

Relatório Técnico Avançado: Engenharia Financeira e Consumo Programático de Créditos de IA no Google Cloud

1. Introdução à Arquitetura de Faturamento e Escopo dos Créditos

A gestão eficiente de recursos em computação em nuvem transcende a mera alocação de máquinas virtuais; ela exige uma compreensão profunda da taxonomia de serviços e da engenharia de custos, especialmente quando se trata de créditos promocionais com escopo restrito. A análise do documento Credits - VoidNx.csv revela a existência de dois ativos financeiros distintos e de alto valor: um crédito de R\$ 6.432,54 designado como "Trial credit for GenAI App Builder" e um montante de R\$ 3.646,57 atribuído ao "Dialogflow CX Trial". Para um desenvolvedor ou arquiteto de soluções, a utilização eficaz desses recursos não é trivial, pois o ecossistema do Google Cloud Platform (GCP) evoluiu sua nomenclatura e arquitetura de faturamento, criando uma dissociação entre os nomes comerciais listados nos créditos e as APIs técnicas necessárias para consumi-los.¹

O cenário atual apresenta um risco financeiro significativo: a presunção incorreta de que o crédito "GenAI App Builder" cobre todo e qualquer uso de inteligência artificial generativa na plataforma. A evidência técnica indica categoricamente que o uso direto de modelos fundacionais (como o Gemini Pro ou Flash) através da vertexai.generative_models ou do AI Studio, sem a camada de orquestração correta, resultará em cobranças diretas na conta de faturamento principal, ignorando o saldo promocional.² Este relatório dissecaria a infraestrutura técnica necessária para rotear o tráfego de desenvolvimento através dos endpoints elegíveis, garantindo que o consumo seja abatido dos créditos disponíveis e não da carteira do usuário.

A validação do consumo em tempo real é outro desafio crítico abordado nesta análise. O painel de faturamento padrão do Google Cloud opera com latência, o que é inaceitável para testes de carga programáticos desenhados para "queimar" créditos. Portanto, a estratégia proposta baseia-se na exportação de dados de faturamento para o BigQuery, permitindo uma auditoria granular baseada em SKUs (Stock Keeping Units) e IDs de crédito. A distinção entre "custo bruto" e "custo líquido" (Net pricing) torna-se o mecanismo central para verificar se a engenharia de consumo está funcionando conforme o planejado.³

2. Anatomia Técnica e Elegibilidade do Crédito "GenAI App Builder"

O crédito identificado como "Trial credit for GenAI App Builder" (ID: 01fcba1620b82830ec89167e55e859767b7a8525813b0b87545996daede01b04) refere-se a um conjunto de produtos que sofreu um *rebranding* significativo, sendo agora englobado sob o guarda-chuva do **Vertex AI Agent Builder**. A compreensão dessa linhagem é vital, pois a documentação técnica e as bibliotecas de cliente (SDKs) frequentemente utilizam a terminologia nova, enquanto o sistema de faturamento retém a nomenclatura legada.

2.1. O Paradigma do Vertex AI Agent Builder vs. Vertex AI Core

A arquitetura do Google Cloud separa os serviços de IA em camadas distintas de abstração. O "Vertex AI Core" fornece acesso bruto aos modelos e infraestrutura de treinamento, enquanto o "Vertex AI Agent Builder" (anteriormente GenAI App Builder) fornece serviços gerenciados de orquestração, busca semântica e sistemas de RAG (Retrieval-Augmented Generation). O crédito em questão é exclusivo para a segunda categoria. Isso significa que ele subsidia a infraestrutura de "Search" e "Conversation", mas não a inferência bruta isolada.²

Para consumir este crédito programaticamente, o desenvolvedor deve abandonar as chamadas diretas de inferência e adotar uma arquitetura baseada em **motores de busca** (Discovery Engine). O sistema de faturamento do Google Cloud detecta a origem da requisição: se o token de acesso for utilizado contra o endpoint `discoveryengine.googleapis.com`, o sistema mapeia o consumo para os SKUs elegíveis do GenAI App Builder. Se a mesma requisição for feita contra `aiplatform.googleapis.com` (para modelos puros), o crédito não é ativado.²

A tabela a seguir ilustra a elegibilidade dos serviços baseada na análise dos SKUs e documentação oficial de preços, demonstrando onde o desenvolvedor deve focar seus esforços de implementação:

Serviço Técnico	Namespace da API	Status do Crédito	SKU de Faturamento Típico
Vertex AI Search	<code>discoveryengine.googleapis.com</code>	Elegível	Search API Request Count - Standard/Enterprise ⁵
Grounded Generation	<code>discoveryengine.googleapis.com</code>	Elegível	Grounded Generation API ⁵

Vector Search	aiplatform.googleapis.com	Parcial/Condicional	Vector Search Capacity / Streaming Update (Depende do contexto do Agent Builder) ⁶
Gemini API (Direct)	generativelanguage.googleapis.com	Ineligível	Vertex AI Prediction / Generative AI Knowledge ²
AutoML / Training	aiplatform.googleapis.com	Ineligível	Custom Model Training ⁷

2.2. A API de Geração Fundamentada (Grounded Generation)

A "Grounded Generation API" representa o vetor de ataque mais eficiente para consumir este crédito enquanto se utiliza modelos de linguagem de última geração como o Gemini. Diferente da API padrão de geração, esta interface exige que a resposta do modelo seja fundamentada em uma fonte de dados — seja ela o Google Search ou um Data Store proprietário criado no Vertex AI Search. O custo associado a esta chamada é roteado para o orçamento do GenAI App Builder, tornando-a a "porta dos fundos" legítima para usar LLMs com o subsídio promocional.²

O funcionamento interno desta API envolve um pipeline complexo onde o *prompt* do usuário é primeiramente analisado para extrair intenções de busca. Em seguida, o sistema recupera documentos relevantes do índice configurado e, finalmente, o modelo Gemini sintetiza uma resposta baseada estritamente nesses documentos. O valor agregado aqui para o desenvolvedor é duplo: o acesso subsidiado ao poder computacional do Gemini e a redução significativa de alucinações, um problema endêmico em LLMs puros. O crédito cobre tanto a operação de recuperação (retrieval) quanto a geração, desde que invocadas através deste pipeline específico.⁹

2.3. Vertex AI Search e Armazenamento Vetorial

Outro componente massivo para o consumo do crédito é o **Vertex AI Search** (anteriormente Enterprise Search). O crédito cobre a indexação de dados e o armazenamento de vetores associados. Para desenvolvedores que buscam construir sistemas de busca semântica corporativa, este é o momento ideal para indexar grandes volumes de dados não estruturados (PDFs, HTML, TXT), pois o custo de armazenamento e ingestão (Document AI parser integrado) é absorvido pelo crédito. A gratuidade abrange até 10 GiB de armazenamento de índice por mês, mas volumes superiores — ideais para testes de estresse

— consumirão o saldo restante.⁵

A infraestrutura de **Vector Search** subjacente, quando provisionada através do fluxo do Agent Builder, utiliza o crédito para o gerenciamento de *shards* e a capacidade de consulta por segundo (QPS). É crucial notar que criar um índice de Vector Search "standalone" via Vertex AI (fora do contexto do Agent Builder) pode cair em uma área cinzenta de faturamento, onde o crédito pode não ser aplicado automaticamente. Portanto, a recomendação técnica é sempre instanciar os índices de busca através do console ou APIs do **Agent Builder/Discovery Engine** para garantir a cobertura financeira.³

3. Implementação Programática: Roteiros de Código e Consumo

A transição da teoria para a prática exige a implementação de clientes API que apontem especificamente para os serviços discoveryengine. Abaixo, detalham-se os cenários de uso com código Python robusto, configurado para garantir a elegibilidade do crédito.

3.1. Caso de Uso 1: Sistema de RAG Gerenciado (Search & Conversation)

Este cenário é o mais alinhado com o propósito do crédito. O objetivo é criar um motor de busca que ingere documentos e permite consultas em linguagem natural. O consumo do crédito ocorre em duas frentes: no armazenamento dos dados indexados e nas requisições de busca processadas.

Para iniciar, o desenvolvedor deve configurar um "Data Store" no console do Agent Builder e vinculá-lo a um aplicativo. O código a seguir demonstra como realizar uma consulta programática que será faturada contra o crédito. A utilização da biblioteca google-cloud-discoveryengine é mandatória; o uso de bibliotecas genéricas HTTP ou de outras interfaces do Vertex AI pode resultar em faturamento incorreto.

