Calo AI Specialist — Retention Agent POC

Executive Summary, Architecture, Prompts, Code, and PRD — 2025-08-26

Executive Summary

This 1-minute prototype demonstrates an AI-powered retention agent that identifies at-risk subscribers and generates tailored offers in UK English or Gulf Arabic. It uses a Hugging Face Inference Router with an OpenAI-compatible client to call GPT-OSS-120B, staying model-agnostic while producing a human-readable plan and a structured JSON block for automation.

الملخص التنفيذي

يعرض هذا النموذج السريع)دقيقة واحدة (وكيلاً ذكياً للاحتفاظ بالعملاء يحدد المشتركين ذوي الخطر ويولّد عروضاً مخصصة مع عميل متوافق مع Hugging Face Inference Router بالعربية الخليجية أو الإنجليزية البريطانية .يعتمد على GPT-OSS-120B، لاستدعاء نموذج OpenAI مما يضمن المرونة واستقلالية اختيار النموذج، مع إنتاج خطة قابلة للقراءة منظمة للتكامل والاتمتة JSON وكتلة.

Architecture

Architecture overview
- UI: Streamlit app with Arabic and English modes.
- Data: CSV upload or synthetic sample generation.
- Scoring: Simple churn score (0 to 100) based on last_order_days, orders_month, lifetime_months, promo_used_recently, avg_spend.
- LLM: HF Inference Router endpoint with OpenAI-compatible client calling `openai/gpt-oss-120b:fireworks-ai`.
- Output: Human-readable plan with bold emphasis, plus a JSON block for system integration.

PRD

Product Requirements (POC)
Goals
- Reduce churn by surfacing targeted, high-quality retention offers quickly.
- Provide bilingual experience: Gulf Arabic and UK English.
- Keep model abstraction to swap providers without code churn.

Scope

- CSV ingestion and sample data generator.
- Churn scoring view with top-N at-risk customers selector.
- Recommendations generated by LLM with a plan + JSON output.

Acceptance Criteria
- App loads and allows language switching.
- User can upload CSV or generate sample data.
- Churn table shows top 5/10/15/20 at-risk customers.
- LLM returns readable paragraphs and a machine-parseable JSON array.
- Bold emphasis visible for key offer parts in both languages.

Metrics (POC)
- Time to generate recommendations.
- Qualitative evaluation of message clarity and relevance.
- Operational readiness for CRM integration (manual review in POC).

Risks & Mitigations
- Arabic rendering in PDFs: prefer DOCX/HTML export or local PDF print with Arabic fonts.
- Model variance: keep prompts deterministic and log all inputs/outputs for QA.

Prompts — English

System prompt (EN)
"You are a marketing assistant specialised for the UK market. Write in professional yet warm UK English."

User prompt (EN)
"1) For each customer, write a short friendly paragraph in English that explains the retention offer with emojis.
2) Then add a <JSON>...</JSON> block containing an array of objects with keys: customer_id, action, message, expected_lift, rationale.
Use UK English tone. Message length ≤ 120 characters.
Customers: [top selected records here]"

القوالب — العربية

الرسالة النظامية (AR)
"أنت مساعد تسويق يتحدث العربية ومخصص لدول الخليج. اكتب بالعربية برسالة ودية ومهنية."

رسالة المستخدم (AR)
"1) اكتب فقرة ودية قصيرة بالعربية لكل عميل تشرح العرض المقترح مع رموز تعبيرية.
2) بعد الفقرات، أضف كتلة <JSON>...</JSON> فيها مصفوفة كائنات بالحقل:
customer_id, action, message, expected_lift, rationale.

القيم بالعربية فقط، وطول message ≤ 120 حرفًا.
السجلات المختارة هنا: [العملاء"]

60s Demo — English

60s Demo Script — English
0–5s: “Hi, I’m [Your Name]. This is a 1-minute AI prototype for Calo.” Launch the app.
5–12s: “I can switch Arabic or English. I’ll pick English for the UK market.” Choose English.
12–18s: “Data can come from CSV or sample data. I’ll generate a clean sample.” Click Generate sample.
18–28s: “Next, we calculate churn risk.” Explain the score and select Top 10.
28–45s: “Now AI generates tailored retention offers.” Generate top 5. Mention HF Router + GPT-OSS-120B.
45–55s: “We get a clean plan plus structured JSON. Key parts are bold, with expected impact per rec.”
55–60s: “Ready as a POC and can plug into CRM and campaigns. Thank you.”

