Quanto Custa de Fato uma Análise de IA?

Como transformei o uso da OpenRouter em um livro-razão de créditos para o Jiddu sem fingir que todo token vale a mesma coisa

Rafael M. Ehlers · 28 de julho de 2026
rafaelmehlers.com.br/essays/what-does-an-ai-analysis-actually-cost

Quando o Jiddu estava aberto para qualquer pessoa, um inocente botão **Analisar** tinha acesso direto ao meu saldo na OpenRouter. Uma busca curta por falácias podia fazer uma única chamada a um modelo barato. Uma revisão adversarial profunda de um artigo podia chamar vários modelos, usar tokens de raciocínio, consultar um crítico independente e repetir uma etapa que retornasse dados inválidos.

Chamar as duas ações de "uma análise" esconderia quase tudo o que determina seu custo.

Criei os Créditos Jiddu para contabilizar esse gasto compartilhado e, no futuro, permitir que usuários comprem pacotes de créditos em vez de terem acesso ilimitado à conta da plataforma. O sistema de cobrança então precisou responder a uma pergunta menos óbvia: quantos créditos uma análise deve consumir quando seu custo real só é conhecido depois que ela termina?

O sistema que construí tem três camadas:

1. A OpenRouter informa o custo real de cada chamada de modelo concluída.
2. O Jiddu converte o total em um número inteiro de créditos.
3. Uma tabela de tarifas protege a conta antes da execução e preenche as lacunas quando os dados de uso estão ausentes.

Essas camadas cumprem funções diferentes. A tabela de tarifas estima o custo antes da execução. O custo informado pela OpenRouter fecha a cobrança depois. O livro-razão guarda detalhes suficientes para explicar o resultado mais tarde.

Tokens descrevem o trabalho; o custo define o preço

A primeira versão de um sistema de créditos é tentadora de escrever como uma fórmula de tokens:

```
credits = input tokens × input rate + output tokens × output rate
```

Essa fórmula só funciona quando a aplicação controla um modelo com um preço estável. O Jiddu não funciona assim.

Modelos diferentes cobram valores diferentes por entrada e saída. Alguns expõem preços separados para entrada em cache ou tokens de raciocínio. O provedor selecionado por trás de uma rota de modelo também pode importar. Os tokenizadores variam, então um texto idêntico

não produz necessariamente a mesma contagem de tokens em modelos diferentes. Uma revisão de artigo pode misturar modelos mais baratos e mais caros no mesmo trabalho.

Copiar essa lógica de preços para o Jiddu criaria uma versão desatualizada de um catálogo que muda fora da aplicação e tornaria cobranças históricas difíceis de defender depois que um provedor alterasse um preço.

A OpenRouter retorna a contabilidade de uso com cada resposta. Ao lado das contagens de tokens de prompt e conclusão, a resposta pode conter:

```
{
  "usage": {
    "prompt_tokens": 10000,
    "completion_tokens": 2000,
    "total_tokens": 12000,
    "cost": 0.0165,
    "prompt_tokens_details": {
      "cached_tokens": 0
    },
    "completion_tokens_details": {
      "reasoning_tokens": 400
    },
    "cost_details": {
      "upstream_inference_cost": 0.015
    }
  }
}
```

O campo importante para o fechamento é **usage.cost**: o valor cobrado da conta OpenRouter por aquela geração. A OpenRouter já aplicou os preços atuais do modelo selecionado ao uso de entrada, saída, cache e raciocínio.

O Jiddu ainda registra as contagens de tokens para auditorias, depuração, análise de custos e decisões futuras de produto. Ele não refaz o cálculo do preço.