Python

```
from google.cloud import discoveryengine_v1 as discoveryengine
from google.api_core.client_options import ClientOptions

# Configurações do Ambiente
# O Location 'global' é padrão para Vertex AI Search, mas 'us' ou 'eu' podem
```

```
ser usados para conformidade.
project_id = "SEU_PROJECT_ID"
location = "global"
data_store_id = "SEU_DATA_STORE_ID"
serving_config_id = "default_search"

def search_enterprise_knowledge_base(search_query: str):
    """
    Executa uma busca semântica contra o Data Store do Vertex AI Search.
    Esta chamada consome o SKU 'Vertex AI Search: Search API Request Count'.
    """

    # Configuração explícita do endpoint para garantir roteamento correto
    client_options = (
        ClientOptions(api_endpoint=f"{location}-discoveryengine.googleapis.com")
        if location != "global"
        else None
    )

    # Instanciação do cliente específico do Discovery Engine
    client = discoveryengine.SearchServiceClient(client_options=client_options)

    # Construção do nome do recurso seguindo a hierarquia estrita do GCP
    serving_config = client.serving_config_path(
        project=project_id,
        location=location,
        data_store=data_store_id,
        serving_config=serving_config_id,
    )

    # Configuração da requisição com especificações de conteúdo
    # O retorno de snippets extractivos aumenta o valor da resposta sem custo
    # adicional de SKU base
    content_search_spec = discoveryengine.SearchRequest.ContentSearchSpec(
        snippet_spec=discoveryengine.SearchRequest.ContentSearchSpec.SnippetSpec(
            return_snippet=True
        ),
        extractive_content_spec=discoveryengine.SearchRequest.ContentSearchSpec.ExtractiveContentSpec()
    )
```

```

        max_extractive_answer_count=1
    ),
)

request = discoveryengine.SearchRequest(
    serving_config=serving_config,
    query=search_query,
    page_size=10,
    content_search_spec=content_search_spec,
    # Habilita expansão de query para melhorar recall (coberto pelo crédito)
    query_expansion_spec=discoveryengine.SearchRequest.QueryExpansionSpec(
        condition=discoveryengine.SearchRequest.QueryExpansionSpec.Condition.AUTO,
    ),
    spell_correction_spec=discoveryengine.SearchRequest.SpellCorrectionSpec(
        mode=discoveryengine.SearchRequest.SpellCorrectionSpec.Mode.AUTO
    ),
)
try:
    response = client.search(request)
    print(f"Resultados para: {search_query}\n")

    for result in response.results:
        data = result.document.derived_struct_data
        print(f"Título: {data.get('title', 'N/A')}")"
        print(f"Link: {data.get('link', 'N/A')}")"
        if data.get('snippets'):
            print(f"Contexto: {data['snippets']['snippet']}")"
            print("-" * 30)

except Exception as e:
    print(f"Falha na execução da busca: {e}")

# Exemplo de chamada
# search_enterprise_knowledge_base("quais são os procedimentos de
segurança?")

```

A execução deste código invoca o SKU Search Standard ou Enterprise, dependendo da configuração do App no console. Ambos são cobertos pelo crédito "GenAI App Builder". A validação técnica ocorre observando o objeto response: se ele retornar dados estruturados

do seu índice, a chamada foi bem-sucedida dentro da infraestrutura subsidiada.¹⁰

3.2. Caso de Uso 2: Geração de Respostas com Grounding (Gemini via Agent Builder)

Este é o método mais sofisticado para utilizar modelos generativos sob a proteção do crédito. Ao invés de apenas buscar documentos, utilizamos a GroundedGenerationServiceClient para solicitar que o Gemini sintetize uma resposta. É imperativo notar que o modelo de faturamento aqui difere do Vertex AI padrão: embora o preço de entrada/saída (tokens) seja baseado no modelo escolhido (ex: Gemini 1.5 Flash), a taxa de serviço de grounding (\$2.50 por 1000 requisições) e a orquestração são cobertas pelo crédito do Agent Builder.⁵