نص الديمو 60 ثانية — العربية

نص الديمو 60 ثانية — العربية
0–5 ثانية: “مرحباً، أنا [اسمك]. هذه نسخة تجريبية دقيقة واحدة.”
5–12 ثانية: “أقدر أبدّل بين العربية والإنجليزية. سأختار الإنجليزية للسوق البريطاني.”
12–18 ثانية: “البيانات من CSV أو بيانات تجريبية. سأولّد عينة نظيفة.”
18–28 ثانية: “نحسب درجة الانسحاب وأختار أعلى 10. شرح المكوّنات واختيار أعلى 10.”
28–45 ثانية: “نولّد عروض احتفاظ مخصصة. توليد أعلى 5. ذكر HF Router + GPT-OSS-120B.”
45–55 ثانية: “خطة منسّقة و JSON متوقّع وتأثير العناصر المهمة بخط عريض. منظّم، مع إبراز العناصر المهمة بخط عريض وتأثير متوقّع.”
55–60 ثانية: “جاهزة كـ POC ويمكن ربطها بـ CRM. شكراً والحملات.”

Code — app.py

```
# app.py
# export HF_TOKEN="hf_..."
# streamlit run app.py

import os
import re
import json
import html
import streamlit as st
import pandas as pd
import numpy as np
from openai import OpenAI

# ----------------------------
# Settings
# ----------------------------
HF_ROUTER = "https://router.huggingface.co/v1"
```

```
DEFAULT_MODEL = "openai/gpt-oss-120b:fireworks-ai"


# ----------------------------
# Client and state
# ----------------------------
def make_client() -> OpenAI:
    token = os.environ.get("HF_TOKEN")
    if not token:
        raise EnvironmentError("HF_TOKEN not set. Please set your Hugging Face token.")
    return OpenAI(base_url=HF_ROUTER, api_key=token)


def init_state():
    if "model" not in st.session_state:
        st.session_state.model = DEFAULT_MODEL
    if "plan_text" not in st.session_state:
        st.session_state.plan_text = ""
    if "parsed_recs" not in st.session_state:
        st.session_state.parsed_recs = None
    if "lang" not in st.session_state:
        st.session_state.lang = "Arabic"
    if "dialect" not in st.session_state:
        st.session_state.dialect = "الخليجية"


# ----------------------------
# LLM helpers
# ----------------------------
def extract_text_from_completion(completion) -> str:
    try:
        choices = getattr(completion, "choices", None) or completion.get("choices")
        first = choices[0]
        msg = getattr(first, "message", None) or first.get("message")
        if isinstance(msg, dict):
            return msg.get("content", "")
        return getattr(msg, "content", "") or getattr(first, "text", "") or ""
    except Exception:
        try:
            return json.dumps(completion, ensure_ascii=False)
        except Exception:
            return str(completion)


def split_natural_and_json(full_text: str):
    if not full_text:
        return "", None
    m = re.search(r"<JSON>([\s\S]*?)</JSON>", full_text, re.IGNORECASE)
    if m:
        natural = (full_text[:m.start()] + full_text[m.end():]).strip()
        return natural, m.group(1).strip()
    m2 = re.search(r"(\{[\s\S]*?\}|\[[\s\S]*?\])", full_text)
    if m2:
```

```
        json_text = m2.group(1).strip()
        natural = (full_text[:m2.start()] + full_text[m2.end():]).strip()
        return natural, json_text
    return full_text, None


# ----------------------------
# Data helpers
# ----------------------------
def validate_uploaded_df(df: pd.DataFrame):
    required = {
        "customer_id",
        "last_order_days",
        "avg_spend",
        "orders_month",
        "lifetime_months",
        "preference",
        "promo_used_recently",
    }
    missing = [c for c in required if c not in df.columns]
    if missing:
        return ("Missing columns: " if st.session_state.lang == "English" else "الأعمدة
الناقصة: ") + ", ".join(missing)
    return None


def generate_sample_subscribers(n=60, lang="Arabic"):
    np.random.seed(42)
    prefs = (
        ["متوازن", "منخفض الكربوهيدرات", "محب اللحوم", "نباتي صارم", "نباتي"]
        if lang == "Arabic"
        else ["Vegetarian", "Vegan", "Meat Lover", "Low Carb", "Balanced"]
    )
    rows = []