A fórmula de conversão

O Jiddu usa duas variáveis de ambiente:

```
JIDDU_CREDIT_MARGIN_MULTIPLIER=1.5
JIDDU_CREDIT_USD_VALUE=0.01
```

O fechamento é:

```
provider cost = sum of usage.cost for every call in the operation

billable cost = provider cost × margin multiplier

credits = ceil(billable cost ÷ dollar value per credit)
```

Com os valores padrão atuais:

```
credits = ceil(OpenRouter cost × 1.5 ÷ 0.01)
```

Um Crédito Jiddu representa um centavo de valor faturável interno. Isso não significa que um centavo foi pago à OpenRouter nem constitui uma promessa de preço ao consumidor. O Stripe ainda não está conectado a esse sistema.

O multiplicador cobre despesas além da inferência: taxas de pagamento, impostos, solicitações com falha, infraestrutura, atendimento ao cliente e variações de preço. Ele é configurável porque essa economia não deveria ficar escondida no código da aplicação.

Um exemplo completo

Suponha que uma chamada de modelo retorne:

```
prompt tokens:      10,000
completion tokens:   2,000
OpenRouter cost:   $0.0165
```

O Jiddu calcula:

```
$0.0165 × 1.5 = $0.02475

$0.02475 ÷ $0.01 = 2.475 credits

ceil(2.475) = 3 credits
```

O livro-razão registra três créditos, além das contagens originais de tokens, custo do provedor, custo faturável, modelo, identificador da geração e configurações de conversão.

O custo armazenado usa milionésimos inteiros de dólar em vez de um valor de ponto flutuante:

```
$0.0165 = 16,500 micros
```

Usar um inteiro evita o acúmulo de erros binários de ponto flutuante nos registros financeiros. O valor exibido em dólares sempre pode ser reconstruído dividindo o inteiro armazenado por um milhão.

Um recurso pode fazer várias chamadas

Um único recurso pode produzir várias gerações no provedor.

Um documento longo é dividido em blocos para caber na janela de contexto de um modelo. Cada bloco pode produzir uma chamada separada. A verificação de fatos acrescenta uma ou mais buscas de confirmação. Uma revisão adversarial de artigo coordena várias etapas especializadas e pode usar outro modelo para uma crítica independente.

O Jiddu soma o custo de todas as chamadas da operação e arredonda apenas uma vez:

```
call 1: $0.012
call 2: $0.035
call 3: $0.004
----------------
total:  $0.051
```

A cobrança em créditos se torna:

```
$0.051 × 1.5 = $0.0765

$0.0765 ÷ $0.01 = 7.65

ceil(7.65) = 8 credits
```

Arredondar cada chamada separadamente cobraria demais por qualquer operação formada por muitas chamadas pequenas. Agregar primeiro mantém o cálculo ligado ao custo da ação completa que o usuário solicitou.

Um multiplicador de 1,5 não é uma margem de 50%

Isso confunde porque markup e margem bruta usam denominadores diferentes.

Se o custo do provedor for um dólar:

```
provider cost: $1.00
multiplier:       1.5
billable value: $1.50
gross profit:   $0.50
```

O markup sobre o custo é de 50%:

```
$0.50 ÷ $1.00 = 50%
```

A margem bruta sobre a receita é de 33,3%:

```
$0.50 ÷ $1.50 = 33.3%
```

Uma margem bruta real de 50% exigiria um multiplicador de 2,0 antes de taxas de pagamento, impostos, infraestrutura e outras despesas.

Com as configurações atuais, 1.000 créditos carregam dez dólares de valor faturável interno e cobrem aproximadamente US$ 6,67 de custo do provedor:

```
$10.00 ÷ 1.5 = $6.67
```

Mantive o multiplicador configurável porque um número que parece generoso na fórmula pode ficar muito mais apertado depois que o restante do negócio chega até ele.

Créditos inteiros criam uma cobrança mínima

Os créditos são inteiros. Qualquer custo medido maior que zero consome pelo menos um:

```
provider cost: $0.001
billable cost: $0.0015
raw credits:   0.15
final charge:  1 credit
```

Isso torna o markup efetivo sobre chamadas muito pequenas bem maior que 1,5. O efeito do arredondamento diminui à medida que a operação cresce.

