Roteiro: CMAB, teste A/B e RSI, em 30 minutos

O bandit contextual, o teste A/B e a automelhoria recursiva num vídeo explicativo, com uma invariante em Z3 ou uma função em Haskell atrás de cada frase

Matheus Hoffmann ORCID

9 de novembro de 2026 · v0, rascunho · rev. f5fd12d

Resumo. Este é o roteiro de um vídeo de 30 minutos, em três blocos de 10 minutos: o bandit contextual de múltiplos braços (CMAB), o teste A/B e a automelhoria recursiva (RSI). Cada cena é um take, com a fala, o que aparece na tela, o gráfico interativo quando ele ajuda, e o código que sustenta a cena. Tudo o que a fala afirma sobre o método tem uma afirmação provada pelo Z3 (`z3-scala/src/main/scala/bandit/domain/Video.scala`, as invariantes R1 a R5; `z3-scala/src/main/scala/bandit/domain/Theorems.scala`, as regras de decisão) ou uma função em Haskell (`haskell/src/Video.hs`, `haskell/src/Pi.hs`, `haskell/app/Video.hs`). Os arquivos estão no repositório h0ffmann/proof. O exemplo simples é o cálculo de π, com o espectador como recompensa; o exemplo complexo é a escolha de imagens da Netflix. O texto segue o português técnico controlado (PTC), nível leve.

Palavras-chave: bandit contextual; teste A/B; automelhoria recursiva; Z3; Haskell; roteiro.

Como ler este roteiro

Padrão de escrita: português técnico controlado (PTC, kayquer/portugues-tecnico-controlado), nível leve, destinatário humano. A fala é em primeira pessoa, porque o vídeo é uma explicação minha. As regras PTC-1 (quem faz, aparece), PTC-7 (número, data e unidade), PTC-8 (ortografia de 1990) e PTC-9 (sem ruído de texto gerado) valem em toda linha. Frase com no máximo 35 palavras. A nota vizinha deste roteiro é `contextual-bandits/README.pt-BR.md`, a página única do bandit (Hoffmann 2026a).

Cada cena é **um take**: um arquivo de áudio (`voice/narrate.py` escreve `scene-N.M.wav`), um arquivo de vídeo (`take-N.M.mp4`) e um corte na edição. Nenhum take depende do anterior; a ordem é a deste documento. Cada cena tem quatro partes. **Fala** é o texto que a voz lê. **Tela** é o que aparece. **Desmos**, quando a cena tem um gráfico que um controle deslizante explica melhor que uma animação pronta: a expressão está em `desmos/`, para colar na calculadora gráfica e gravar a tela. **Código** é o arquivo que sustenta a cena. Tudo o que a fala afirma sobre o método tem uma função em Haskell (`haskell/src/Video.hs`, `haskell/src/Pi.hs`, `haskell/app/Video.hs`) ou uma afirmação provada pelo Z3. As invariantes R1 a R5 dos requisitos do autor estão em `z3-scala/src/main/scala/bandit/domain/Video.scala`; os teoremas das regras de decisão, em `z3-scala/src/main/scala/bandit/domain/Theorems.scala`. Os números citados estão em `results.md`, escrito por `cabal run video`. Python só faz animação (Manim), em `animation/`. Todos os caminhos são relativos à raiz do repositório h0ffmann/proof (Hoffmann 2026b), ou à pasta deste roteiro, `contextual-bandits/script/`.

Duração: 30 minutos, em três blocos de 10 minutos: **CMAB** (0:00 a 10:00), **teste A/B** (10:00 a 20:00) e **RSI** (20:00 a 30:00). A fala tem cerca de 130 palavras por minuto; cada cena traz a duração prevista.

Estilo: o de 3Blue1Brown (Grant Sanderson). Fundo escuro, uma ideia por tela, os objetos se transformam em vez de cortar, texto curto na tela e a fala carrega o argumento. Cores: azul para o que o modelo acredita, amarelo para a escolha, vermelho para o arrependimento. O personagem π de 3b1b não aparece: é dele. As animações são feitas com Manim, a biblioteca que Grant Sanderson criou e que a Manim Community mantém (Manim Community 2026). Os créditos (cena 3.6) nomeiam 3Blue1Brown, Manim, Claude Code, os modelos Claude que fizeram este material, o custo total e a lista de prompts (`PROMPTS.md`).

Marcadores: `[a confirmar: …]` é uma informação que o original não deu e que eu não inventei. ⚠ é uma afirmação que eu não confirmei numa fonte primária.

