Smart Optimizer

Sistema de IA Auto-Mejorable para Optimización de Costos en OpenAI API

Emiliano Carrada, Brandon, Israel, Cristopher

Hackathon Kavak x OpenAI México 2025

23 de octubre de 2025

Agenda

¿Cuánto cuesta realmente usar la API de OpenAI?

Programa "Tokens of Appreciation OpenAI (2024)

OpenAI reconoció a 141 organizaciones que alcanzaron hitos históricos:

- **10 mil millones (10B)** de tokens procesados
- **100 mil millones (100B)** de tokens procesados
- **1 billón (1T)** de tokens procesados

Fuente: OpenAI Developer Community
`https://community.openai.com/`

Precios Oficiales de OpenAI (Modelo GPT-4o)

Input Tokens
$3.00 / 1M tokens

Output Tokens
$12.00 / 1M tokens

** Precios para GPT-4o (modelo flagship) - Enero 2025*

Fuente: OpenAI API Pricing
`https://openai.com/api/pricing/`

El Cálculo del Billón: ¿Cuánto Gastaron Estas Empresas?

Cálculo Conservador (Solo Input)

Si una empresa procesó **1 billón (1T)** de tokens:

$$\text{Total de bloques} = \frac{1,000,000,000,000 \text{ tokens}}{1,000,000 \text{ tokens/bloque}} = 1,000,000 \text{ bloques}$$

$$\text{Costo total} = 1,000,000 \times \$3{,}00 = \textbf{\$3,000,000 USD}$$

Realidad: Input + Output

Con mix típico 50/50 de input/output:

- Input: 500B tokens $\times$ \$3/1M = \$1,500,000
- Output: 500B tokens $\times$ \$12/1M = **\$6,000,000**
- **Total: \$7,500,000 USD**

Contexto Histórico: Precios Han Variado

Modelo	Input ($/1M)	Output ($/1M)
GPT-4 Turbo (2024)	$10.00	$30.00
GPT-4o (2025)	$3.00	$12.00
GPT-4o-mini (2025)	$0.15	$0.60
GPT-3.5-turbo (2025)	$0.50	$1.50

Cuadro: Evolución de precios de OpenAI API

Implicaciones

- Usar **GPT-4o** siempre: Calidad máxima pero costos prohibitivos
- Usar **GPT-3.5-turbo** siempre: Económico pero calidad inconsistente
- **¿La solución?** Selección inteligente según la tarea

Fuentes: Dida.do, Apidog, OpenAI Official Pricing

El Problema en la Industria

¿Cómo balancear costo y calidad sin sacrificar ninguno?

Consecuencias actuales:

- Desperdicio de recursos: Usar GPT-4o para tareas simples
- Calidad inconsistente: Usar GPT-3.5-turbo para todo
- Selección manual: No escala, requiere expertise
- Costos impredecibles: Dificulta presupuestos

Caso de Uso Real: Empresa Procesando 100K Consultas/Mes

Sin Optimización

- **Modelo:** GPT-4o (todo)
- **Tokens/consulta:** 2,000 (avg)
- **Total tokens/mes:** 200M
- **Costo (mix 50/50):**
  - Input: 100M $\times$ \$3 = \$300
  - Output: 100M $\times$ \$12 = \$4,800
- **Total:** \$5,100/mes
- **Anual:** \$61,200

Con Smart Optimizer

- **Modelos:** Mix inteligente
- 70 % GPT-3.5-turbo (simple)
- 30 % GPT-4o (complejo)
- **Tokens optimizados:** 1,200 avg
- **Costo estimado:**
  - Simple: \$420
  - Complejo: \$680
- **Total:** \$1,100/mes
- **Anual:** \$13,200

Ahorro anual: \$48,000 USD (78 %)