Python

```
from google.cloud import discoveryengine_v1 as discoveryengine

def generate_grounded_response(prompt_text: str):
    """
    Gera uma resposta usando Gemini fundamentada nos documentos do Data Store.
    Consome o crédito 'GenAI App Builder' através da API Grounded Generation.
    """
    client = discoveryengine.GroundedGenerationServiceClient()

    # Referência ao Data Store criado anteriormente
    serving_config =
        f"projects/{project_id}/locations/{location}/collections/default_collection/dataStores/{data_store_id}/servingConfigs/default_search"

    # Configuração da requisição de geração
    request = discoveryengine.GenerateGroundedContentRequest(
        location=client.common_location_path(project=project_id, location=location),
        generation_spec=discoveryengine.GenerateGroundedContentRequest.GenerationSpec(
            model_id="gemini-1.5-flash", # Modelo eficiente e coberto
            temperature=0.7 # Controle de criatividade
        ),
        contents=[
```

```

        discoveryengine.GroundedGenerationContent(
            role="user",
            parts=[discoveryengine.GroundedGenerationContent.Part(text=prompt_text)],
        )
    ],
    # Especificação de Grounding: Força o modelo a usar o Data Store
    grounding_spec=discoveryengine.GenerateGroundedContentRequest.GroundingSpec(
        grounding_sources=
    ),
)
try:
    response = client.generate_grounded_content(request=request)

    # A resposta contém o texto gerado E as citações (claims/facts)
    candidate = response.candidates
    print(f'Reposta Gerada: {candidate.content.parts.text}')

    # Análise de Grounding (Metadados vitais para validação)
    if candidate.grounding_metadata.grounding_support:
        print(f'Nível de Suporte: {candidate.grounding_metadata.grounding_support}')

except Exception as e:
    print(f'Erro na geração fundamentada: {e}')

```

Este script é fundamental para o desenvolvedor que deseja funcionalidades de chat (chatbot) sem incorrer nos custos de APIs de predição padrão. A presença de grounding_metadata na resposta é a prova técnica de que a requisição passou pelo pipeline do Agent Builder.⁹

4. Engenharia de Consumo para Dialogflow CX

O crédito de R\$ 3.646,57 para o Dialogflow CX opera sob uma lógica distinta, baseada em "sessões" e não em tokens ou processamento de armazenamento. O Dialogflow CX é uma máquina de estados conversacional, e o faturamento é acionado a cada interação (Request) ou por duração de áudio processado. Para consumir este crédito de forma válida, é necessário gerar tráfego contra um agente CX configurado.

4.1. Estrutura de Custos e SKUs de Sessão

O crédito cobre especificamente os SKUs listados como Text session e Audio session. No modelo de preços do Dialogflow CX, uma sessão de chat custa aproximadamente \$0.007 por requisição. Isso implica que, para consumir o valor total do crédito, seriam necessárias centenas de milhares de requisições, tornando-o ideal para testes de carga massivos e validação de fluxos complexos.¹²

É importante diferenciar entre o Dialogflow CX (Essentials) e o uso de recursos "Standard" (como Generative Fallbacks) dentro do CX. O crédito listado é para o produto principal, mas o uso de recursos generativos dentro do Dialogflow pode consumir créditos de forma acelerada ou cair em SKUs híbridos. A estratégia mais segura é focar em sessões determinísticas (flows padrão) ou interações de áudio, que possuem um custo unitário mais elevado (\$0.001 por segundo ou \$0.06/minuto dependendo da configuração), permitindo um consumo mais eficiente do saldo.¹²

4.2. Automação de Consumo via Python (Simulação de Sessões)

Para validar a aplicação e consumir o crédito, deve-se criar um script que simule usuários interagindo com o agente. O uso de uuid para gerar IDs de sessão distintos é crucial; manter o mesmo ID de sessão por longos períodos pode bater em limites de cota ou distorcer a métrica de "sessão" para fins de faturamento.

Python

```
import uuid
from google.cloud import dialogflowcx_v3 as dialogflow

def execute_dialogflow_load_test(project_id, location_id, agent_id,
interaction_script):
    """
    Executa um teste de carga simulando múltiplos usuários conversando com o
    agente.
    Consome o crédito 'Dialogflow CX Trial' através de requisições detect_intent.
    """

    client_options = None
    if location_id != "global":
        api_endpoint = f"{location_id}-dialogflow.googleapis.com:443"
        client_options = {"api_endpoint": api_endpoint}
```

```
session_client = dialogflow.SessionsClient(client_options=client_options)

# Itera sobre o 'script' de interações para simular conversas
for interaction in interaction_script:
    # Gera um novo Session ID para cada 'usuário' simulado
    # Isso cria novas sessões lógicas no backend do Dialogflow
    session_id = str(uuid.uuid4())

    session_path = session_client.session_path(
        project=project_id,
        location=location_id,
        agent=agent_id,
        session=session_id,
    )

    text_input = dialogflow.TextInput(text=interaction)
    query_input = dialogflow.QueryInput(text=text_input, language_code="pt-BR")

    request = dialogflow.DetectIntentRequest(
        session=session_path, query_input=query_input
    )

    try:
        response = session_client.detect_intent(request=request)
        print(f"Sessão {session_id[:8]}... | Input: {interaction}")
        print(f"Resposta Agente: {' '.join([msg.text.text for msg in response.query_result.response_messages])}")

        # Validação: O ID da resposta e o Intent detectado confirmam o
        # processamento
        if response.query_result.match.intent:
            print("Intent Detectada:")
            print(response.query_result.match.intent.display_name)
            print("-" * 20)

    except Exception as e:
        print(f"Erro na interação Dialogflow: {e}")

# Dados de Teste
```

```
# interacoes = ["Quero comprar uma passagem", "Verificar saldo", "Falar com atendente"]
# execute_dialogflow_load_test("seu-projeto", "us-central1", "seu-agent-id",
interacoes)
```