Voz

A narração é sintetizada, em português do Brasil, e depois recebe o timbre de uma voz de referência com o OpenVoice. Dois fatos limitam a montagem:

- O OpenVoice V2 sintetiza, por si, só inglês, espanhol, francês, chinês, japonês e coreano (MyShell AI 2024b). O gerador de base que ele usa, o MeloTTS, não tem português (MyShell AI 2024a).
- O OpenVoice separa a fala em duas partes: um gerador de base, que diz as palavras, e um conversor de timbre, que troca a cor da voz. A entrada do conversor "pode estar em qualquer idioma" (MyShell AI 2024b). Então a base vem de outro gerador, com português do Brasil, e o OpenVoice só clona o timbre.

A base é o XTTS v2 (Coqui), com `language="pt"`; ele aceita 17 idiomas e português é um deles (Coqui 2024). Alternativa mais leve: a voz `pt_BR-faber-medium` do Piper (Rhasspy 2023). ⚠ Eu não listei as outras vozes pt-BR do Piper.

A voz de referência é a minha, gravada por mim: 30 s de leitura limpa, sem música, em `voice/reference.wav`. `[a confirmar: quer a sua voz, ou uma voz de locutor neutra? Uma voz de pessoa real pede autorização dela.]`

Código. `voice/narrate.py` lê as cenas deste roteiro (os arquivos de `prose-pt-BR/`), sintetiza cada **Fala** com o XTTS v2 e passa o áudio pelo conversor de timbre do OpenVoice. `voice/requirements.txt` lista os pacotes. Eu não executei esse arquivo aqui: os modelos pesam alguns GB e pedem GPU.

1 CMAB (0:00 a 10:00)

1.1 Gancho (0:00 a 1:00)

Fala. Todo agente que aprende precisa de um mecanismo de decisão com estrutura de recompensa. Ele tem uma crença de hoje: "esta opção agrada em 67 % das vezes". Aí chega um sinal novo do usuário: gostou, não gostou. A pergunta é sempre a mesma: a crença nova ainda escolhe a mesma coisa que a crença de hoje? Se sim, nada muda. Se não, o agente muda, e muda por evidência, não por palpite. Esse mecanismo tem nome: bandit contextual de múltiplos braços, CMAB. Ele é velho, de 1952, e está no centro de um assunto que começou a ser padronizado em 2026: a automelhoria recursiva, RSI. Num ambiente real você sempre vai querer comparar a sua crença atual com uma crença nova, a do usuário. Hoje as interfaces agênticas já leem esse sinal direto da conversa, e fazem coisas que não cabem neste vídeo. ⚠ `[a confirmar: você citou "Side Agents" da Anthropic;`

`eu não achei uma fonte primária com esse nome. Qual é a referência?]` Neste vídeo cabem três coisas: o CMAB, o teste A/B e a RSI. Dez minutos para cada, com código para cada frase.

Tela. Duas barras: "crença atual" e "crença nova", com a seta de escolha mudando de uma opção para a outra. Em baixo, os três títulos dos blocos.

Código. `haskell/app/Video.hs`, `beliefs`, com a regra de `betaBernoulli` em `haskell/src/Models.hs`. A saída em `results.md`: a crença atual escolhe a opção 0 (0,67); depois de 4 sinais positivos para a opção 1 e 2 negativos para a 0, a crença nova escolhe a opção 1 (0,62). "Mudou de ideia? True". A invariante R1 em Z3: `by_distance`, a recompensa é o input do usuário, em [0, 1].

1.2 Steal like an artist: de onde eu tirei isso (1:00 a 3:00)

Fala. Nada aqui é meu. Austin Kleon escreveu em 2012, em *Steal Like an Artist*: "nothing is original… all creative work builds on what came before" (Wikipedia contributors 2026c). Esta é a minha linha do tempo com essa ideia. Em 2018 eu ouvi falar de bandit pela primeira vez, de um amigo do trabalho. `[a confirmar: quem, e em que mês de 2018?]` Em dezembro de 2017 a Netflix tinha publicado "Artwork Personalization at Netflix": a imagem de cada título, escolhida para cada assinante por um bandit contextual (Dezeen 2017; Hoffmann 2026a). Eu li depois. Em 2018, Renan Moreira de Oliveira, engenheiro de dados da globo.com, apresentou "Machine Learning na globo.com", com um slide chamado "MAB ML PARA AB RL" (Oliveira 2018). `[a confirmar: ele foi para a Uber? Eu não encontrei a confirmação.]` Tiago Motta, também da globo.com, conta que o serviço de bandit deles nasceu num hackday em 2017, em Scala, inspirado no livro de John Myles White, e que ele "ainda é usado" no portal. Ele publicou isso em 2024 (Motta 2024). E em 2026 o bandit aparece dentro dos ciclos de automelhoria dos agentes: o laço que decide o que mudar no próprio agente é um laço de escolha com recompensa parcial. Essa é a parte nova. O resto é roubo bem creditado. A linha do tempo tem um buraco de 58 anos entre Robbins, em 1952 (Wikipedia contributors 2026b), e o LinUCB, em 2010 (Li et al. 2010). Nesse buraco o problema ficou na estatística e na economia; a internet deu a ele milhões de rodadas por dia, e aí ele virou engenharia. Reparem na ordem: a teoria veio primeiro, a Netflix depois, a globo.com quase junto, e eu por último. Nenhum desses passos inventou o anterior. Cada um pegou o anterior e mudou uma coisa: o contexto, a escala, a infraestrutura, e agora o alvo do laço.