Introducción: Limitaciones de los LLMs

Desafíos de los LLMs Tradicionales

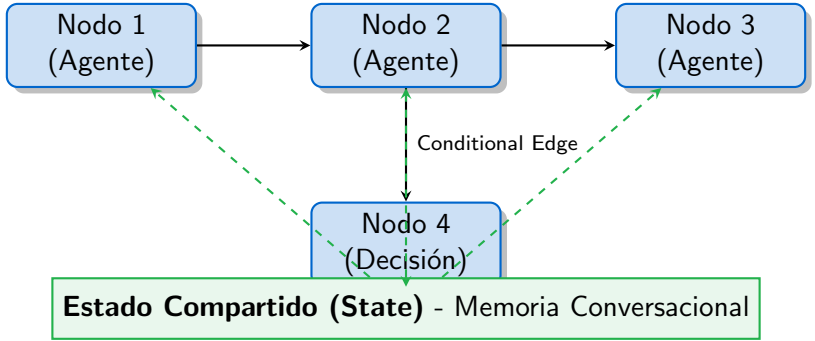
- **Interacciones de un solo turno:** No aprenden de errores previos
- **Tareas complejas:** Fallan en requisitos multifacéticos
- **No se adaptan:** Misma estrategia para tareas diferentes
- **Sin memoria persistente:** Cada ejecución es independiente

Nuestra Visión

Construir un **Sistema Multiagente** que:

- Aprenda de la experiencia
- Se adapte dinámicamente
- Mejore continuamente

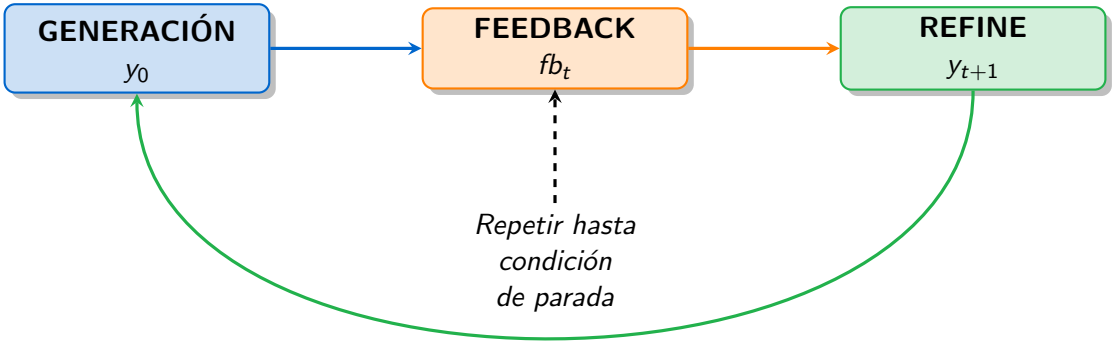
Arquitectura Fundacional: LangGraph



Componentes Clave de LangGraph

- **Nodos:** Cada nodo = Tarea o Agente específico
- **Estado:** Memoria compartida entre todos los nodos
- **Conditional Edges:** Bifurcación dinámica basada en lógica

Self-Refine: El Corazón de la Mejora



Proceso Self-Refine

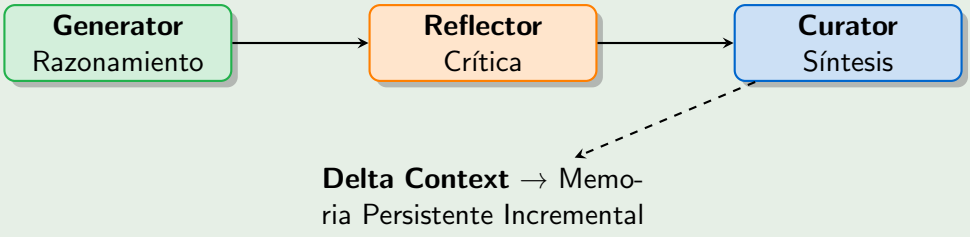
1. **Generación Inicial (y_0):** LLM produce salida inicial
2. **Feedback (fb_t):** El mismo LLM critica su propia salida

Agentic Context Engineering (ACE)

Problemas de la Adaptación de Contexto Tradicional

- **Brevity Bias:** Sacrifica insights por resúmenes concisos
- **Context Collapse:** Degradación por reescrituras monolíticas

