**Advance AI Features**

**🧠 1. Vector Embeddings with Snowflake + Salesforce**

**✅ Architecture Highlights:**

* **Embedding Engine**:
  + External LLM (e.g. OpenAI, Cohere) called via Apex Named Credential → returns embeddings.
* **Vector Storage**:
  + Use Snowflake’s **Snowpark Container Services** + **Snowflake Cortex functions** (or native table for vector embeddings using VECTOR datatype).
  + Table: DocumentEmbeddings  
    Fields: document\_id, section, vector, metadata\_json

**🔗 Salesforce Integration:**

* On document upload to Salesforce Files or ContentVersion:
  + Trigger Apex Flow → call LLM for embedding
  + Store metadata + vector in Snowflake via **Snowflake JDBC / REST driver**
  + Query similarity via Snowflake SQL UDF + expose top matches in Lightning Component

**🤖 2. Agentic Document Reasoning (Salesforce + Snowflake)**

**✅ Architecture Highlights:**

* **Agents as Snowflake Services**:
  + Deploy reasoning agents in **Snowpark Python** or **Snowpark Container Runtime**
  + Each agent can fetch clauses, infer purpose, recommend tone, and flag gaps

**🔗 Salesforce Integration:**

* Create Apex interface to invoke agent services hosted in Snowflake
* Store reasoning outputs in custom object: ClauseInsight\_\_c  
  Fields: Document\_\_c, Purpose\_\_c, RiskLevel\_\_c, SuggestedTone\_\_c
* Use Salesforce Flow to orchestrate agent calls and populate clause recommendations for legal review.

**🕸️ 3. Graph View of Document Relationships (Salesforce + Snowflake)**

**✅ Architecture Highlights:**

* Store relationships in Snowflake: DocumentGraph  
  Nodes: Document, Clause, Entity  
  Edges: References, Amends, Extends, etc.
* Generate adjacency matrix + visual context using Snowflake SQL Graph functions or JSON hierarchical views.

**🔗 Salesforce Integration:**

* Query graph data from Snowflake → render graph in a **Lightning Web Component** using D3.js
* Support click-through to related ContentVersion, clause, or version history

**📥 4. Multimodal Ingestion via Kasetti IDP + Snowflake + Salesforce**

**✅ Architecture Highlights:**

* **Input Types**: PDF, image, scanned docs, audio, video
* **Kasetti’s IDP Solution**: Handles OCR, layout parsing, transcription
* Parsed outputs pushed to Snowflake tables: ParsedContent, ClauseBlocks, SourceModality

**🔗 Salesforce Integration:**

* Document upload triggers Apex → POST to IDP endpoint with Salesforce File
* Receive parsed text + structure → store in Snowflake
* Ingested clauses pulled into Salesforce via external object or scheduled batch
* Display clause previews, tone assignment, context mapping inside **ClauseComposer Lightning Tab**

**🧩 Admin Workflow in Salesforce**

* **Cue Editor & Preview Dashboard** powered by live data from Snowflake
* **Model Calls** routed from Salesforce Apex to external LLMs
* **Compliance Flagging** based on cue-context match or agent judgment stored in Snowflake

**📦 Deployment Model**

* Salesforce Managed Package:
  + Custom Objects: Clause\_\_c, Cue\_\_c, Insight\_\_c
  + Apex classes for integration
  + Flows for ingestion + preview
* Snowflake Environment:
  + Cortex + Snowpark containers
  + Vector DB + document metadata
  + REST endpoints exposed for Salesforce

**🧩 Salesforce Admin Console Page – Snowflake Feature Hub**

**🎛️ Component: SnowflakeFeatureControlPanel (LWC Page Layout)**

+----------------------------------------------------------+

| ⚙️ Snowflake-Enabled Intelligence Features |

+----------------------------------------------------------+

[🔄 Scroll Toggles to Enable Features – Disabled until subscribed]

Feature | Status | Usage Link

----------------------------------------------------------

▢ Vector Semantic Search | 🔒 Off | View Samples

▢ Agentic Clause Reasoning | 🔒 Off | Learn More

▢ Document Graph Navigator | 🔒 Off | Launch Preview

▢ Multimodal Upload + Parsing | 🔒 Off | Supported Formats

⚠️ Subscription Required to Activate Features

[ 📝 Manage Subscription ] → Opens `SubscriptionConfigFlow`

**🛒 Subscription Module – SubscriptionManager Architecture**

**📦 Custom Objects:**

| **Object Name** | **Purpose** |
| --- | --- |
| Subscription\_\_c | Stores user's active plans |
| Feature\_\_c | Lists available Snowflake-backed tools |
| UsageLog\_\_c | Tracks invocation & utilization |
| SnowflakeTenant\_\_c | Maps Salesforce org to Snowflake env |

**🔧 Apex Flow: SubscriptionConfigFlow**

* Step 1: Choose Plan (e.g., Basic, Enhanced, Enterprise)
* Step 2: Enter Snowflake Account ID / Workspace
* Step 3: Validate Connectivity via REST call (Named Credential)
* Step 4: Show Available Features → Toggle Activation

**💳 Billing Integration (Optional):**

* Map utilization (e.g., storage, rows queried, models invoked) from Snowflake’s consumption metadata
* Use REST callout to Snowflake's usage APIs
* Show estimated bill or send to Salesforce Billing / Revenue Cloud

**🔄 Feature Activation Logic**

* Each Scroll Toggle in UI binds to Feature\_\_c.Status
* When clicked:
  + Check Subscription\_\_c for active entitlement
  + If enabled:
    - Set Feature\_\_c.Status = Active
    - Log activation in UsageLog\_\_c
    - Call Snowflake REST API (if required) to provision resources

**🔌 Snowflake Integration Highlights**

* Use Apex Named Credentials to invoke:
  + Vector queries (QUERY VECTOR ...)
  + Document graph (GRAPH VIEW ...)
  + Agentic services (via Snowpark Container API endpoint)
  + Parsed content from Kasetti IDP module
* Async results passed back to Salesforce → Display in ClauseComposerPreviewHub

**🔒 Security & Compliance Notes**

* Use Custom Metadata for API Keys or Secrets
* Ensure Snowflake calls scoped per org via SnowflakeTenant\_\_c
* Log every feature toggle & clause injection in UsageLog\_\_c with timestamp and source

Absolutely, Vinod! Let’s build this out into two parts:

1. 🎟️ A **Dynamic License Entitlement Flow** for org-tiered access to Snowflake-powered features
2. 🛍️ A **Salesforce AppExchange Listing Mockup** that showcases these features in a compelling, SaaS-ready way

**🎟️ Part 1: Dynamic License Entitlement Flow – SnowflakeFeatureEntitlementManager**

**🧭 Org Tier Strategy**

| **Org Tier** | **Access Level** | **Notes** |
| --- | --- | --- |
| Startup | ✘ Preview only | No activation, sample previews only |
| Professional | ✔ Core features | Vector search + basic ingestion |
| Enterprise | ✔ All modules + audit logs | Includes agents, graph view, multimodal |
| Custom Tier | 💬 Admin-defined entitlements | Flagged for manual review/license team |

**🔧 Key Objects**

| **Object Name** | **Description** |
| --- | --- |
| OrgTier\_\_c | Maps Salesforce Org ID to license tier |
| FeatureEntitlement\_\_c | Feature toggles per tier with expiration & flags |
| UsageMeter\_\_c | Real-time tracking (calls, embeddings, bytes used) |
| ActivationAudit\_\_c | Logs all activations with who, when, and scope |

**🛠️ Flow: ActivateSnowflakeFeatureFlow**

1. **Check Tier** → Auto-assess via OrgTier\_\_c
2. **List Eligible Features** → UI toggles filtered by tier
3. **Usage Preview** → Show embeddings consumed, query volume
4. **Activate Feature** → Update FeatureEntitlement\_\_c and log to ActivationAudit\_\_c
5. **Notify Billing** → Push consumption stats if configured to integrate with Salesforce Billing or your internal entitlement system

Supports renewal flows, usage throttling, and prompts for upgrade.

**🛍️ Part 2: Salesforce AppExchange Listing Page – Mock UI**

**🎨 Listing Title:**

**ClauseComposer AI – Snowflake-Powered Legal Intelligence Platform**

**📦 Package Summary:**

Empower your legal workflows with clause-level intelligence, semantic search, and multimodal ingestion. This Salesforce-native solution integrates tightly with Snowflake for advanced document parsing, agentic clause reasoning, and compliance graph views. Perfect for AppExchange-savvy enterprises scaling their contract frameworks.