    for i in range(1, n + 1):
        last_order_days = int(np.clip(np.random.exponential(12), 0, 120))
        avg_spend = round(np.random.uniform(1.5, 10), 2)
        orders_month = int(np.random.poisson(3))
        lifetime_months = int(np.random.exponential(8))
        pref = np.random.choice(prefs)
        promo_used = np.random.choice([0, 1], p=[0.7, 0.3])
        rows.append(
            {
                "customer_id": f"C{i:04d}",
                "last_order_days": last_order_days,
                "avg_spend": avg_spend,
                "orders_month": orders_month,
                "lifetime_months": lifetime_months,
                "preference": pref,
                "promo_used_recently": int(promo_used),
            }
        )
    return pd.DataFrame(rows)
```

```
def churn_score(df: pd.DataFrame) -> pd.Series:
    score = (
        (df["last_order_days"] * 1.5)
        - (df["orders_month"] * 8)
        - (df["lifetime_months"] * 0.5)
        + (5 * (1 - (df["promo_used_recently"])))
    ) / (df["avg_spend"] + 1)
    s = 100 * (score - score.min()) / (score.max() - score.min() + 1e-6)
    return s.round(1)


# ----------------------------
# UI styling and rendering
# ----------------------------
def style_primary(color: str, lang: str):
    direction_css = (
        \"""
        .stApp { direction: rtl; }
        .stMarkdown, .stText { text-align: right; }
        [data-testid="stSidebar"] { direction: rtl; }
        \"""
        if lang == "Arabic"
        else \"""
        .stApp { direction: ltr; }
        .stMarkdown, .stText { text-align: left; }
        [data-testid="stSidebar"] { direction: ltr; }
        \"""
    )
    st.markdown(
        f\"\"\"
        <style>
        {direction_css}
        body, div, p, span {{
            font-family: "Tajawal","Cairo","Noto Kufi Arabic","Segoe UI",Arial,sans-
serif;
        }}
        div.stButton>button {{
            background: {color};
            color: white;
            border-radius: 8px;
            border: 0;
            padding: 0.5rem 0.8rem;
        }}
        .highlight {{
            background: rgba(0,0,0,0.03);
            padding: 0.9rem 1.1rem;
            border-radius: 10px;
            border: 1px solid #eee;
        }}
        .plan-wrapper {{
            background: #ffffff;
```

```
            border: 1px solid #eee;
            border-radius: 12px;
            padding: 1rem 1.25rem;
        }}
        .plan-wrapper h3 {{
            margin: 0 0 12px 0;
            font-weight: 700;
        }}
        .plan-line {{
            line-height: 1.9;
            margin: 0 0 8px 0;
            font-size: 1.05rem;
        }}
        </style>
        \"\"\",
        unsafe_allow_html=True,
    )


def render_bold(text: str) -> str:
    esc = html.escape(text)
    esc = re.sub(r"\\*\\*(.+?)\\*\\*", r"<strong>\\1</strong>", esc)
    esc = re.sub(r"__(.+?)__", r"<strong>\\1</strong>", esc)
    esc = re.sub(r"(\\d+(?:\\.\\d+)?\\s*[٪%])", r"<strong>\\1</strong>", esc)
    for pat in [r"\\bdiscount\\b", r"\\boffer\\b",
r"\\bfree\\s+(?:delivery|shipping)\\b", r"\\bvoucher\\b", r"\\bcoupon\\b",
r"\\bpromo\\b", r"\\bcode\\b", r"\\bpoints?\\b", r"\\bdouble\\s+points?\\b",
r"\\bsave\\b"]:
        esc = re.sub(pat, lambda m: f"<strong>{m.group(0)}</strong>", esc,
flags=re.IGNORECASE)
    for pat in [r"خصم", r"عرض", r"كوبون", r"قسيمة", r"رمز", r"نقاط",
r"مضاعفة\\s+النقاط", r"(?:شحن|توصيل)\\s+مجاني", r"مجاني"]:
        esc = re.sub(pat, lambda m: f"<strong>{m.group(0)}</strong>", esc)
    return esc


def render_plan_text(text: str, lang: str) -> str:
    lines = [ln.strip() for ln in text.splitlines()]
    cleaned = [l for l in lines if l]
    paras_html = [f"<p class='plan-line'>{render_bold(l)}</p>" for l in cleaned]
    title = "Plan - English text" if lang == "English" else "الخطة - نص عربي"