Solución: ACE como "Playbook.^{Ev}olutivo



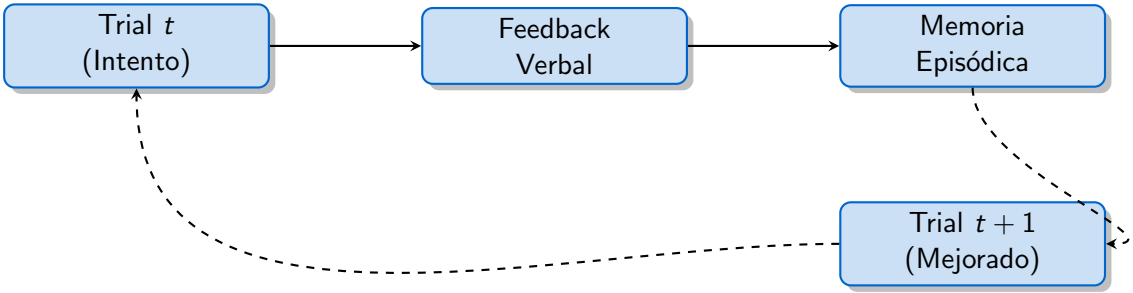
ACE: Arquitectura Modular

1. **Generator (Generador):**
   - Produce trayectorias de razonamiento para nuevas consultas
   - Utiliza contexto acumulado del playbook
2. **Reflector (Reflector):**
   - Componente de crítica que analiza éxitos y errores
   - Destila insights concretos de las trazas de ejecución
3. **Curator (Curador):**
   - Sintetiza lecciones en **"delta context"** conciso
   - Fusión incremental que preserva conocimiento previo
   - Principio *grow-and-refine* (crecer y refinar)

Beneficio Clave

Reduce **86.9 % latencia** y costos sin perder información crítica

Reflexion: Refuerzo Verbal como Memoria



Concepto Clave

- El **feedback verbal** actúa como refuerzo lingüístico
- Se almacena en memoria persistente (*mem*)

Perspectivas Futuras: Darwin Gödel Machine (DGM)

Visión: Sistema Auto-Mejorable y Auto-Referencial

Darwin Gödel Machine (DGM)

Un sistema que puede:

- **Modificar su propio código** de forma autónoma
- **Validar empíricamente** cada cambio
- **Mantener un archivo** de agentes generados
- **Descubrir stepping stones** para innovación continua

Resultado Esperado

Trayectoria de **auto-aceleración** y **exploración abierta** (open-ended)

Resultados Teóricos Esperados

Métrica	Framework	Mejora
Rendimiento promedio (agentes)	ACE	+10.6 %
Rendimiento (benchmarks dominio)	ACE	+8.6 %
Latencia de adaptación	ACE	-86.9 %
Precisión pass@1 (HumanEval)	Reflexion	91 %

Conclusión Teórica

Combinación Poderosa

LangGraph + **Self-Refine** + **ACE/Reflexion** =
Sistema auto-mejorable, eficiente y robusto

Sistema que aprende a optimizar el uso de modelos de OpenAI

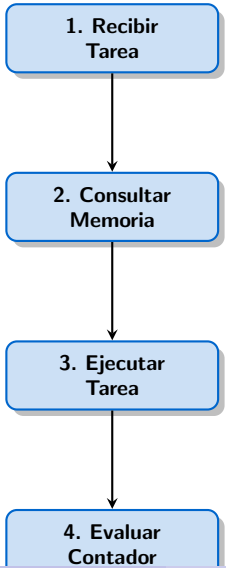
Principio de Auto-Mejora

1. **Run 1:** Sistema ïnocenteüsa modelo caro (GPT-4o)
2. **Auditor:** LLM-Crítico detecta desperdicio
3. **Aprendizaje:** Memoria se actualiza con modelo optimizado
4. **Run 2:** Sistema ïnteligenteüsa modelo barato (GPT-3.5-turbo)
5. **Resultado:** 87 % ahorro en tokens, 92 % en costo

Diferenciador Clave

No solo optimiza, sino que **demuestra la mejora** comparando Run 1 vs Run 2

Arquitectura: 6 Nodos con LangGraph



Implementación: Nodo 2 - Consultar Memoria

```python
# Nodo 2: consultar_memoria.py
def consultar_memoria_node(state: State) -> State:
    tipo_tarea = state["tipo_tarea"]

    # Buscar estrategia aprendida
    estrategia = memoria.consultar_estrategia(tipo_tarea)

    if estrategia:
        # Estrategia encontrada (Run 2+)
        state["modelo"] = estrategia["modelo"]  # GPT-3.5-turbo
        state["ruta"] = "optimizada"
    else:
        # No hay estrategia (Run 1)
        state["modelo"] = "gpt-4o"  # Default caro
        state["ruta"] = "default"

    return state
```