Este script ataca diretamente o SKU A1CC-751A-CDCC (Text session) listado no arquivo CSV do usuário.¹ Para acelerar o consumo, recomenda-se paralelizar este script usando asyncio ou threading, simulando centenas de usuários simultâneos, o que configura um teste de estresse legítimo para a infraestrutura do bot.

5. Validação Financeira e FinOps (Otimização e Monitoramento)

A maior ansiedade de quem utiliza créditos de nuvem é a incerteza: "Estou gastando o crédito ou o meu dinheiro?". A resposta definitiva não reside no painel simples do console, mas sim na análise de dados brutos de faturamento.

5.1. O Mecanismo de "Net Pricing" e Latência

O termo "Net pricing" encontrado no CSV¹ indica que o crédito é aplicado sobre o valor líquido dos serviços. No entanto, o sistema de faturamento do Google Cloud não é síncrono. Existe um atraso (latência) que varia de 6 a 24 horas entre o consumo do recurso (API call) e o reflexo financeiro no painel.¹³ Isso significa que um desenvolvedor que roda um script massivo e verifica o saldo 5 minutos depois verá "R\$ 0,00" de custo, o que é um falso negativo perigoso.

Para mitigar isso, a única fonte confiável é o **Cloud Billing Export para o BigQuery**. Esta ferramenta exporta logs detalhados de cada transação de custo, permitindo filtrar por SKUs e identificar se um crédito promocional foi aplicado (campo credits).

5.2. Query de Auditoria Definitiva (BigQuery)

Uma vez ativado o export de faturamento (tabela gcp_billing_export_v1...), a seguinte consulta SQL permite verificar exatamente se os créditos específicos (pelos seus IDs ou nomes) estão abatendo os custos dos serviços utilizados.

SQL

```

SELECT
    invoice.month,
    -- Detalhes do Serviço Consumido
    service.description AS service_name,
    sku.description AS sku_name,
    sku.id AS sku_id,

    -- Detalhes do Crédito Aplicado
    credits.name AS credit_name,
    credits.type AS credit_type,
    credits.amount AS credit_amount, -- Valor negativo indicando desconto

    -- Análise Financeira
    cost AS gross_cost, -- Custo bruto antes do crédito
    (cost + IFNULL(credits.amount, 0)) AS net_cost_to_wallet, -- O que sai do bolso

    usage_start_time
FROM
    `SEU_PROJETO.SEU_DATASET.gcp_billing_export_v1_SEU_BILLING_ID`,
    UNNEST(credits) AS credits
WHERE
    -- Filtragem pelos Créditos Específicos do Usuário
    (credits.name LIKE '%GenAI App Builder%' OR
     credits.name LIKE '%Dialogflow CX Trial%' OR
     credits.name LIKE
     '%01fcba1620b82830ec89167e55e859767b7a8525813b0b87545996daede01b0
     4%') -- ID do CSV

    AND usage_start_time >= TIMESTAMP("2025-01-01") -- Período de interesse
ORDER BY
    usage_start_time DESC
LIMIT 1000

```

Interpretação Estratégica dos Resultados:

- **Credit Amount:** Deve aparecer como um valor negativo.
- **Net Cost to Wallet:** Se esta coluna for consistentemente 0.0, o consumo está 100% coberto.

- **Service Name:** Se aparecerem serviços não esperados (como "Compute Engine" fora do contexto), é um sinal de alerta para revisar a arquitetura.