    return f"<div class='plan-wrapper'><h3>{title}</h3>{''.join(paras_html)}</div>"


# ------------------------------
# Sidebar
# ------------------------------
def sidebar(lang: str):
    st.header("Settings" if lang == "English" else "الإعدادات")
    lang_choice = st.radio("Language / اللغة", ["Arabic", "English"], index=0 if lang
== "Arabic" else 1)
    st.session_state.lang = lang_choice
    lang = lang_choice
```

```
    if lang == "Arabic":
        st.session_state.dialect = st.selectbox("اللهجة", ["الفصحى", "الخليجية"], index=1)
    if lang == "English":
        theme = st.selectbox("Theme color", ["Green", "Blue", "Purple"])
        color_map = {"Green": "#22c55e", "Blue": "#3b82f6", "Purple": "#8b5cf6"}
    else:
        theme = st.selectbox("لون الواجهة", ["أخضر", "أزرق", "بنفسجي"])
        color_map = {"22": "#أخضرc55e", "3": "#أزرقb82f6", "8": "#بنفسجيb5cf6"}
    style_primary(color_map.get(theme, "#22c55e"), lang)
    st.session_state.model = st.text_input("Model ID" if lang == "English" else "معرف النموذج", value=st.session_state.model)
    st.markdown("**Data source**" if lang == "English" else "**مصدر البيانات**")
    uploaded = st.file_uploader("Upload CSV" if lang == "English" else "ارفع ملف CSV", type=["csv"])
    if uploaded is not None:
        try:
            up_df = pd.read_csv(uploaded)
            err = validate_uploaded_df(up_df)
            if err:
                st.error(err)
            else:
                st.session_state["subscribers_df"] = up_df
                st.success("CSV uploaded successfully." if lang == "English" else "تم تحميل بيانات CSV بنجاح.")
        except Exception as e:
            st.error(("Failed to upload file: " if lang == "English" else "فشل تحميل الملف: ") + str(e))
    if st.button("Generate sample data" if lang == "English" else "إنشاء بيانات تجريبية"):
        st.session_state["subscribers_df"] = generate_sample_subscribers(lang=lang)
    if st.button("Clear data" if lang == "English" else "مسح البيانات"):
        st.session_state.pop("subscribers_df", None)
        st.session_state.plan_text = ""
        st.session_state.parsed_recs = None
        st.success("Cleared." if lang == "English" else "تم المسح.")


# ------------------------------
# Tabs
# ------------------------------
def tab_data_view(df: pd.DataFrame, lang: str):
    st.subheader("Data preview" if lang == "English" else "معاينة البيانات")
    st.dataframe(df, use_container_width=True)
    with st.expander("Data quality tips" if lang == "English" else "نصائح جودة البيانات"):
        st.markdown(
            "- Ensure all required columns exist.\\n- Boolean values like promo_used_recently must be 0 or 1.\\n- Numeric columns should not contain text."
            if lang == "English"
            else "- تأكد من وجود جميع الأعمدة المطلوبة كـ القيم المنطقية.\\n- يجب أن تكون 0 أو 1 promo_used_recently بدون قيم نصية في الأعمدة الرقمية."
```

```
        )


def tab_scoring_view(df: pd.DataFrame, lang: str):
    st.subheader("Churn score calculation" if lang == "English" else "حساب درجة الانسحاب")
    try:
        df["churn_score"] = churn_score(df)
    except Exception as e:
        st.error(f"Failed to calculate churn score: {e}" if lang == "English" else f"تعذر حساب درجة الانسحاب: {e}")
        st.stop()
    c1, c2, c3, c4 = st.columns(4)
    c1.metric("Customers" if lang == "English" else "عدد العملاء", len(df))
    c2.metric("Avg score" if lang == "English" else "متوسط الدرجة",
f"{df['churn_score'].mean():.1f}%")
    c3.metric("Max score" if lang == "English" else "أعلى درجة",
f"{df['churn_score'].max():.1f}%")
    c4.metric("Min score" if lang == "English" else "أدنى درجة",
f"{df['churn_score'].min():.1f}%")
    top_n = st.selectbox("Show top at-risk customers" if lang == "English" else "عرض الأعلى خطراً", [20, 15, 10, 5], index=0)