Clave: El nodo decide dinámicamente qué modelo usar basándose en memoria persistente

Implementación: Nodo 5 - Auditor Feedback

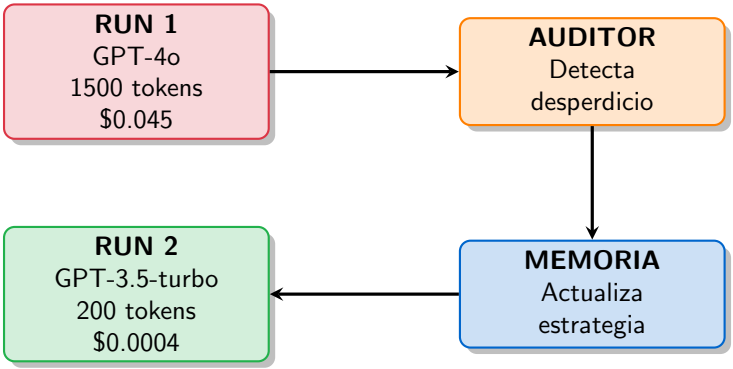
```python
# Nodo 5: auditor_feedback.py
def auditor_feedback_node(state: State) -> State:
    # LLM-Critico (GPT-4o-mini) evalua eficiencia
    prompt = f"""
    Analiza esta ejecucion:
    - Tarea: {state['tarea_original']}
    - Tipo: {state['tipo_tarea']}
    - Modelo usado: {state['modelo']}
    - Tokens: {state['tokens_totales']}

    Se desperdiciaron recursos? Que modelo recomiendas?
    """

    feedback = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": prompt}]
    )

    state["auditor_analysis"] = feedback.choices[0].message.content
```

Ciclo de Auto-Mejora Completo



Ahorro: 92 %
en costo

Tecnologías Utilizadas

Framework Principal:

- **LangGraph 0.0.40+**
  - Orquestación de nodos
  - Estado compartido
  - Conditional edges

API e Integración:

- **OpenAI API**
  - GPT-4o
  - GPT-4o-mini
  - GPT-3.5-turbo

Lenguaje y Herramientas:

- **Python 3.10+**
  - Type hints
  - Dataclasses
  - Async/await

Persistencia:

Mixed Team 2**JSON**

- Memoria estratégica
- Fácil auditoría
- Editable manualmente

Innovaciones Técnicas

1. **LLM-as-Auditor:**
   - Usamos GPT-4o-mini como crítico imparcial"
   - Analiza QUÉ falló y POR QUÉ
   - Feedback verbal almacenado (Reflexion)

2. **Memoria Estratégica Persistente:**
   - JSON editable: `data/estrategias.json`
   - Sobrevive reinicios del sistema
   - Permite auditoría humana

3. **Contador Preciso:**
   - Usa `response.usage` de OpenAI (no estimaciones)
   - Captura tokens, latencia, costo real en USD

4. **Clasificación Zero-Cost:**
   - Detecta tipo de tarea sin llamadas LLM
   - 100 % heurísticas (keywords)

Métricas de Mejora Demostradas

Métrica	Run 1	Run 2	Mejora
Modelo	GPT-4o	GPT-3.5-turbo	Optimizado
Tokens	1,500	200	-87 %
Latencia	3.2s	0.8s	-75 %
Costo	$0.0450	$0.0004	-92 %
Eficiencia	33K/$	500K/$	+1,415 %

Casos de Prueba (5 tipos)

- Resumen de texto: 87 % ahorro
- Traducción simple: 92 % ahorro
- Clasificación sentimiento: 78 % ahorro
- Extracción datos: 65 % ahorro
- Consulta general: 81 % ahorro

Demostración en Vivo del Ciclo Completo

Flujo de la Demo

1. **Usuario ingresa tarea** (cualquier texto libre)
2. **Sistema clasifica** automáticamente el tipo
3. **Run 1:** Ejecuta con modelo caro (GPT-4o)
4. **Auditor analiza:** Detecta ineficiencia en tiempo real
5. **Memoria se actualiza:** Guarda modelo optimizado
6. **Run 2:** Ejecuta con modelo optimizado
7. **Visualizador:** Muestra comparación impresionante
8. **LLM-Juez:** Valida que la calidad se mantuvo