Por enquanto, aceito o piso de um crédito porque créditos fracionários complicariam a interface, o livro-razão e os futuros pacotes. O sistema é proporcional nos tamanhos normais de análise, mas não em chamadas muito pequenas.

Se o uso real produzir muitas operações abaixo de um centavo, posso reduzir a unidade de crédito, agregar cobranças ao longo do tempo ou armazenar unidades internas fracionárias enquanto mostro saldos arredondados ao usuário.

Reservar primeiro, fechar depois

O custo exato só fica disponível quando o provedor responde. Mesmo assim, o Jiddu precisa saber se um usuário pode pagar por uma operação antes de iniciar uma revisão cara.

A tabela de tarifas faz essa verificação antes da execução. O Jiddu estima a cobrança máxima provável com base no recurso, na cadeia de modelos, no número de blocos e nas possíveis chamadas de fallback, e então reserva essa quantidade de créditos na conta.

Uma reserva não altera o saldo, mas reduz o valor disponível para outras operações simultâneas:

```
balance:   100 credits
reserved:   20 credits
available:  80 credits
```

Quando a operação termina, o Jiddu fecha a cobrança com base no uso real.

Se a reserva era de 20 créditos e a cobrança real é sete, o Jiddu debita sete e libera os 13 restantes. Se a cobrança real é 24, debita 24. A reserva é uma verificação de solvência, não um teto de cobrança.

Um estouro pode deixar a conta negativa. Novas reservas ficam bloqueadas até que o saldo seja recarregado. Permitir o estouro é mais preciso do que descartar o custo do provedor porque a estimativa estava errada.

Reservas pendentes expiram depois de duas horas. Isso impede que uma aba do navegador, uma falha de rede ou um trabalho abandonado bloqueie créditos para sempre.

A tabela de tarifas também é um fallback

A OpenRouter normalmente retorna `usage.cost`, mas o sistema de cobrança ainda precisa de um fallback determinístico.

Uma resposta pode conter contagens de tokens e omitir o custo. Uma chamada pode falhar depois que o provedor já executou trabalho. Uma integração pode retornar metadados de uso incompletos. O Jiddu precisa de um resultado determinístico nesses casos.

A tabela de tarifas atual atribui uma estimativa fixa a cada modelo conhecido:

Modelo	Créditos de fallback por chamada
MiniMax M3	1
GPT-5.4 mini	2
Perplexity Sonar Reasoning Pro	2
GPT-5.5	10
Claude Opus 4.8	18
GPT Live Transcribe	20 por sessão
Modelo desconhecido	20

Esses números são versionados, então uma linha do livro-razão pode mostrar qual versão da tabela foi usada mesmo depois que os valores atuais mudarem.

O Jiddu aplica o fallback apenas às chamadas que não podem ser associadas a um custo informado. Considere uma operação com duas chamadas:

```
GPT-5.4 mini
reported cost: $0.012
converted charge: ceil($0.012 × 1.5 ÷ $0.01) = 2 credits

MiniMax M3
reported cost: missing
fallback charge: 1 credit

total: 3 credits
```

Um `cost: 0` explícito é diferente de um custo ausente. Zero significa que a OpenRouter informou que a geração foi gratuita, então o Jiddu não cobra crédito de fallback. Ausente significa que o preço é desconhecido, então a tabela de tarifas se aplica.

A mesma tabela fornece a estimativa de reserva antes da execução. A contabilidade real do provedor substitui essa estimativa quando a operação termina.

Repetições e etapas com falha ainda têm custo

Um modelo pode retornar texto sintaticamente válido que falha na validação de esquema do Jiddu. Uma solicitação pode ser interrompida depois que o provedor já gerou tokens. Um modelo de fallback pode então concluir a mesma etapa.

Uma ação do usuário pode consumir inferência paga em várias tentativas.

