, generate the most accurate and efficient SQL query.
    Ensure that the query follows best practices, maintains security, and enforces strict access control based on the user's role.

    Access Control Guidelines:
    1. **Role-Based Access**:
        - For **admin** users: Grant full access to all data, including sensitive columns such as {', '.join(SENSITIVE_RESULT_COLUMNS)}.
        - For **user** users: Limit access to non-sensitive data and exclude sensitive columns like {', '.join(SENSITIVE_RESULT_COLUMNS)} unless absolutely required. In cases wh

        Depending on the user role ({user_role}), the following logic applies:
        - **admin**: Full access to all data and columns.
        - **user**: Restrict access to sensitive columns, and if they must be included, apply **masking or obfuscation** techniques.
```

RBAC Implementation:

Admin users have full access to all data, including sensitive fields (mobile, email, **ID number** (Aadhaar), **SSN**(PAN)).

```
📌 Generated SQL Query:
 SELECT * FROM user WHERE user_id = 1;
   user_id  user_type_id first_name last_name org_name       mobile_number  \
0        1             3      Kelli      Beck     None  622.307.3599x38658

              email aadhaar_number  pan_number verification_status  \
0  uruiz@example.com    782857990351  2040377790            Verified

              created     created_by            modified   modified_by  \
0  2025-03-01 06:39:31  Amanda Powers  2025-03-01 06:39:31  Ashley Ponce

   is_active
0          1
```

Secure SQL Query

Ensures admin users can retrieve all relevant columns, including hashed sensitive details.

```
📌 Generated SQL Query:
 SELECT * FROM user;
      user_id  user_type_id first_name last_name  \
0           1             4   Lawrence    Monroe
1           2             4       Jade    Morgan
2           3             4        Amy      Davis
3           4             3     Marcus      Davis
4           5             2   Patricia     Zuniga
...       ...           ...        ...        ...
9995     9996             4      Debra      Lewis
9996     9997             1       Carl   Johnson
9997     9998             2     Steven        Lee
9998     9999             1    Heather     Prince
9999    10000             2     Robert     Nelson

                         org_name           mobile_number  \
0                  Alvarez Group       491.260.8924x5849
1                Kennedy-Alvarez          +1-338-397-8686
2                Campbell and Sons             607-859-1629
3     Fernandez, Rivera and Miller        (858)883-4244x885
4                Stewart Ltd              445-285-2123x897
...                          ...                     ...
9995             Bowman Inc          985.450.4250x61433
9996                   None  001-226-587-5776x85275
9997          Nielsen-James              2774272166
9998                   None       906.634.0920x1371
9999  Smith, Schmidt and Boone              2555589621

                          email  \
0          margaret05@example.net
1            ohunter@example.org
2          masonholly@example.net
3            wsmith@example.net
4       brianpatterson@example.net
...                          ...
9995     andersenkathy@example.net
9996       whiteclaire@example.net
```

User Role Access Control

- **Limited access** to non-sensitive columns.
- **Sensitive data** (e.g., mobile numbers, emails, financial details) is excluded unless absolutely required.
- **Masking or obfuscation** techniques are applied when needed.

```
user_role = ['admin', 'user']
### Access Control ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""
    You are an expert SQL assistant with in-depth knowledge of databases and query optimization.
    Given a user request, database schema information, and the user's role, generate the most accurate and efficient SQL query.
    Ensure that the query follows best practices, maintains security, and enforces strict access control based on the user's role.

    Access Control Guidelines:
    1. **Role-Based Access**:
        - For **admin** users: Grant full access to all data, including sensitive columns such as {', '.join(SENSITIVE_RESULT_COLUMNS)}.
        - For **user** users: Limit access to non-sensitive data and exclude sensitive columns like {', '.join(SENSITIVE_RESULT_COLUMNS)} unless absolutely required. In cases wh

        Depending on the user role ({user_role}), the following logic applies:
        - **admin**: Full access to all data and columns.
        - **user**: Restrict access to sensitive columns, and if they must be included, apply **masking or obfuscation** techniques.
```

```
📌 Generated SQL Query:
 SELECT user_id, user_type_id, first_name, last_name, org_name, verification_status, created, created_by, modified, modified_by, is_active
FROM user
WHERE user_id = 1;
   user_id  user_type_id first_name last_name       org_name  \
0        1             4   Lawrence    Monroe  Alvarez Group

  verification_status              created       created_by  \
0            Verified  2025-02-26 04:48:08  Thomas Carrillo

              modified      modified_by  is_active
0  2025-02-26 04:48:08  Richard Johnson          1
```

Anonymization & Pseudonymization

- **Secure SQL Execution:** Queries apply anonymization before displaying results to protect sensitive data.
- **Anonymization Techniques:**
- **Hashing:** Converts Aadhaar/PAN into SHA-256 hashes.
- **Masking:** Partially hides mobile numbers and emails.
- **Tokenization:** Replaces sensitive fields with unique tokens.
- **Privacy-Preserving Query Output:** Mobile numbers and emails are masked, retrieving only necessary payment details while maintaining data integrity.

```
### Secure Prompt Design ###
def format_query_prompt(user_query, schema_info):
    prompt_template = f"""

    You are an expert SQL assistant with deep knowledge of databases and query optimization.
    Given a user request and database schema information, generate the most accurate and efficient SQL query.
    Ensure that the query follows best practices for data anonymization and maintains security.