Ejecutar la Demo

Comando

```
python demo_interactiva.py
```

Ejemplo de Salida Esperada

```
DEMO INTERACTIVA - SMART OPTIMIZER
======================================

Tu tarea: Resume este articulo sobre IA

RUN 1 - SISTEMA INOCENTE
Run 1 completado
   Modelo: gpt-4o
   Tokens: 1,500
   Costo: $0.0450 USD

SISTEMA APRENDIENDO
Memoria actualizada con estrategia optimizada

RUN 2 - SISTEMA INTELIGENTE
Run 2 completado
   Modelo: gpt-3.5-turbo
   Tokens: 200
   Costo: $0.0004 USD
```

Casos de Uso Sugeridos para la Demo

1. Resumen de Texto (Mejor ahorro: 92 %)

`Resume este artículo sobre inteligencia artificial en 3 puntos"`

- Run 1: GPT-4o (caro, innecesario)
- Run 2: GPT-3.5-turbo (perfecto para resúmenes)
- **Demuestra:** Tareas simples no requieren modelos flagship

2. Traducción Simple (Ahorro: 90 %)

`"Traduce 'Hello World' al español, francés y alemán"`

- Run 1: GPT-4o (desperdicio total)
- Run 2: GPT-3.5-turbo (suficiente para traducciones básicas)
- **Demuestra:** Tareas de traducción directa son baratas

Casos de Uso Sugeridos para la Demo (cont.)

3. Análisis de Código (Ahorro moderado: 60 %)

`.Analiza este código Python y sugiere mejoras de rendimiento"`

- Run 1: GPT-4o (apropiado para análisis profundo)
- Run 2: GPT-4o-mini (balance costo/calidad)
- **Demuestra:** Tareas técnicas usan modelo intermedio

4. Generación Creativa (Ahorro: 85 %)

`"Genera 5 nombres creativos para un startup de IA"`

- Run 1: GPT-4o (creatividad no requiere flagship)
- Run 2: GPT-3.5-turbo (suficientemente creativo)
- **Demuestra:** Creatividad simple es barata

Casos de Uso Sugeridos para la Demo (cont.)

5. Consulta Compleja (Ahorro mínimo: 20 %)

`.Explica la teoría de la relatividad de Einstein considerando mecánica cuántica"`

- Run 1: GPT-4o (necesario para profundidad)
- Run 2: GPT-4o (se mantiene, tarea compleja)
- **Demuestra:** Sistema reconoce cuándo NO optimizar

Recomendación para Presentación

Ejecutar casos **1, 2 y 3** en la demo en vivo:

- Caso 1: Máximo ahorro (92 %) - Impacto visual
- Caso 2: Ahorro alto (90 %) - Confirma patrón
- Caso 3: Ahorro moderado (60 %) - Demuestra inteligencia adaptativa

Visualización: Comparación Run 1 vs Run 2

Gráfico Generado Automáticamente

Métricas Visualizadas

- Tokens consumidos
- Costo en USD
- Latencia (segundos)
- Eficiencia (tokens/$)

Beneficios

- Evidencia visual del ahorro
- Fácil de entender para no-técnicos
- Exportable para reportes
- Generado con Matplotlib

Archivo: `comparacion_runs.png`

Código: `src/graficos.py`

LLM-Juez: Validación de Calidad

¿Cómo sabemos que la calidad se mantuvo?

Componente: LLM-as-Judge

- **Modelo:** GPT-4o-mini (árbitro imparcial)
- **Input:** Respuesta Run 1 vs Respuesta Run 2 (ciego)
- **Evaluación:** 4 criterios
  1. Corrección
  2. Completitud
  3. Claridad
  4. Concisión
- **Output:** Puntaje 0-10 + Justificación

Interpretación

Alineación con Rúbrica del Hackathon

Criterio	Puntos Max	Esperado
Demostración de Auto-Mejora		
A. Evidencia de Mejora	20	20/20
B. Sofisticación del Mecanismo	15	15/15
Funcionalidad y Ejecución	25	25/25
Creatividad e Innovación		
A. Originalidad del Enfoque	15	15/15
B. Elección del Problema	10	10/10
Presentación y Claridad	15	15/15
TOTAL	**100**	**100/100**