**🖼️ Screenshots Carousel (Mock Cards)**

| **Screenshot Title** | **Description** |
| --- | --- |
| "Feature Hub Toggles" | Shows admin UI with Snowflake features, toggle switches & plan status |
| "Clause Preview Dash" | Real-time clause simulation from Snowflake embeddings |
| "Graph View of Docs" | Interactive D3-based document relationship viewer |
| "Multimodal Ingestion" | Upload flow with support for PDFs, images, and scans |
| "License Management UI" | Flow with plan selection, limits, and activation history |

**📊 Feature Matrix:**

| **Feature** | **Starter** | **Professional** | **Enterprise** |
| --- | --- | --- | --- |
| Clause Preview Dashboard | ✅ | ✅ | ✅ |
| Semantic Vector Search | 🚫 | ✅ | ✅ |
| Agentic Clause Reasoning | 🚫 | 🚫 | ✅ |
| Document Graph View | 🚫 | ✅ | ✅ |
| Multimodal Ingestion | 🚫 | ✅ | ✅ |
| Audit Logs & Reviewer Flows | 🚫 | 🚫 | ✅ |

**🛒 Call to Action:**

🌐 Visit the ClauseComposer Portal to configure Snowflake linkages, preview features, and start a free trial.  
📞 Enterprise clients can schedule onboarding with a dedicated architect.

Yes Vinod, your solution is absolutely viable—and with the right architecture, it can scale beautifully. You’re essentially creating a **multi-tenant usage billing dashboard** layered over Snowflake, with **Streamlit as the front-end**, and Salesforce administering entitlements. Let’s map it cleanly:

**✅ Solution Architecture Overview**

**🧩 Key Components**

| **Layer** | **Tool/Platform** | **Role** |
| --- | --- | --- |
| Front-End | Streamlit in Snowflake | Org-specific admin dashboard with usage & billing |
| Back-End | Snowflake (Warehouses + Views) | Usage tracking, metering, pricing logic |
| Billing Logic | Snowflake SQL / Python UDFs | Apply % markup, calculate usage per org |
| Org Identity | Salesforce (via SnowflakeTenant\_\_c) | Maps subscription holders to org IDs |
| Activation Flow | Salesforce Flow & Metadata | Controls feature access and tier limits |
| Reporting | Streamlit + Snowflake Dashboards | View consumption, feature toggles, billing history |

**🔧 Data Architecture in Snowflake**

**Tables Required:**

* usage\_events: logs per-org warehouse activity, LLM usage, storage
* org\_profile: org ID, subscription plan, contact info
* feature\_activation: tracks which org has activated which modules
* billing\_snapshot: stores monthly markup summary and usage
* rate\_card: configurable % markup per feature, tier, org type

**Pricing Logic:**

SELECT

org\_id,

SUM(raw\_credits\_used \* markup\_percent / 100) AS billable\_amount

FROM

usage\_events

JOIN

rate\_card USING (feature)

GROUP BY org\_id;

**🎨 Streamlit App Modules**

**🔐 Admin Login**

* Auth via Snowflake roles or SSO
* Dashboard auto-filters to org context

**📊 Usage Overview**

* Warehouse/LLM activity per day
* Multimodal ingestion stats
* Cost summary with markup tier breakdown

**🧮 Billing Page**

* Monthly snapshot with export option
* Visual markup preview (chart: raw vs marked cost)
* Dispute or contact flow (e.g., via embedded Salesforce form)

**⚙️ Feature Toggle Manager**

* Activate features based on org tier
* Greyed-out toggles if not subscribed
* Live status sync with Salesforce via REST or webhooks

**🔄 Sync with Salesforce**

* Push activation info to FeatureEntitlement\_\_c
* Pull org ID from SnowflakeTenant\_\_c
* Optionally store monthly billing in OrgInvoice\_\_c

**✨ Bonus Enhancements**

* Alerts in Streamlit for usage overages or quota breaches
* Prepayment credit system or feature bundling
* Forecast panel showing estimated next bill
* Admin panel in Salesforce mirrors Streamlit info with click-through

**🔔 Alerts in Streamlit for Usage Overages or Quota Breaches**

**🧠 Solution Logic**

* Define usage caps for features based on org tier
* Monitor usage\_events in real time or batch
* Trigger alerts when thresholds are crossed

**🔧 Schema**

TABLE org\_quota {

org\_id STRING,

feature STRING,

daily\_limit INT,

monthly\_limit INT,

alert\_email STRING

}

TABLE usage\_events {

org\_id STRING,

feature STRING,

usage\_amount INT,

timestamp DATETIME

}

**🖥️ Streamlit UI**

* **Usage Dashboard** with:
  + 🔴 Red banners when usage exceeds quota
  + 🟡 Yellow warnings at 80% utilization
  + 📩 “Send Alert” toggle per feature per org
  + Option to configure alert destinations

**🔁 Backend Logic**

# Pseudocode

if daily\_usage > daily\_limit:

send\_alert("Daily limit breached")

**💳 Prepayment Credit System & Feature Bundling**

**🧠 Solution Logic**

* Org prepays credits
* Deduct credit balance as features are used
* Bundled features give discounted rates

**🔧 Schema**

TABLE org\_credits {

org\_id STRING,

credit\_balance DECIMAL(12,2),

last\_recharge DATE

}

TABLE feature\_bundle {

bundle\_id STRING,

features ARRAY,

discount\_percent DECIMAL(5,2)

}

**🖥️ Streamlit UI**

* **Wallet Panel**
  + 💰 Credit balance
  + 📆 Recharge history
  + ⚙️ Bundle selector (with rate preview)
  + 📉 Graph: usage vs credits left

**🔁 Backend Logic**

# Usage deduction

credit\_balance -= feature\_usage \* (1 - discount\_percent / 100)

**📈 Forecast Panel for Estimated Next Bill**

**🧠 Solution Logic**

* Use last 30/60/90 days usage trend
* Predict future usage with simple linear regression or Snowflake ML

**🔧 Schema**

TABLE forecast\_snapshot {

org\_id STRING,

feature STRING,

forecast\_date DATE,

estimated\_cost DECIMAL(12,2)

}

**🖥️ Streamlit UI**

* **Forecast Page**
  + 📊 Line chart: past vs forecasted usage
  + 📅 Dropdown to select forecast window
  + 🔍 Option to switch between feature tiers
  + 🧮 Exportable forecast summary

**🔁 Backend Logic**

# Sample forecasting model (simplified)

predicted\_usage = slope \* days\_ahead + intercept

estimated\_cost = predicted\_usage \* markup

**🛠️ Salesforce Admin Panel Mirroring Streamlit**

**🧠 Solution Logic**

* Push usage & billing data from Snowflake to Salesforce via REST API
* Visual components mirror Streamlit panels
* Use AdminView\_\_c, OrgBilling\_\_c, and FeatureStatus\_\_c objects

**🔧 Schema Mapping**

| **Snowflake Table** | **Salesforce Object** |
| --- | --- |
| org\_profile | Account |
| billing\_snapshot | OrgBilling\_\_c |
| feature\_activation | FeatureStatus\_\_c |
| usage\_events | Custom Dashboard UI |

**🖥️ Salesforce UI Elements**

* **Lightning Component or LWC**:
  + 💡 Info cards per org: usage, status
  + 📄 Invoice viewer with export
  + 🔘 Toggle features with real-time sync
  + 🔗 “Open Streamlit Dashboard” (URL redirect)

**🔁 Backend Sync**

# Push updates from Snowflake to Salesforce

PUT /services/data/vXX.X/sobjects/OrgBilling\_\_c/{id}

**🧪 Sample Schema Overview**

TABLE org\_profile (

org\_id STRING PRIMARY KEY,

name STRING,

plan STRING,

start\_date DATE,

snowflake\_tenant STRING

)

TABLE billing\_snapshot (

org\_id STRING,

month DATE,

raw\_usage DECIMAL,

markup\_percent INT,

final\_bill DECIMAL

)

TABLE feature\_activation (

org\_id STRING,

feature STRING,

is\_active BOOLEAN,

activated\_on DATE

)

**🧿 Mock UI Structure (Streamlit Sketch)**

Main Sidebar:

- Org Selector

- Plan Info

- Credit Wallet (live badge)

Main Panel Tabs:

1. Usage & Overages 📊

- Line chart + alerts

2. Billing & Forecasts 📈

- Summary + predictive chart

3. Feature Manager ⚙️

- Toggles + Bundle Picker

4. Settings 💼

- Email alerts, quota setup

a ready-to-execute starter kit broken into two core parts: **Snowflake SQL Scripts** for data foundation and **Streamlit layout code** for UI components. You’ll be able to plug and play, then adapt based on the scale of your org data and feature set.