    st.dataframe(
        df.sort_values("churn_score", ascending=False).head(top_n),
        use_container_width=True,
        column_config={
            "churn_score": st.column_config.ProgressColumn(
                "Churn score" if lang == "English" else "درجة الانسحاب",
                help="Relative score 0 to 100" if lang == "English" else "قيمة نسبية من 0 إلى 100",
                min_value=0, max_value=100, format="%.1f%%",
            )
        },
    )
    with st.expander("How churn score is calculated" if lang == "English" else "كيف نحسب درجة الانسحاب"):
        st.markdown(
            "Composite score: longer since last order raises risk, frequent monthly orders and long lifetime reduce risk, and recent promo use reduces risk. Normalised to 0 to 100."
            if lang == "English"
            else "نحسب درجة مركبة تعتمد على عدة عوامل: تأخر آخر طلب يرفع الخطر، تكرار الشراء وطول العمر يقللان الخطر، واستخدام عرض ترويجي مؤخراً يقلل الخطر. نطبع النتيجة إلى نطاق 0 حتى 100."
        )


def _emoji_for_lift(lift: str) -> str:
    s = str(lift or "").strip().lower()
    if s in ["high", "مرتفع", "عالي"]:
        return "🔥"
    if s in ["medium", "متوسط"]:
```

```
        return "👍"
    if s in ["low", "منخفض"]:
        return "🔔"
    return "📈"


def tab_recommendations_view(df: pd.DataFrame, lang: str, dialect: str):
    st.subheader("Generate recommendations" if lang == "English" else "توليد التوصيات")
    df_scored = df.copy()
    if "churn_score" not in df_scored.columns:
        df_scored["churn_score"] = churn_score(df_scored)
    top_k = st.selectbox("Number of top at-risk customers" if lang == "English" else "عدد العملاء الأعلى خطراً", [5, 10, 15, 20], index=0)
    selected = df_scored.sort_values("churn_score", ascending=False).head(top_k)
    st.dataframe(
        selected[["customer_id", "last_order_days", "orders_month", "avg_spend", "churn_score"]],
        use_container_width=True,
        column_config={
            "churn_score": st.column_config.ProgressColumn(
                "Churn score" if lang == "English" else "درجة الانسحاب",
                min_value=0, max_value=100, format="%.1f%%",
            )
        },
    )
    if st.button("Generate retention recommendations" if lang == "English" else "توليد توصيات الاحتفاظ"):
        top_records = selected.to_dict(orient="records")
        if lang == "Arabic":
            sys_prompt = "أنت مساعد تسويق يتحدث العربية ومخصص لدول الخليج. اكتب بالعربية برسالة ودية ومهنية."
            user_prompt = (
                f"استخدم اللهجة: {dialect}. "
                "1) اكتب فقرة ودية قصيرة بالعربية لكل عميل تشرح العرض المقترح مع رموز تعبيرية. "
                "2) بعد الفقرات، أضف كتلة <JSON>...</JSON> فيها مصفوفة كائنات بالحقل: "
                "customer_id, action, message, expected_lift, rationale. "
                "القيم بالعربية فقط، وطول message ≤ 120 حرفًا. "
                f"العملاء: {json.dumps(top_records, ensure_ascii=False)}"
            )
        else:
            sys_prompt = "You are a marketing assistant specialised for the UK market. Write in professional yet warm UK English."
            user_prompt = (
                "1) For each customer, write a short friendly paragraph in English that explains the retention offer with emojis. "
                "2) Then add a <JSON>...</JSON> block containing an array of objects with keys: "
                "customer_id, action, message, expected_lift, rationale. "
                "Use UK English tone. Message length ≤ 120 characters. "
```

```
                f"Customers: {json.dumps(top_records, ensure_ascii=False)}"
            )
        with st.spinner("Generating plan and recommendations..." if lang == "English"
else "جاري توليد الخطة والتوصيات"):