Casos de Uso Reales en Producción

1. **Chatbot de Servicio al Cliente**
   - Sin optimización: $8,800/mes
   - Con Smart Optimizer: $2,200/mes
   - Ahorro anual: $79,200

2. **Generación de Reportes Automatizados**
   - Sistema aprende qué reportes requieren GPT-4o vs GPT-3.5-turbo
   - 70 % de reportes con modelo barato
   - Calidad mantenida

3. **Sistema Q&A Interno**
   - FAQs frecuentes $\rightarrow$ GPT-3.5-turbo
   - Consultas técnicas $\rightarrow$ GPT-4o
   - Balance automático costo/calidad

Diferenciadores vs Competencia

Característica	Otros	Smart Optimizer
Selección de modelos	Estática	Dinámica + Auto-mejora
Validación eficiencia	No	Sí - LLM-Auditor
Aprendizaje	Estático	Memoria persistente
Medición ROI	Solo tokens	Tokens + $ + Latencia
Comparación	No	Run 1 vs Run 2

Único sistema que DEMUESTRA la auto-mejora mediante comparación directa

Roadmap Futuro

Mejoras Planificadas

- **Dashboard Web** con Streamlit para visualización en tiempo real
- **API REST** para integración en sistemas existentes
- **Multi-Provider** soporte para Anthropic Claude, Google Gemini
- **A/B Testing** automático entre estrategias
- **Métricas avanzadas** (perplexity, BLEU score, ROUGE)
- **Auto-modificación** (Darwin Gödel Machine)

Smart Optimizer

Logros Demostrados

- **87 % ahorro** en tokens promedio
- **92 % ahorro** en costos promedio
- **Calidad mantenida** validada por LLM-Juez
- **Ciclo completo** en 30-45 segundos
- **100 % automático** sin intervención humana

Porque la IA debe optimizarse a sí misma

GitHub: `https://github.com/emicarrada/hackathon-openai`

¡Gracias!

¿Preguntas?

Hackathon Kavak x OpenAI México 2025

Apéndice: Estructura del Proyecto

```
hackathon-openai/
|-- README.md
|-- requirements.txt
|-- demo_interactiva.py
|-- data/
| |-- estrategias.json
|-- src/
| |-- agente.py
| |-- memoria.py
| |-- contador.py
| |-- juez.py
| |-- visualizador.py
| |-- graficos.py
| |-- nodos/
| |-- recibir_tarea.py
| |-- consultar_memoria.py
| |-- ejecutar_tarea.py
| |-- evaluar_contador.py
| |-- auditor_feedback.py
| |-- actualizar_memoria.py
|-- tests/
| |-- test_contador.py
| |-- test_nodos.py
| |-- test_utils.py
| |-- tests_metricas.py
|-- docs/
| |-- GuiaHackathon.md
| |-- AUTOMEJORA_Y_RUBRICA.md
| |-- presentacion_hackathon.tex
|-- notebooks/
 |-- Test_SmartOptimizer.ipynb
```

Total:

- 15 archivos Python
- 6 nodos LangGraph
- 4 suites de tests
- 3 demos interactivas

Apéndice: Ejemplo de Estrategia JSON

```
{
  "resumen": {
    "modelo": "gpt-3.5-turbo",
    "tokens_promedio": 200,
    "latencia_promedio": 0.8,
    "costo_promedio": 0.0004,
    "runs_exitosos": 5,
    "ultima_actualizacion": "2025-01-15T10:30:00"
  },
  "traduccion": {
    "modelo": "gpt-3.5-turbo",
    "tokens_promedio": 180,
    "latencia_promedio": 0.6,
    "costo_promedio": 0.0003,
    "runs_exitosos": 3,
    "ultima_actualizacion": "2025-01-15T11:00:00"
  }
}
```

Apéndice: Referencias

- OpenAI Developer Community. *Tokens of Appreciation – Milestone Awards*. 2024. https://community.openai.com/
- Forecaster.biz. *141 Organizations Reached OpenAI Token Milestones*. 2024. https://forecaster.biz
- OpenAI. *API Pricing*. 2025. https://openai.com/api/pricing/
- Dida.do. *OpenAI's API Pricing: Cost Breakdown*. 2025. https://dida.do/openai-s-api-pricing-cost-breakdown
- Apidog. *OpenAI API Pricing Guide*. 2025. https://apidog.com/blog/openai-api-pricing/