Esta consulta fornece a "prova de vida" de que os scripts Python estão acionando os mecanismos de faturamento corretos e que o Google está aplicando os créditos conforme as regras de negócio dos SKUs.⁴

6. Conclusão e Recomendações Estratégicas

A análise minuciosa dos créditos disponíveis aponta para uma oportunidade de desenvolvimento robusto em tecnologias de ponta, desde que respeitadas as restrições arquiteturais. O crédito de "GenAI App Builder" deve ser canalizado para a construção de sistemas de **RAG (Retrieval-Augmented Generation)** utilizando a Discovery Engine API e a Grounded Generation API, evitando o uso direto de modelos no Vertex AI Studio. Já o crédito do "Dialogflow CX" deve ser exaurido através de testes de carga em agentes conversacionais complexos, validando fluxos de atendimento e processamento de áudio.

Para o desenvolvedor, a segurança financeira reside na implementação de scripts de validação contínua via BigQuery, superando a latência dos painéis tradicionais. Ao alinhar o código Python com os SKUs elegíveis identificados (como Vertex AI Search: Search API Request Count e Text session), transforma-se um passivo potencial (cobrança inesperada) em um ativo estratégico de aprendizado e inovação tecnológica.

Resumo Operacional:

1. **Prioridade 1:** Ativar Exportação de Faturamento para BigQuery.
2. **Prioridade 2:** Migrar chamadas de vertexai para google.cloud.discoveryengine.
3. **Prioridade 3:** Implementar script de *load testing* para Dialogflow CX.
4. **Prioridade 4:** Monitorar net_cost_to_wallet diariamente via SQL.

Esta abordagem garante a maximização do retorno sobre os mais de R\$ 10.000,00 em créditos disponíveis, com risco financeiro mitigado por engenharia de dados precisa.

Referências citadas

1. Credits - VoidNx.csv
2. PSA / Guide: How to Actually Use the \$1000 "GenAI App Builder" Credit (It's NOT for the standard Gemini API) - Reddit, acessado em dezembro 29, 2025, https://www.reddit.com/r/googlecloud/comments/1ldj0uf/psa_guide_how_to_actually_use_the_1000_genai_app/
3. which GCP products fall under "Trial credit for GenAI App Builder" : r/googlecloud - Reddit, acessado em dezembro 29, 2025, https://www.reddit.com/r/googlecloud/comments/1h2umes/which_gcp_products_fall_under_trial_credit_for/

4. Track your spend-based milestone credits (contract pricing) | Cloud Billing, acessado em dezembro 29, 2025,
<https://docs.cloud.google.com/billing/docs/how-to/spend-milestone-credits>
5. Pricing | Vertex AI Search | Google Cloud, acessado em dezembro 29, 2025,
<https://cloud.google.com/generative-ai-app-builder/pricing>
6. Vector Search | Vertex AI | Google Cloud Documentation, acessado em dezembro 29, 2025, <https://docs.cloud.google.com/vertex-ai/docs/vector-search/overview>
7. Trial credit for GenAI App Builder \$1000, plus \$300 "free trial"? : r/googlecloud - Reddit, acessado em dezembro 29, 2025,
https://www.reddit.com/r/googlecloud/comments/1ilpy9c/trial_credit_for_genai_app_builder_1000_plus_300/
8. Vertex AI APIs for building search and RAG experiences - Google Cloud Documentation, acessado em dezembro 29, 2025,
<https://docs.cloud.google.com/generative-ai-app-builder/docs/builder-apis>
9. Generate grounded answers with RAG | Vertex AI Search - Google Cloud Documentation, acessado em dezembro 29, 2025,
<https://docs.cloud.google.com/generative-ai-app-builder/docs/grounded-gen>
10. Class SearchRequest (0.16.0) | Python client libraries - Google Cloud Documentation, acessado em dezembro 29, 2025,
https://docs.cloud.google.com/python/docs/reference/discoveryengine/latest/google.cloud.discoveryengine_v1alpha.types.SearchRequest
11. Search a data store | Vertex AI Search - Google Cloud Documentation, acessado em dezembro 29, 2025,
<https://docs.cloud.google.com/generative-ai-app-builder/docs/samples/genappbuilder-search>
12. Conversational Agents pricing | Google Cloud, acessado em dezembro 29, 2025,
<https://cloud.google.com/dialogflow/pricing>
13. Understand the Cloud Billing data tables in BigQuery, acessado em dezembro 29, 2025,
<https://docs.cloud.google.com/billing/docs/how-to/export-data-bigquery-tables>
14. Example queries for Cloud Billing data export, acessado em dezembro 29, 2025,
<https://docs.cloud.google.com/billing/docs/how-to/bq-examples>