**🧠 Part 1: Snowflake SQL Scripts**

**1. 🎯 Org Profile Setup**

CREATE TABLE org\_profile (

org\_id STRING PRIMARY KEY,

org\_name STRING,

plan STRING,

start\_date DATE,

snowflake\_tenant STRING

);

**2. 📦 Usage Logging**

CREATE TABLE usage\_events (

usage\_id STRING,

org\_id STRING,

feature STRING,

usage\_amount DECIMAL(10,2),

usage\_unit STRING,

timestamp TIMESTAMP

);

**3. 💳 Credit System**

CREATE TABLE org\_credits (

org\_id STRING PRIMARY KEY,

credit\_balance DECIMAL(12,2),

last\_recharge DATE

);

**4. 📊 Billing Snapshot + Markup**

CREATE TABLE billing\_snapshot (

org\_id STRING,

billing\_month DATE,

raw\_usage DECIMAL(10,2),

markup\_percent DECIMAL(5,2),

final\_bill DECIMAL(12,2)

);

**5. 🔔 Quota & Alerts**

CREATE TABLE org\_quota (

org\_id STRING,

feature STRING,

daily\_limit DECIMAL(10,2),

monthly\_limit DECIMAL(10,2),

alert\_email STRING

);

**6. ⏩ View for Billing Page**

CREATE OR REPLACE VIEW v\_billing\_summary AS

SELECT

org\_id,

billing\_month,

raw\_usage,

markup\_percent,

ROUND(raw\_usage \* (1 + markup\_percent / 100), 2) AS final\_bill

FROM billing\_snapshot;

**🖥️ Part 2: Streamlit Layout Code**

import streamlit as st

import pandas as pd

from snowflake.connector import connect

# Sidebar: Org Selector

st.sidebar.title("🔍 Org Context")

selected\_org = st.sidebar.selectbox("Choose Organization", ["Org A", "Org B", "Org C"])

# Main Page Title

st.title("🧮 Usage & Billing Dashboard")

# Tabs

tab1, tab2, tab3 = st.tabs(["📊 Usage", "💳 Billing", "⚙️ Feature Manager"])

with tab1:

st.subheader("🚦 Usage Metrics")

st.metric(label="Today’s Usage", value="123 Units")

st.metric(label="Monthly Total", value="4567 Units")

st.warning("⚠️ Org is approaching quota!")

with tab2:

st.subheader("📄 Billing Summary")

st.write("Below is the estimated billing based on current markup rates.")

df\_billing = pd.DataFrame({

"Month": ["July", "August"],

"Raw Usage ($)": [120.00, 150.50],

"Markup (%)": [20, 20],

"Final Bill ($)": [144.00, 180.60]

})

st.dataframe(df\_billing)

st.line\_chart(df\_billing["Final Bill ($)"])

with tab3:

st.subheader("🔘 Feature Activation")

st.checkbox("Enable AI Templates")

st.checkbox("Enable Forecast Panel")

st.checkbox("Enable Credit Monitoring")

st.button("Save Feature Changes")

# Footer

st.info("Streamlit App powered by Snowflake usage data")

You're building something exceptional, Vinod—so let’s wrap this blueprint into a **modular architecture**, **tier-aware billing logic**, and a **Salesforce sync strategy** that supports scale, compliance, and clarity across the board 💼⚙️

**🧱 End-to-End Architecture Flow**

graph TD

A[User/Org] --> B[Streamlit Front-End]

B --> C1[Usage Tracker]

B --> C2[Billing & Credits Dashboard]

B --> C3[Forecast Panel]

B --> C4[Feature Toggles]

B --> D1[Alert Manager]

C1 --> E1[Snowflake Table: usage\_events]

C2 --> E2[Snowflake Table: billing\_snapshot]

C2 --> E3[Snowflake Table: org\_credits]

C3 --> E4[Snowflake View: forecast\_snapshot]

D1 --> E5[Snowflake Table: org\_quota]

C4 --> E6[Snowflake Table: feature\_activation]

E\* --> F[Snowflake View: v\_billing\_summary]

F --> G[Salesforce Objects]

G --> H[Admin Panel in Salesforce]

Data flows from user activity → processed in Snowflake → visualized in Streamlit → synced to Salesforce for governance and audits.

**📊 Billing Logic with Multi-Tier Scaling**

**🎚️ Tier Definitions**

TABLE tier\_plan (

plan STRING PRIMARY KEY,

markup\_percent DECIMAL(5,2),

quota\_daily INT,

quota\_monthly INT

)

| **Plan** | **Markup %** | **Daily Quota** | **Monthly Quota** |
| --- | --- | --- | --- |
| Basic | 10% | 100 units | 2000 units |
| Pro | 15% | 500 units | 10000 units |
| Premium | 20% | Unlimited | Unlimited |

**⚙️ Dynamic Logic**

SELECT

e.org\_id,

p.plan,

SUM(e.usage\_amount) AS total\_usage,

p.markup\_percent,

ROUND(SUM(e.usage\_amount) \* (1 + p.markup\_percent / 100), 2) AS final\_bill

FROM usage\_events e

JOIN org\_profile o ON e.org\_id = o.org\_id

JOIN tier\_plan p ON o.plan = p.plan

GROUP BY e.org\_id, p.plan, p.markup\_percent;

**🎨 Visual Flow in Streamlit**

* Auto-select markup % based on plan
* Warning bar when nearing plan quotas
* Option to upgrade plan or recharge credits

**🔄 Salesforce Sync Strategy**

**🧬 Object Mapping**

| **Snowflake Table** | **Salesforce Object** | **Use Case** |
| --- | --- | --- |
| org\_profile | Account | Org master record |
| billing\_snapshot | OrgBilling\_\_c | Monthly usage & billing info |
| feature\_activation | FeatureStatus\_\_c | Toggles per org |
| forecast\_snapshot | Forecast\_\_c | Estimated usage projections |

**🔁 API Calls**

**1. Push Billing Info**

POST /services/data/vXX.X/sobjects/OrgBilling\_\_c

{

"Org\_\_c": "001XYZ123",

"Billing\_Month\_\_c": "2025-08-01",

"Final\_Amount\_\_c": 180.60

}

**2. Update Feature Status**

PATCH /services/data/vXX.X/sobjects/FeatureStatus\_\_c/{id}

{

"Is\_Active\_\_c": true

}

**3. Sync Forecasts**

PUT /services/data/vXX.X/sobjects/Forecast\_\_c/{id}

{

"Estimated\_Cost\_\_c": 210.80,

"Forecast\_Date\_\_c": "2025-09-01"

}

**🖥️ Admin Panel in Salesforce**

* Built with **Lightning Components** or **LWC**
* 🔁 Mirrors Streamlit dashboards
* 🔗 “Open Usage Dashboard” button (redirects to Streamlit)
* ⏱️ Audit logs for billing, features, alerts

**💡 Final Notes**

This bundled framework gives you:

* Modular deployment across any org size or tier ✅
* Billing logic that adapts seamlessly to usage and plan level 📈
* Streamlit as a dynamic interface, and Salesforce as your audit & control layer 🔐

! Here’s your build-ready **technical documentation** for the usage billing Streamlit app in Snowflake, complete with:

* ✅ Flow diagrams
* 🧬 Schema annotations
* 🚀 Deployment tips

This will help you confidently present to stakeholders, onboard engineering teams, or refine for investor-readiness.

**🧭 System Overview**

This architecture tracks Snowflake usage, applies markup-based billing, visualizes consumption in Streamlit, and syncs metadata to Salesforce for governance.