            try:
                client = make_client()
                completion = client.chat.completions.create(
                    model=st.session_state.model,
                    messages=[{"role": "system", "content": sys_prompt}, {"role":
"user", "content": user_prompt}],
                )
                full_text = extract_text_from_completion(completion)
                natural, json_text = split_natural_and_json(full_text)
                if natural:
                    st.session_state.plan_text = natural
                parsed = None
                if json_text:
                    try:
                        parsed = json.loads(json_text)
                    except Exception:
                        parsed = None
                        st.warning("Could not parse JSON." if lang == "English" else
"تعذر قراءة JSON.")
                st.session_state.parsed_recs = parsed
                st.success("Generated successfully." if lang == "English" else " تم
التوليد بنجاح.")
            except Exception as e:
                st.error(("Failed: " if lang == "English" else "فشل: ") + str(e))
    if st.session_state.plan_text:
        st.markdown(render_plan_text(st.session_state.plan_text, lang),
unsafe_allow_html=True)
    if isinstance(st.session_state.parsed_recs, list):
        st.markdown("### Recommendations" if lang == "English" else "### التوصيات")
        for rec in st.session_state.parsed_recs:
            cid = rec.get("customer_id", "-")
            action = rec.get("action", "")
            message_raw = rec.get("message", "")
            lift = rec.get("expected_lift", "")
            rationale = rec.get("rationale", "")
            emoji = _emoji_for_lift(lift)
            with st.expander(f"{emoji} {cid} - {action}"):
                st.markdown(("**Message:** " if lang == "English" else "**الرسالة:**
") + render_bold(message_raw), unsafe_allow_html=True)
                st.markdown(("**Expected impact:** " if lang == "English" else
"**التأثير المتوقع:** ") + f"**{html.escape(str(lift))}**")
                if rationale:
                    st.markdown(("**Rationale:** " if lang == "English" else
"**السبب:** ") + html.escape(str(rationale)))


# ----------------------------
# App
# ----------------------------
```

```
def app():
    st.set_page_config(page_title="Subscription Growth Agent", layout="wide",
initial_sidebar_state="expanded")
    init_state()
    with st.sidebar:
        sidebar(st.session_state.lang)
    if st.session_state.lang == "English":
        st.title("🌎 Subscription Growth Agent - Calo")
        st.markdown("🤠 Helps identify at-risk customers and generate retention
recommendations.")
    else:
        st.title("🌎 وكيل نمو الاشتراكات - Calo")
        st.markdown("🤠 يساعد هذا التطبيق على التعرف على العملاء ذوي خطر الانسحاب
وتوليد توصيات احتفاظ باللغة العربية.")
    if "subscribers_df" not in st.session_state:
        st.info("No data yet. Upload a CSV or generate sample data from the sidebar."
if st.session_state.lang == "English" else "لا توجد بيانات بعد. ارفع CSV أو أنشئ
بيانات تجريبية من الشريط الجانبي.")
        st.stop()
    df = st.session_state["subscribers_df"]
    tabs = (["Data", "Scoring", "Recommendations"] if st.session_state.lang ==
"English" else ["البيانات", "الحساب", "التوصيات"])
    tab_data, tab_scoring, tab_reco = st.tabs(tabs)
    with tab_data:
        tab_data_view(df, st.session_state.lang)
    with tab_scoring:
        tab_scoring_view(df, st.session_state.lang)
    with tab_reco:
        tab_recommendations_view(df, st.session_state.lang, st.session_state.dialect if
st.session_state.lang == "Arabic" else "")
if __name__ == "__main__":
    try:
        app()
    except EnvironmentError as e:
        st.error(str(e))
    except Exception as e:
        st.exception(e)
```