Quando os dados de uso do provedor estão disponíveis, o Jiddu registra e cobra cada tentativa, mesmo que o resultado não possa ser usado. A repetição bem-sucedida é registrada separadamente. Isso mantém o livro-razão alinhado com a conta que pagou a cobrança externa.

Essas tentativas com falha também fornecem evidências para melhorar o produto. Se um modelo provoca repetições pagas com frequência, o problema aparece na taxa de conclusão, nos registros de uso e no custo por análise bem-sucedida. Esconder essas chamadas dentro de um preço fixo por recurso ocultaria um problema de engenharia atrás da conta do cliente.

O que o livro-razão guarda

Um saldo sozinho pode dizer ao usuário que créditos desapareceram. Não consegue explicar por quê.

Cada transação de uso registra:

- o recurso e o resultado produzido;
- o débito de créditos e o saldo após o fechamento;
- a estimativa de reserva e qualquer estouro;
- tokens de prompt, conclusão, total, raciocínio e cache;

- custo do provedor e custo faturável em micros;
- o multiplicador de margem e o valor em dólares por crédito;
- identificadores de modelo e geração para cada chamada;
- se a chamada foi concluída;
- modelos de fallback e seus créditos;
- a versão da tabela de tarifas.

Esse registro torna uma cobrança reproduzível. Com o custo do provedor e as configurações de conversão, posso calcular o débito novamente meses depois. Os dados por chamada mostram se a despesa veio de um prompt longo, raciocínio pesado, um crítico caro ou uma repetição.

Créditos concedidos manualmente e débitos de uso permanecem separados. Quando um administrador acrescenta 500 créditos, essa é uma transação. Quando uma análise consome oito, essa é outra. Alterar o saldo diretamente sem registrar os dois eventos tornaria a conciliação impossível.

A transcrição ao vivo é a exceção atual

O modo ao vivo do Jiddu envia áudio do navegador diretamente para a Realtime API da OpenAI usando `gpt-live-transcribe`. Esse tráfego não passa pela OpenRouter, então não existe um `usage.cost` da OpenRouter para fechar.

A implementação atual cobra 20 créditos fixos por uma sessão de transcrição limitada a 60 minutos. As janelas de análise ao vivo continuam usando a OpenRouter e fecham a cobrança com base no uso real, como as outras chamadas de modelo.

O preço fixo da transcrição é conservador. A OpenAI atualmente lista o `gpt-live-transcribe` a US$ 0,017 por minuto, então um cálculo mais proporcional poderia ser:

```
transcription cost = billable duration × $0.017 per minute

credits = ceil(transcription cost × margin multiplier ÷ value per credit)
```

Ainda não implementei esse cálculo. Ele exige um registro confiável no servidor para início, fim, reconexão e duração máxima da sessão. Um cronômetro no navegador seria fácil de manipular e difícil de auditar.

Até que essa medição exista, a tarifa fixa permanece separada e visível, sem fingir que é baseada em uso.

O que acontece quando o Stripe chegar

Os Créditos Jiddu ainda são uma unidade contábil interna. Um crédito representa US$ 0,01 de valor faturável durante o fechamento, mas os usuários ainda não podem comprar um pacote.

Quando o Stripe chegar, o preço de venda e o valor de fechamento devem permanecer separados. Um pacote pode incluir desconto por volume, promoção, imposto ou taxa de pagamento sem alterar quanto uso do provedor um crédito cobre dentro do livro-razão.

Manter esses conceitos separados também impede que uma futura página de preços reescreva o uso histórico. A transação registra o multiplicador, o valor do crédito, o custo do provedor e a versão da tabela de tarifas que valiam quando ela aconteceu.

Antes de escolher pacotes para venda, quero distribuições reais: custo por recurso, custo por resultado bem-sucedido, gasto com repetições, proporção de tokens de raciocínio, estouros de reserva e tamanho do efeito de arredondamento. O livro-razão agora coleta esses números.