**🔁 End-to-End Architecture Flow**

graph TD

U[Org User] -->|Uses Features| SE[Snowflake Event Logger]

SE -->|Logs usage| T1[usage\_events]

U -->|Views Dashboard| ST[Streamlit UI]

ST -->|Displays| T2[billing\_snapshot] & T3[org\_credits]

ST -->|Manages| T4[feature\_activation] & T5[org\_quota]

T2 -->|Summarized via| V1[v\_billing\_summary]

V1 --> SF[Salesforce Sync]

SF -->|Mirrors| A1[OrgBilling\_\_c], A2[FeatureStatus\_\_c], A3[Forecast\_\_c]

**📐 Database Schema Annotations**

**🔶 org\_profile**

Stores basic org metadata

CREATE TABLE org\_profile (

org\_id STRING PRIMARY KEY,

org\_name STRING,

plan STRING, -- Basic, Pro, Premium

start\_date DATE,

snowflake\_tenant STRING -- Optional external reference

);

**🔶 usage\_events**

Tracks feature-level usage

CREATE TABLE usage\_events (

usage\_id STRING PRIMARY KEY,

org\_id STRING,

feature STRING,

usage\_amount DECIMAL(10,2),

usage\_unit STRING,

timestamp TIMESTAMP

);

**🔶 billing\_snapshot**

Monthly billing summary with markup

CREATE TABLE billing\_snapshot (

org\_id STRING,

billing\_month DATE,

raw\_usage DECIMAL(10,2),

markup\_percent DECIMAL(5,2),

final\_bill DECIMAL(12,2)

);

**🔶 org\_credits**

Tracks prepayment credits per org

CREATE TABLE org\_credits (

org\_id STRING PRIMARY KEY,

credit\_balance DECIMAL(12,2),

last\_recharge DATE

);

**🔶 feature\_activation**

Controls which orgs have access to which features

CREATE TABLE feature\_activation (

org\_id STRING,

feature STRING,

is\_active BOOLEAN,

activated\_on DATE

);

**🔶 org\_quota**

Defines usage caps and alert logic

CREATE TABLE org\_quota (

org\_id STRING,

feature STRING,

daily\_limit DECIMAL(10,2),

monthly\_limit DECIMAL(10,2),

alert\_email STRING

);

**⚙️ Deployment Tips**

**1. Snowflake Setup**

* Use **role-based access control** with one role per org
* Store usage logs with **Snowpipe** or warehouse history queries
* Create secure views (v\_billing\_summary, v\_quota\_usage) exposed to Streamlit

**2. Streamlit in Snowflake**

* Host Streamlit in **Snowflake Native App** (for AppExchange scale)
* Use st.secrets for connection string to Snowflake
* Define modular components:
  + UsagePanel()
  + BillingPanel()
  + CreditWallet()
  + FeatureManager()

**3. Markup Logic**

Use tier logic from org\_profile.plan:

SELECT

e.org\_id,

o.plan,

t.markup\_percent,

SUM(e.usage\_amount) AS raw,

ROUND(SUM(e.usage\_amount \* (1 + t.markup\_percent / 100)), 2) AS final\_bill

FROM usage\_events e

JOIN org\_profile o USING (org\_id)

JOIN tier\_plan t ON o.plan = t.plan

GROUP BY e.org\_id, o.plan, t.markup\_percent;

**4. Salesforce Sync**

* Use **Snowflake External Functions** or middleware (e.g., Azure Function, Zapier)
* Push to objects:
  + OrgBilling\_\_c monthly
  + FeatureStatus\_\_c with toggles
  + Forecast\_\_c for estimates

Sample API Call:

POST /services/data/vXX.X/sobjects/OrgBilling\_\_c

{

"Org\_\_c": "001XYZ123",

"Final\_Amount\_\_c": 240.50,

"Billing\_Month\_\_c": "2025-08"

}

**📊 Optional Add-ons**

| **Module** | **Purpose** | **Table/View** |
| --- | --- | --- |
| Credit Forecasting | Predict next bill | forecast\_snapshot |
| Bundle Optimizer | Group features for discounts | feature\_bundle |
| Quota Breach Alerts | Email alerts via Streamlit triggers | org\_quota |

Absolutely, Vinod—let’s architect these add-ons so they plug seamlessly into your Snowflake + Streamlit + Salesforce stack. They’ll expand your app’s intelligence, empower client decisions, and enhance billing transparency.

**🔮 1. Credit Forecasting System**

**🧠 Purpose**

Helps orgs predict credit burn rate based on historical usage patterns and forecasted workloads.

**🧬 Schema Design**

CREATE TABLE forecast\_snapshot (

org\_id STRING,

forecast\_date DATE,

feature STRING,

projected\_usage DECIMAL(10,2),

projected\_cost DECIMAL(12,2),

confidence\_score DECIMAL(5,2)

);

**⚙️ Forecast Logic**

* Use **rolling average**, **trend extrapolation**, or **Snowflake ML-powered regression**
* Confidence score reflects model certainty (e.g., based on variance)

-- Simple rolling forecast

SELECT

org\_id,

feature,

AVG(usage\_amount) \* forecast\_days AS projected\_usage,

ROUND(projected\_usage \* markup\_rate, 2) AS projected\_cost

FROM usage\_events

WHERE timestamp >= CURRENT\_DATE - INTERVAL '30' DAY

GROUP BY org\_id, feature;

**🖼️ UI**

* Streamlit 📈 Forecast Panel tab
* Line chart: Past vs. projected usage
* Confidence meter
* “What-if” input box (custom workload projections)

**📦 2. Feature Bundle Optimizer**

**🧠 Purpose**

Helps orgs select bundled features to save on usage costs and activate capabilities more strategically.

**🧬 Schema Design**

CREATE TABLE feature\_bundle (

bundle\_id STRING PRIMARY KEY,

bundle\_name STRING,

features ARRAY,

discount\_percent DECIMAL(5,2),

eligibility\_plan STRING

);

CREATE TABLE org\_bundle\_subscription (

org\_id STRING,

bundle\_id STRING,

activated\_on DATE

);

**⚙️ Discount Logic**

* If org activates full feature set from a bundle, apply discount in billing view
* Eligibility constrained by plan tier (Basic, Pro, Premium)

-- Apply discount

SELECT

org\_id,

feature,

usage\_amount,

CASE

WHEN feature IN (SELECT VALUE FROM FLATTEN(INPUT => b.features)) THEN usage\_amount \* (1 - b.discount\_percent / 100)

ELSE usage\_amount

END AS effective\_usage

FROM usage\_events

JOIN org\_bundle\_subscription USING (org\_id)

JOIN feature\_bundle b USING (bundle\_id);

**🖼️ UI**

* Streamlit ⚙️ Feature Manager tab
* Bundle selector + preview cost savings
* Activate/deactivate button
* Tooltip showing eligibility based on current plan

**🚨 3. Quota Breach Alert System**

**🧠 Purpose**

Detects usage spikes or nearing limits, notifies admins and users with tailored messaging.

**🧬 Schema Design**

CREATE TABLE org\_quota (

org\_id STRING,

feature STRING,

daily\_limit DECIMAL(10,2),

monthly\_limit DECIMAL(10,2),

alert\_email STRING,

alert\_threshold\_percent INT DEFAULT 80

);

CREATE TABLE quota\_alert\_log (

org\_id STRING,

feature STRING,

breach\_level STRING, -- 'Warning', 'Critical'

detected\_on TIMESTAMP,

alert\_sent BOOLEAN

);

**⚙️ Alert Logic**

-- Detect breaches

SELECT

q.org\_id,

q.feature,

SUM(e.usage\_amount) AS usage\_today,

CASE

WHEN usage\_today > q.daily\_limit THEN 'Critical'

WHEN usage\_today > q.daily\_limit \* q.alert\_threshold\_percent / 100 THEN 'Warning'

END AS breach\_level

FROM usage\_events e

JOIN org\_quota q USING (org\_id, feature)

WHERE CAST(e.timestamp AS DATE) = CURRENT\_DATE

GROUP BY q.org\_id, q.feature, q.daily\_limit;

**📧 Email Notifications**

* Use external functions or integration with Azure Function/Lambda
* Tailored messages:
  + 🔴 “Quota Breach Alert: Feature X has exceeded limits”
  + 🟡 “Usage Warning: You’ve reached 80% of your quota”

**🖼️ UI**

* Streamlit sidebar badge ⚠️
* Quota Monitor tab:
  + Daily & monthly progress bars
  + Alert logs
  + Configurable threshold sliders

**🤖 Integration Tip**

Each module stores outputs and summaries in forecast\_snapshot, org\_bundle\_subscription, and quota\_alert\_log, which can be exposed via:

* Streamlit views and toggles
* Salesforce sync:
  + Forecast\_\_c → projected usage
  + FeatureBundle\_\_c → bundle tracking
  + QuotaAlert\_\_c → incident log

Absolutely, Vinod! Here's a production-ready example of how to implement **quota breach alert triggers using Snowflake Tasks**. This setup scans daily usage, compares it with quota limits, and logs alerts if thresholds are breached.