    Data Anonymization Guidelines:
    1. **General Anonymization**:
        - **Sensitive Data**: Sensitive columns in the database include: {', '.join(SENSITIVE_RESULT_COLUMNS)}.
        - For any query involving sensitive data, ensure that it is anonymized before displaying it in the results.

    2. **Anonymization Techniques**:
        - **Hashing**: Sensitive columns like `aadhaar_number`, `pan_number` should be hashed using algorithms such as SHA-256. For example, replace `aadhaar_number` with `SHA2
        - **Masking**: For columns like `email` or `mobile_number`, apply partial masking. For example, mask the `mobile_number` as `987****1234` or the `email` as `****@domain.
        - **Tokenization**: Sensitive fields can be replaced with tokenized values that cannot be traced back to the original values. For example, a `pan_number` could be repla

    3. **Query Design**:
        - Apply anonymization to sensitive data before including them in the query result.
        - Always ensure that the data returned does not contain any personal or confidential information that could be used to identify an individual.
        - Where applicable, replace sensitive columns with their anonymized versions in the final query results, making sure to maintain the integrity and structure of the quer
```

```
user_query = "give me payment details with mobile number and email"
db_path = "/content/AgroFarmServ.db"
secure_sql = generate_sql_query(user_query, db_path)
execute_sql_query(secure_sql)
```

```
📌 Generated SQL Query:
 SELECT
    payment_details.payment_id,
    payment_details.user_order_id,
    payment_details.payment_amount,
    payment_details.payment_method,
    payment_details.payment_status,
    payment_details.transaction_date,
    SUBSTR(user.mobile_number, 1, 3) || '****' || SUBSTR(user.mobile_number, -4) AS mobile_number,
    SUBSTR(user.email, 1, 4) || '****@' || SUBSTR(user.email, -10) AS email
FROM
    payment_details
JOIN
    user_order ON payment_details.user_order_id = user_order.user_order_id
JOIN
    user ON user_order.user_id = user.user_id;
    ```
      payment_id user_order_id payment_amount payment_method payment_status \
0   1   8965   178.21   Debit Card   Failed
1   3   1757   175.22   Debit Card   Refunded
2   7   167   500.96   Credit Card   Refunded
3   9   5385   387.51   Credit Card   Refunded
4   10   191   226.64 Bank Transfer   Refunded
...   .
7550   9995   1691   288.91   Debit Card   Failed
7551   9997   322   747.90   PayPal   Refunded
7552   9998   1272   121.59   Credit Card   Pending
7553   9999   2823   541.66   Credit Card   Pending
7554   10000   6813   606.84   Debit Card   Failed

     transaction_date mobile_number   email
0   2025-01-02   978****9923 shan****@xample.org
1   2025-02-24   538****6592 uols****@xample.org
2   2025-02-12   (55****x236 dian****@xample.org
3   2025-02-05   +1-****1777 rach****@xample.net
4   2025-01-02   (32****4662 brit****@xample.net
...   .
7550   2025-01-20   606****5071 lfra****@xample.net
7551   2025-01-11   (62****1620 unde****@xample.net
7552   2025-02-11   380****1633 carr****@xample.net
7553   2025-02-05   833****x534 scot****@xample.org
7554   2025-01-02   001****3001 will****@xample.org

[7555 rows x 8 columns]
```
```

SWOT Analysis

Strengths

- **Direct Connection & Fair Pricing:** Eliminates middlemen, ensuring competitive pricing and improved margins for farmers.
- Robust Data Infrastructure with real-time analytics.
- **RAG Integration:** Enhances customer support, delivering quick, context-aware responses.

Opportunities

- **Market Expansion** with additional agricultural services.
- **Collaborations** with agri-tech firms, financial institutions, and government agencies to boost reach and trust.
- **Services:** Leveraging analytics and RAG to offer tailored insights, and crop forecasting.
- **Digital Penetration:** Increasing smartphone adoption and digital literacy in rural regions bolster platform adoption.

Weaknesses

- **Tech Adoption Barriers:** Limited digital literacy among some farmers. Poor internet access could impact real-time data & service delivery.
- **Operational Complexity:** Integrated with RAG systems increases system complexity and costs with Initial reliance on stable internet and device availability.

Threats

- **Competitive Pressure:** Emerging agri-tech startups and traditional middlemen may challenge platform adoption.
- **Regulatory Risks:** Potential changes in agricultural policies or digital transaction regulations could impact operations.
- **Resistance to Change:** Established market structures and reluctance from traditional stakeholders could impede progress.

Conclusion

- **AgroFarmServ enhances trust** by connecting farmers directly with reliable service providers, eliminating delays and uncertainties.
- **Fair pricing is ensured** by removing middlemen, reducing inflated costs, and streamlining transactions.
- **Scalable service network** expands availability across regions, overcoming geographical hurdles.
- **Integrating the RAG system** enhances support efficiency and clear communication, leading to higher user satisfaction, ensuring accurate service delivery.
- **Transparent pricing structures** and secure payment systems reduce disputes and delays.
- **This solution drives** sustainable growth and digital transformation in the agricultural sector.


Thanks