**🔧 Step-by-Step SQL Architecture**

**🗃️ 1. Define Required Tables**

These should already exist based on your previous setup:

CREATE TABLE usage\_events (

org\_id STRING,

feature STRING,

usage\_amount DECIMAL(10,2),

timestamp TIMESTAMP

);

CREATE TABLE org\_quota (

org\_id STRING,

feature STRING,

daily\_limit DECIMAL(10,2),

alert\_threshold\_percent INT DEFAULT 80

);

CREATE TABLE quota\_alert\_log (

org\_id STRING,

feature STRING,

breach\_level STRING,

detected\_on TIMESTAMP,

alert\_sent BOOLEAN

);

**🧠 2. Create Breach Detection Stored Procedure**

CREATE OR REPLACE PROCEDURE detect\_quota\_breach()

RETURNS STRING

LANGUAGE SQL

AS

$$

INSERT INTO quota\_alert\_log

SELECT

q.org\_id,

q.feature,

CASE

WHEN SUM(e.usage\_amount) > q.daily\_limit THEN 'Critical'

WHEN SUM(e.usage\_amount) > q.daily\_limit \* q.alert\_threshold\_percent / 100 THEN 'Warning'

ELSE NULL

END AS breach\_level,

CURRENT\_TIMESTAMP,

FALSE

FROM org\_quota q

JOIN usage\_events e

ON q.org\_id = e.org\_id AND q.feature = e.feature

WHERE CAST(e.timestamp AS DATE) = CURRENT\_DATE

GROUP BY q.org\_id, q.feature, q.daily\_limit, q.alert\_threshold\_percent

HAVING breach\_level IS NOT NULL;

$$;

**⏰ 3. Schedule Task to Run Daily**

CREATE OR REPLACE TASK task\_detect\_quota\_breach

WAREHOUSE = your\_warehouse\_name

SCHEDULE = 'USING CRON 0 3 \* \* \* UTC' -- Runs daily at 3 AM UTC

AS

CALL detect\_quota\_breach();

💡 Tip: You can add email alerts by connecting to external functions that ping admin emails when new critical alerts are inserted.

**🧪 Bonus: Preview Trigger Results Before Production**

CALL detect\_quota\_breach();

SELECT \* FROM quota\_alert\_log ORDER BY detected\_on DESC;

Brilliant! Here’s how we can **extend the quota breach alert system** with live user awareness in Streamlit and optional **Snowflake notifications** for admin teams 🛠️📣

**🚦 Streamlit Banner Integration**

**🖼️ UI Logic**

* When an alert is logged in quota\_alert\_log, Streamlit displays a contextual banner.
* Banners adapt based on breach level:
  + 🟡 Warning → “You’ve reached 80% of your daily limit for Feature X.”
  + 🔴 Critical → “Quota breached! Feature X usage exceeded daily threshold.”

**🧱 Implementation Sample**

import streamlit as st

import snowflake.connector

import pandas as pd

# Connect and fetch latest alerts

query = """

SELECT org\_id, feature, breach\_level, detected\_on

FROM quota\_alert\_log

WHERE detected\_on::DATE = CURRENT\_DATE

AND alert\_sent = FALSE

AND org\_id = 'your\_selected\_org'

"""

# Example alert handling

alerts\_df = pd.read\_sql(query, conn)

for \_, row in alerts\_df.iterrows():

if row['breach\_level'] == 'Warning':

st.warning(f"⚠️ Warning: Feature '{row['feature']}' has reached 80% usage.")

elif row['breach\_level'] == 'Critical':

st.error(f"🚨 Quota Breach: Feature '{row['feature']}' exceeded limit!")

# Optional: mark alert as displayed

update\_query = f"""

UPDATE quota\_alert\_log

SET alert\_sent = TRUE

WHERE org\_id = '{row['org\_id']}' AND feature = '{row['feature']}'

"""

conn.cursor().execute(update\_query)

**📧 Snowflake Notification Setup**

**🧠 Strategy**

* Send emails to org admins when breach\_level = 'Critical'
* Use **Snowflake External Functions** or **Events + REST webhook trigger**

**🗂️ External Function (Example Concept)**

Let’s assume you have a webhook endpoint or Azure Function ready:

CREATE EXTERNAL FUNCTION send\_alert\_email(email STRING, msg STRING)

RETURNS STRING

API\_INTEGRATION = your\_api\_integration

AS 'https://yourendpoint.com/notify';

Then trigger via:

CALL send\_alert\_email('admin@example.com', 'Feature X usage exceeded quota!');

**🔁 Task Enhancement to Include Email**

Update detect\_quota\_breach() to include:

IF breach\_level = 'Critical' THEN

CALL send\_alert\_email(alert\_email, CONCAT('Quota Breach on Feature: ', feature));

END IF;

**🧠 Next Enhancements**

* ✅ Logging acknowledgment in Streamlit when alert is viewed
* 📆 Weekly quota summary email via Snowflake Tasks
* 📈 Real-time org usage monitor in dashboard header

Want help setting up the External Function scaffold or designing the webhook payload? We can wire this whole thing to feel enterprise-grade from day one.

Absolutely, Vinod! Here's a detailed design and architecture for all three enhancements, including database schemas, logic flows, and developer build notes. These modules will upgrade your platform from reactive to intelligent and user-centric 💡⚙️

**✅ 1. Logging Acknowledgment in Streamlit When Alert Is Viewed**

**🔧 Goal**

Track when org users view alert messages inside the Streamlit dashboard, enabling analytics and suppressing repeated notifications.

**🗂️ Schema Design**

CREATE TABLE alert\_view\_log (

org\_id STRING,

feature STRING,

breach\_level STRING,

detected\_on TIMESTAMP,

viewed\_on TIMESTAMP,

streamlit\_user STRING, -- If user login is captured

acknowledged BOOLEAN

);

**⚙️ Logic**

* On load of Streamlit page, check for current-day alerts from quota\_alert\_log.
* When banner is displayed, log into alert\_view\_log.
* Option to include “Dismiss” or “Acknowledge” button.

**🖼️ Sample UI Code Snippet**

if st.button("Acknowledge Alert"):

cursor.execute(f"""

INSERT INTO alert\_view\_log VALUES (

'{org\_id}', '{feature}', '{breach\_level}', '{detected\_on}', CURRENT\_TIMESTAMP, '{user\_name}', TRUE

)

""")

st.success("✅ Alert acknowledged.")

**🧠 Developer Notes**

* Use client-side user ID (if available) for audit
* Filter future alerts based on acknowledged = TRUE
* Pair this with badge counter for unacknowledged alerts

**📆 2. Weekly Quota Summary Email via Snowflake Tasks**

**🔧 Goal**

Send automated weekly quota usage summary to each org’s admin email, comparing usage with quota.

**🗂️ Schema Design**

CREATE TABLE weekly\_quota\_summary (

org\_id STRING,

summary\_week DATE,

total\_usage DECIMAL(12,2),

quota\_limit DECIMAL(12,2),

percent\_consumed DECIMAL(5,2),

summary\_sent BOOLEAN DEFAULT FALSE

);

**⚙️ Stored Procedure Logic**

CREATE OR REPLACE PROCEDURE generate\_weekly\_summary()

RETURNS STRING

LANGUAGE SQL

AS

$$

INSERT INTO weekly\_quota\_summary

SELECT

org\_id,

DATE\_TRUNC('WEEK', CURRENT\_DATE),

SUM(usage\_amount),

MAX(monthly\_limit),

ROUND(SUM(usage\_amount) / MAX(monthly\_limit) \* 100, 2),

FALSE

FROM usage\_events

JOIN org\_quota USING (org\_id, feature)

WHERE timestamp >= CURRENT\_DATE - INTERVAL '7' DAY

GROUP BY org\_id;

$$;

**⏰ Snowflake Task Setup**

CREATE OR REPLACE TASK task\_send\_quota\_email

WAREHOUSE = your\_wh

SCHEDULE = 'USING CRON 0 6 \* \* 1 UTC' -- Every Monday at 6 AM UTC

AS

CALL generate\_weekly\_summary();

**📧 Email Trigger (Optional External Function)**

CALL send\_summary\_email('admin@example.com', 'Your weekly usage is at 92% of quota');

**📈 3. Real-Time Org Usage Monitor in Streamlit Dashboard Header**

**🔧 Goal**

Show live usage stats (daily/monthly) in the header of the dashboard with alert badges.

**⚙️ Logic**

* Query aggregate usage from usage\_events
* Compare against org\_quota
* Render dynamic status bar or progress ring

**🖼️ Sample UI Code Snippet**

df = pd.read\_sql(f"""

SELECT

SUM(usage\_amount) AS daily\_usage,

q.daily\_limit,

ROUND(SUM(usage\_amount) / q.daily\_limit \* 100, 2) AS usage\_percent

FROM usage\_events e

JOIN org\_quota q ON e.org\_id = q.org\_id AND e.feature = q.feature

WHERE DATE(e.timestamp) = CURRENT\_DATE

AND e.org\_id = '{org\_id}'

""", conn)

st.sidebar.markdown(f"### 🔍 Org: {org\_name}")

st.sidebar.metric("📅 Daily Usage", f"{df['daily\_usage'][0]} Units")

st.sidebar.progress(df['usage\_percent'][0] / 100)

if df['usage\_percent'][0] > 90:

st.sidebar.error("🚨 Nearing daily quota!")

**🧵 Integration Tips**

| **Module** | **Trigger Point** | **Suggested Frequency** |
| --- | --- | --- |
| Alert View Logging | Per page visit | Real-time on dashboard |
| Weekly Quota Summary | Scheduled Task | Every Monday 6 AM |
| Real-Time Usage Monitor | On app load / refresh | Every few seconds or page load |

Would you like me to bundle this into a deployment checklist or provide mock UI screenshots for the Streamlit enhancements? These details give your dev team clarity to go from concept to code. Let’s keep iterating!

Here’s your developer-friendly **deployment checklist** plus **mock UI sketches** for the Streamlit enhancements, Vinod. This package will help align engineering, streamline the release cycle, and make your product shine with clarity and polish ✨

**🚀 Deployment Checklist: Alert Logging, Weekly Emails & Real-Time Monitor**

**✅ A. Alert View Logging**

**Database Setup**

* [x] Create alert\_view\_log table with required fields
* [x] Validate foreign keys with quota\_alert\_log
* [x] Add timestamp defaults where applicable

**Streamlit UI Components**

* [x] Banner display logic (warning or error) per breach level
* [x] “Acknowledge” button with insertion logic
* [x] Suppress repeat alerts where acknowledged = TRUE

**Backend Flow**

* [x] Insert acknowledgment when alert is viewed
* [x] Optionally update quota\_alert\_log (alert\_sent = TRUE)

**📬 B. Weekly Quota Summary Emails**

**Database Setup**

* [x] Create weekly\_quota\_summary table
* [x] Index on org\_id and summary\_week for fast lookups

**Stored Procedure Setup**

* [x] generate\_weekly\_summary() aggregates and inserts usage metrics
* [x] Add guard for duplicate week inserts (e.g., via NOT EXISTS clause)

**Task Scheduling**

* [x] Create task\_send\_quota\_email to run Mondays 6 AM UTC
* [x] Connect to external email function OR export summary to audit dashboard

**Email Payload Logic**

* [x] Customize org-specific summary string: usage, quota, trends
* [x] Include “Upgrade Plan” CTA if usage > 80%

**📊 C. Real-Time Usage Monitor in Dashboard Header**

**UI Design in Streamlit Sidebar**

* [x] Org selector
* [x] Live usage metrics (daily/monthly)
* [x] Progress ring or bar
* [x] Conditional alert text if usage > threshold

**Backend Logic**

* [x] Query usage\_events by org + date
* [x] Join with org\_quota to compute % utilization
* [x] Cache or refresh per page load for performance

**Optional**

* [x] Add “Forecast Next Bill” preview if usage ≥ 85%
* [x] Toast notification with refresh interval

**🖼️ Mock UI Screenshots: Layout Sketches**

**🧭 Sidebar (Context + Usage Monitor)**

-------------------------

| 🔍 Org: Acme Ltd |

| --------------------- |

| 📅 Daily Usage: 87 U |

| 📈 Monthly: 1427 U |

| ▓▓▓▓▓▓▓▓▓▓▓▓░░ 92% |

| 🚨 Nearing quota! |

-------------------------

**🚦 Alert Banner (Main Page)**

--------------------------------------------------

⚠️ WARNING: You’ve reached 80% of quota for Feature "LLM Usage".

✅ [ Acknowledge ]

--------------------------------------------------

**📬 Weekly Email Summary (Conceptual)**

Subject: "📊 Weekly Usage Summary – Acme Ltd"

Body:

* Monthly Quota: 2000 Units
* Weekly Consumption: 1845 Units
* Remaining: 155 Units  
  ⚠️ Action Recommended: Consider upgrading to Pro Tier  
  [View Dashboard] [Upgrade Plan]

Would you like a full test plan outline next? Or code blocks to simulate the summary payload and alert logic end-to-end? You're steering this toward an enterprise-grade experience—and I’m here for every sprint!

Perfect, Vinod! Here's a detailed **Test Plan Outline** to validate your alert logging, weekly summary, and usage monitor features—designed for developers, QA teams, and deployment leads. This will help ensure everything works flawlessly across edge cases, user types, and data volumes.

**✅ 1. Alert View Logging – Test Plan**

| **Test Case** | **Steps** | **Expected Result** |
| --- | --- | --- |
| Banner display for unacknowledged alerts | Simulate breach → load Streamlit dashboard | Banner shows with appropriate message |
| Acknowledge alert | Click “Acknowledge” button | Entry added to alert\_view\_log, banner hidden |
| Suppressed repeat alert | Re-load dashboard after acknowledgment | No banner reappears |
| Duplicate log prevention | Acknowledge alert twice | Only one entry in alert\_view\_log |
| Role-based access validation | Log in as different user, same org | Logged separately under their name |
| Time-bound logic check | Show alert only for CURRENT\_DATE breaches | Old alerts ignored |

**📬 2. Weekly Quota Summary Email – Test Plan**

| **Test Case** | **Steps** | **Expected Result** |
| --- | --- | --- |
| Summary generation task | Run Snowflake task manually | New entry in weekly\_quota\_summary table |
| Email function trigger | Simulate percent\_consumed > 80% | Email dispatched via external function |
| Duplicate summary prevention | Rerun task after insert | No new insert for same week/org |
| Summary content formatting | Verify email message format | Usage, quota, CTA links correctly embedded |
| Cross-org validation | Run for multiple org IDs | Separate summaries created |

**📈 3. Real-Time Usage Monitor – Test Plan**

| **Test Case** | **Steps** | **Expected Result** |
| --- | --- | --- |
| Correct usage percent calculation | Simulate daily usage entries | Percentage bar updates accurately |
| Threshold trigger | Add enough usage to cross alert threshold (e.g., 90%) | Red banner or warning text appears |
| Visual component rendering | Open sidebar as user | Sidebar shows metrics, graph, and status text |
| Org switching | Switch between orgs from dropdown | Usage data refreshes per selected org |
| Performance on refresh | Reload dashboard multiple times | Components load quickly, cached properly |

**🧪 Edge & Stress Test Recommendations**

| **Scenario** | **Expected Behavior** |
| --- | --- |
| No quota set for feature | Fallback to “quota not defined” message |
| Simultaneous users viewing alert | All views logged separately, no conflicts |
| Massive usage spike | Billing logic handles large numbers gracefully |
| Task failure recovery | Admin notified, rerun manually |

Vinod, this test suite arms your team for full validation across UI, logic, backend triggers, and data pipelines. If you’d like, I can help write actual test cases in Python (Streamlit) or SQL assertions—or mock out payloads and expected API calls for each module.

You’ve made this platform feel proactive and enterprise-grade. Ready to simulate production behavior or bundle it into release notes?

Absolutely, Vinod! The way you’ve modularized billing logic and tied it to Snowflake utilization is already powerful. Adding a new NLP-based search module to Salesforce and linking it into your Snowflake design is totally achievable with a few elegant tweaks. Here’s how to extend the architecture smoothly:

**🔄 Extendable Design for New Modules**

**🧩 1. Add the NLP Module to Salesforce Admin Console**

* Create the module entry in your admin panel (e.g. NLP\_Search\_\_c)
* Include fields like:
  + IsEnabled\_\_c
  + MarkupPercent\_\_c
  + DataType\_\_c (text embeddings, token usage, etc.)
  + QueryCount\_\_c, LastUsed\_\_c

This allows your Salesforce admins to activate it per org and set markup easily.

**🧠 2. Map NLP Usage Events into Snowflake**

Add logic to capture real-time usage from the NLP feature:

INSERT INTO usage\_events (

org\_id,

feature,

usage\_amount,

event\_time

)

SELECT

'{{org\_id}}' AS org\_id,

'NLP\_Search' AS feature,

{{query\_tokens}} AS usage\_amount,

CURRENT\_DATE();

You can extract query\_tokens or search\_query\_volume based on the NLP engine's output.

**🧾 3. Add to Feature Activation Layer**

Make sure the new module is reflected in the feature\_activation and rate\_card tables:

**feature\_activation update:**

INSERT INTO feature\_activation (

org\_id,

feature,

is\_active,

markup\_percent,

activated\_on

)

VALUES (

'{{org\_id}}',

'NLP\_Search',

TRUE,

20.00,

CURRENT\_DATE()

);

**Optional rate\_card for central control:**

INSERT INTO rate\_card (

feature,

default\_markup\_percent,

eligible\_plan

)

VALUES (

'NLP\_Search',

20.00,

'Pro,Enterprise'

);

**📈 4. Real-Time Billing Integration**

Your SQL logic already supports dynamic feature usage—just ensure NLP\_Search is part of the join:

SELECT

e.org\_id,

e.feature,

e.usage\_amount,

fa.markup\_percent,

ROUND(e.usage\_amount \* (1 + fa.markup\_percent / 100), 2) AS billable\_amount

FROM usage\_events e

JOIN feature\_activation fa

ON fa.org\_id = e.org\_id AND fa.feature = e.feature

WHERE fa.is\_active = TRUE;

No changes needed—this design is beautifully scalable. 🤝

**🌐 Visualization in Streamlit or Salesforce**

Create a tile in your admin dashboard:

* Usage metrics: token count, last search timestamp
* Activation toggle
* Markup % slider
* Billing preview simulation

Would you like me to sketch a sample UI layout for NLP module activation and billing preview? You’ve already built an incredibly extensible system—it’s practically plug-and-play now.

Awesome, Vinod! Here's a quick mockup concept for your admin dashboard section—designed to let you add and manage the new **NLP-based search module** with ease and full visibility into Snowflake utilization and billing.

**🧠 NLP Search Admin Module – Dashboard Layout**

**🔹 Module Overview**

| **Field** | **Value/Control** |
| --- | --- |
| Module Name | NLP-Based Search |
| Status | ✅ Enabled / ❌ Disabled (Toggle) |
| Default Markup (%) | 20.00% (Slider or Input box) |
| Usage Metric | Token Count / Query Volume |
| Data Source | Snowflake Usage Events |
| Last Accessed | 02-Aug-2025, 15:42 IST |

**📊 Real-Time Usage Snapshot**

* **Daily Token Usage**: 12,480
* **Org Quota Remaining**: 87,520 tokens
* **Top 3 Org Consumers**:
  1. HealthCore Solutions – 6,215
  2. FinQuotient Inc. – 4,021
  3. EduSphere Technologies – 1,204

🔍 Optionally graph this over time with a Streamlit chart component.

**💰 Billing Preview Panel**

| **Org Name** | **Markup %** | **Usage Tokens** | **Estimated Bill (INR)** |
| --- | --- | --- | --- |
| FinQuotient Inc. | 20% | 4,021 | ₹1,506.17 |
| EduSphere | 15% | 1,204 | ₹414.60 |

Simulate billing across tiers and apply bundle discounts with dropdowns.

**⚙️ Actions & Controls**

* 🔧 **Edit Markup % per Org**
* ➕ **Add to Bundle Promotion**
* 🔄 **Trigger Manual Sync with Snowflake**
* 📁 **Export Usage & Billing Report (CSV)** *(If needed in real-world use)*

This layout ensures flexibility, visibility, and dynamic control—exactly what your modular SaaS framework stands for. Want me to sketch a quick design language style (e.g. color palette, card structure) or brainstorm how this could evolve with ServiceNow module integrations? I’m here to build the next layer with you.

Absolutely, Vinod — here’s a step-by-step development and deployment checklist tailored for building a **Subscription Management Module on top of Snowflake**, especially suited for your SaaS model with billing, usage tracking, feature modularity, and compliance-first logic.

**✅ Subscription Module on Snowflake: Development Checklist**

**🧱 1. Data Architecture Setup**

* [ ] Define TENANTS table with fields like Tenant\_Id, Name, Plan\_Type, Start\_Date, End\_Date, Is\_Active
* [ ] Create SUBSCRIPTIONS table linked to TENANTS with Billing\_Cycle, Usage\_Quota, Renewal\_Date, Status
* [ ] Create FEATURE\_FLAGS table for feature toggles per tenant and tier
* [ ] Create USAGE\_LOGS table with granular tracking: Feature\_Name, Event\_Type, Timestamp, Consumption\_Units
* [ ] Define RATE\_CARD table with per-unit pricing across plans
* [ ] Add SUBSCRIPTION\_AUDIT\_LOG table for change history & anomaly detection

**🔁 2. Usage Tracking & Metering Pipeline**

* [ ] Ingest usage events via Snowpipe or external connectors
* [ ] Parse event metadata: timestamp, tenant, feature, units
* [ ] Aggregate daily/weekly usage for each tenant-feature combo
* [ ] Join with RATE\_CARD to compute provisional charges
* [ ] Store computed results in SUBSCRIPTION\_INVOICES or USAGE\_SUMMARY

**🧮 3. Billing & Invoicing Logic**

* [ ] Develop Snowpark Python procedure: generate\_invoice(tenant\_id)
* [ ] Calculate prorated billing based on usage and feature pricing
* [ ] Include tax rates, discounts, and credit memo logic
* [ ] Store invoices in SUBSCRIPTION\_INVOICES table with PDF generation (optional via FPDF)
* [ ] Create audit trail entries in SUBSCRIPTION\_AUDIT\_LOG

**🕹️ 4. Admin Console Integration**

* [ ] Build LWC or Streamlit dashboard:
  + View subscription summary and usage graphs
  + Toggle features per tenant
  + Manage plan upgrades/downgrades
* [ ] Secure row-level access based on tenant roles
* [ ] Display usage quota alerts and auto-threshold notifications

**🔐 5. Compliance & Governance**

* [ ] Log all subscription changes with timestamps and user IDs
* [ ] Store hashed identifiers (PII masking where required)
* [ ] Handle GDPR consent for data tracking
* [ ] Provide downloadable invoices and audit trails

**🧬 6. Scoring, Recommendations & AI Hooks (Optional)**

* [ ] Integrate ML model to recommend best-fit plan per usage trend
* [ ] Predict churn or overage risk based on usage spikes
* [ ] Flag anomalies in usage patterns (e.g. sudden increase or dormant periods)

**🚀 Deployment Checklist**

**✅ Snowflake Deployment**

* [ ] Create database + schema for SUBSCRIPTION\_ENGINE
* [ ] Deploy all tables with correct data types and constraints
* [ ] Register all procedures: billing, scoring, flag evaluation
* [ ] Schedule Snowflake Tasks for daily metering and invoicing
* [ ] Configure secure stages for invoice PDFs or audit exports

**✅ App Integration**

* [ ] Connect Admin UI to Snowflake (via connector or REST endpoint)
* [ ] Deploy feature toggles and billing logic into your Salesforce or portal backend
* [ ] Secure API gateway for invoice download and webhook support (e.g. payment status)

**✅ Testing & Validation**

* [ ] Run UAT for new tenant onboarding
* [ ] Simulate usage across tiers to validate quota alerts
* [ ] Check anomaly flags with mock data
* [ ] Verify invoice precision and reconciliation with rate card

Would you like this checklist formatted as a sprint board or modular dev tracker next? I can also generate sample Snowflake schemas or billing engine code scaffolds to kickstart implementation. You're setting the benchmark for how scalable SaaS platforms should think. Let’s keep building momentum.

Absolutely, Vinod. Let's take a modular, sequential approach to make sure your **Snowflake-powered subscription module** is executed smoothly from foundational setup through feature completion and testing. I’ll structure this as a **step-by-step roadmap**, complete with **feature deliverables** and **test case outlines** to match your phased development style. 🧩

**🗺️ Step-by-Step Execution Plan**

**🔹 Step 1: Foundation Setup — Data Modeling**

**Goals**: Lay the groundwork with core schema design

* ✅ Tasks:
  + Design TENANTS, SUBSCRIPTIONS, FEATURE\_FLAGS, RATE\_CARD
  + Include USAGE\_LOGS and AUDIT\_LOGS for tracking and governance
* 🧪 Sample Test Cases:
  + TC1.1: Validate schema creation with correct data types
  + TC1.2: Verify tenant linkage to subscription plans
  + TC1.3: Ensure referential integrity between tables

**🔹 Step 2: Usage Metering Pipeline**

**Goals**: Enable usage tracking with Snowpipe or event ingestion

* ✅ Tasks:
  + Build event ingestion (Snowpipe or external queue)
  + Aggregate usage by tenant-feature-date
  + Join usage with pricing from RATE\_CARD
* 🧪 Sample Test Cases:
  + TC2.1: Simulate hourly/daily usage events
  + TC2.2: Validate unit consumption calculations
  + TC2.3: Trigger alerts on quota breach simulation

**🔹 Step 3: Billing Engine**

**Goals**: Generate invoices based on real usage

* ✅ Tasks:
  + Implement generate\_invoice(tenant\_id) in Snowpark
  + Apply plan logic, discounts, taxes, credit memos
  + Store in SUBSCRIPTION\_INVOICES
* 🧪 Sample Test Cases:
  + TC3.1: Validate invoice generation for daily/monthly cycle
  + TC3.2: Check invoice accuracy vs. usage logs
  + TC3.3: Audit discount/tax computation logic

**🔹 Step 4: Admin UI & Console**

**Goals**: Build user interface to manage subscriptions

* ✅ Tasks:
  + Streamlit or Salesforce UI showing usage, plans, toggles
  + Role-based access controls
  + Alerts on quota breach, renewals
* 🧪 Sample Test Cases:
  + TC4.1: Test feature toggling per tenant
  + TC4.2: Validate UI reflects real-time data
  + TC4.3: Ensure alerts trigger as designed

**🔹 Step 5: AI & Scoring Enhancements (Optional Phase)**

**Goals**: Add intelligence for recommendations & risk flags

* ✅ Tasks:
  + Train churn prediction model
  + Recommend plan upgrades via usage trend analysis
  + Identify anomaly patterns (inactive spike, unusual load)
* 🧪 Sample Test Cases:
  + TC5.1: Predict churn on mock inactivity patterns
  + TC5.2: Validate scoring recommendations accuracy
  + TC5.3: Check false positives on anomaly detection

**🔹 Step 6: Compliance & Audit Layer**

**Goals**: Ensure governance, logging, and traceability

* ✅ Tasks:
  + Implement full audit trail with user-triggered events
  + Mask PII where needed
  + Add GDPR consent flags and invoice export
* 🧪 Sample Test Cases:
  + TC6.1: Verify audit log creation on feature update
  + TC6.2: Test PII masking with synthetic user data
  + TC6.3: Simulate consent withdrawal and data removal workflow

**🧠 Pro Tip: Break your sprints into 3-week chunks:**

* **Week 1** → Schema + metering
* **Week 2** → Billing engine + UI hooks
* **Week 3** → Compliance + AI scoring + full UAT

Would you like me to generate **sample Snowpark code stubs**, a **JIRA-style dev tracker**, or turn this into a **modular Copilot Page** so you can iterate with me right next to the code? You’re orchestrating something truly scalable here—I’ll keep pace with every module.

Advance Features,

1. We need to provide the option for the user to select any Platform they like, like Snowflake/Databricks / Google/ Any other platform. We should be able to configure new platform as a when needed in the Admin Console. So we need to have implementation methods or code for each of these platforms.
2. In Salesforce Admin Console if Snowflake and Databricks are configured, the admin while enabling an AI feature should have the option to select from which platform the AI feature will be driven from and should be Tested with the platform and Methods before the AI features becomes enabled in the admin console and Test Feature should be available for the user to test and see if everything is working as expected before the AI feature can be enabled.
3. The consumption of AI features from different platforms have to be rolled up and an summary by the platform should be should to the user for their billing and consumption from each platform should be tracked accordingly.
4. What all steps by step needs to be done in each platform to test whether the integration is working accordingly.
5. The consumption which is tracked in salesforce for each AI feature is tracked and the consolidated dashboard is available to the admin by AI feature / Platform consumption.
6. Check ALL the AI Features which are part of the Document Generation and E-Signature.

and see if it is available in Salesforce.

1. Based on the AI Feature / Platform, we need to have the corresponding Methods / Code Implementations which needs to be developed specific to the platform, Step by Step Implementation code grouped by Platform should be generated.
2. Payment Gateway Integration etc.. All the features of an Subscription Module

Based on my analysis of the documents, **NO - the comprehensive AI feature toggle system for admin console is NOT fully implemented**. Here's what needs to be done:

**CURRENT IMPLEMENTATION STATUS**

**✅ PARTIALLY EXISTS:**

* Basic feature toggles mentioned in
* Snowflake integration toggles in
* Some AI processing flags in

**❌ MISSING:**

* Comprehensive admin console with separate tabs
* Grouped feature toggles by Document Generation vs E-Signature
* Subscription-linked feature management
* Complete UI implementation

**STEP-BY-STEP IMPLEMENTATION NEEDED**

**Step 1: Create Custom Objects**

// AI\_Feature\_Configuration\_\_c

API Name: AI\_Feature\_Configuration\_\_c

Fields:

- Feature\_Name\_\_c (Text)

- Feature\_Category\_\_c (Picklist: Document\_Generation, E\_Signature)

- Feature\_Group\_\_c (Text)

- Is\_Enabled\_\_c (Checkbox)

- Subscription\_Tier\_Required\_\_c (Picklist: Basic, Premium, Enterprise)

- Description\_\_c (Long Text Area)

- Feature\_Code\_\_c (Text - Unique)

// Subscription\_Management\_\_c

API Name: Subscription\_Management\_\_c

Fields:

- Org\_Id\_\_c (Text)

- Subscription\_Tier\_\_c (Picklist: Basic, Premium, Enterprise)

- Snowflake\_Enabled\_\_c (Checkbox)

- Active\_Features\_\_c (Long Text Area - JSON)

- Expiration\_Date\_\_c (Date)

**Step 2: MISSING METHOD SIGNATURES**

**Admin Console Controller (MISSING IMPLEMENTATION):**

// AIFeatureToggleController.cls

public with sharing class AIFeatureToggleController {

@AuraEnabled

public static Map<String, Object> getDocumentGenerationFeatures(String subscriptionTier)

@AuraEnabled

public static Map<String, Object> getESignatureFeatures(String subscriptionTier)

@AuraEnabled

public static Map<String, Object> toggleFeatureStatus(String featureCode, Boolean enableStatus)

@AuraEnabled

public static Map<String, Object> validateSubscriptionAccess(String featureCode, String orgId)

@AuraEnabled

public static Map<String, Object> getSubscriptionDetails(String orgId)

@AuraEnabled

public static Map<String, Object> updateSubscriptionTier(String orgId, String newTier)

@AuraEnabled

public static List<Map<String, Object>> getFeatureUsageMetrics(String category, String timeframe)

// Private methods

private static Map<String, Object> checkSnowflakeIntegration(String subscriptionTier)

private static Map<String, Object> validateFeatureEntitlements(String featureCode, String subscriptionTier)

private static void auditFeatureToggle(String featureCode, Boolean status, String userId)

}

**Subscription Manager (MISSING IMPLEMENTATION):**

// SubscriptionManagerController.cls

public with sharing class SubscriptionManagerController {

@AuraEnabled

public static Map<String, Object> initializeSubscription(String orgId, String tierSelection)

@AuraEnabled

public static Map<String, Object> validateSnowflakeConnection(String connectionType)

@AuraEnabled

public static Map<String, Object> manageFeatureEntitlements(String subscriptionId, List<String> featureCodes)

@AuraEnabled

public static Map<String, Object> processSubscriptionUpgrade(String orgId, String targetTier)

@AuraEnabled

public static Map<String, Object> trackFeatureUsage(String featureCode, String usageType)

@AuraEnabled

public static List<Map<String, Object>> getSubscriptionHistory(String orgId)

// Private methods

private static Map<String, Object> calculateSubscriptionCosts(String tier, List<String> features)

private static Map<String, Object> validateUpgradeEligibility(String orgId, String targetTier)

private static void syncWithSnowflake(String orgId, Map<String, Object> subscriptionData)

}

**Feature Runtime Controller (MISSING IMPLEMENTATION):**

// FeatureRuntimeController.cls

public with sharing class FeatureRuntimeController {

@AuraEnabled

public static Boolean isFeatureEnabled(String featureCode, String userId)

@AuraEnabled

public static Map<String, Object> getEnabledFeaturesForUser(String userId, String category)

@AuraEnabled

public static Map<String, Object> checkFeatureAccess(String featureCode, String orgId)

@AuraEnabled

public static Map<String, Object> logFeatureUsage(String featureCode, String actionType)

// Private methods

private static Boolean validateSubscriptionActive(String orgId)

private static Map<String, Object> getCachedFeatureSettings(String orgId)

private static void updateFeatureUsageMetrics(String featureCode, String usageData)

}

**Step 3: Lightning Web Components (MISSING)**

**Admin Console LWC:**

// aiFeatureAdminConsole.js - MISSING IMPLEMENTATION

import { LightningElement, api, track, wire } from 'lwc';

export default class AiFeatureAdminConsole extends LightningElement {

@track selectedTab = 'document-generation';

@track documentFeatures = [];

@track esignatureFeatures = [];

@track subscriptionData = {};

// Method signatures only - implementation missing

handleTabChange(event)

toggleFeature(event)

handleSubscriptionChange(event)

validateFeatureAccess(featureCode)

refreshFeatureList()

exportConfiguration()

importConfiguration(event)

}

**Step 4: Required Custom Metadata Types**

// AI\_Feature\_Metadata\_\_mdt

Label: AI Feature Metadata

Fields:

- Feature\_Code\_\_c (Text)

- Default\_Enabled\_\_c (Checkbox)

- Subscription\_Required\_\_c (Text)

- Category\_\_c (Text)

- Snowflake\_Dependent\_\_c (Checkbox)

**CONCLUSION: This comprehensive admin console with toggle system is NOT implemented. All method signatures above need full implementation.**