Tela. Linha do tempo animada (Manim), um marco por vez: 1952 Robbins formula o problema · 2010 LinUCB (Li, Chu, Langford, Schapire) · 2012 *Steal Like an Artist* · 2017-12 Netflix · 2017 hackday da globo.com · 2018 o amigo, e os slides da globo.com · 2024 o post de Tiago Motta · 2026 RSI. Cada marco entra com a fonte em letra pequena.

Código. `animation/timeline.py`, uma cena Manim com os marcos acima, a fazer depois da aprovação deste roteiro. As fontes estão na lista de referências.

1.3 O método de escolha, no exemplo simples: π e você, um braço só (3:00 a 5:00)

Fala. Vamos começar pelo exemplo mais simples que eu consegui, e com você no meio dele. Dois algoritmos calculam π. O de Leibniz, de 1674: π sobre 4 é 1, menos um terço, mais um quinto, menos um sétimo, e assim por diante. Bonito, e lento: com 100.000 termos ele acerta 5 dígitos. O de Ramanujan, de 1914, o matemático indiano: 1 sobre π é uma soma em que cada termo tem fatoriais e a constante 26.390. Um termo só acerta 7 dígitos; seis termos

acertam 48. Guardem Ramanujan para a próxima cena. Agora só Leibniz, um braço só: o *single-armed bandit*. O agente tem uma aproximação, π de N, que você já viu e já avaliou. A cada rodada ele decide uma de duas coisas. Aproveitar: mostrar π de N de novo, e receber a mesma resposta de sempre. Ou explorar: gastar um termo, calcular π de N mais 1, mostrar, e ouvir o que você diz. Você responde: está bom? Com quantos dígitos? Essa resposta é a recompensa. O agente não sabe matemática. Ele não lê a fórmula. Ele só sabe o que você disse da aproximação que ele mostrou. Esse é o dilema inteiro, com um braço só: ficar com o que se tem, ou pagar para saber mais. Na simulação, o usuário dá uma nota pela distância até π, de 0 a 1. Em 20 rodadas o agente aproveitou 9 vezes e explorou 11, e chegou a π de 12; a nota média de explorar foi 0,76, a de aproveitar 0,14. Faz sentido: com Leibniz, cada termo novo ainda agrada um pouco mais, e explorar continua pagando. No modo interativo, você é o usuário, e a tabela é a sua. O ciclo é sempre o mesmo: decidir, mostrar, receber a sua resposta, atualizar a crença, decidir de novo. A regra que decide é a política.

Tela. Uma soma de Leibniz crescendo, estilo 3b1b, oscilando em volta de π. Dois botões: "π(N)" e "π(N+1)". Um balão "está bom?" com a resposta do usuário entrando como número. Abaixo, as duas barras de crença do agente, azul, uma por ação, e a seta amarela da escolha.

Desmos. `desmos/1-3-pi.txt`: a soma parcial `L(n)` contra a reta `y = π`, e os dígitos corretos `-log10|L(n) - π|`; o controle `N` anda termo a termo. A cena 1.4 usa o mesmo arquivo com `R(n)`, de Ramanujan.

Código. Z3, `z3-scala/src/main/scala/bandit/domain/Video.scala`: `step`, e as afirmações `step_exploit` (aproveitar não gasta termo), `step_explore_n` (explorar gasta exatamente um), `step_explore_reward` (a nota não cai) e `step_reward_bounded` (a nota fica em [0, 1]). `leibnizPair_pos`: cada par de Leibniz é positivo, logo as somas pares crescem. Haskell, `haskell/src/Video.hs` (os mesmos nomes) e `haskell/src/Pi.hs` (`leibnizS`, `digitsCorrect`); a simulação é `singleArmed` em `haskell/app/Video.hs`. A tabela de dígitos por termo e a de 20 rodadas estão em `results.md`.

1.4 O nome, letra por letra: de um braço a vários (5:00 a 6:30)

Fala. B de bandit. A máquina caça-níquel é o "one-armed bandit", o bandido de um braço só, porque ela leva o seu dinheiro. M e A de *multi-armed*: agora são várias alavancas lado a lado, cada uma com um prêmio médio que ninguém conhece. Herbert Robbins formulou essa versão em 1952 (Wikipedia contributors 2026b). No nosso exemplo entram mais duas alavancas. A de Gauss, o príncipe dos matemáticos: o algoritmo de Gauss-Legendre, pela média aritmético-geométrica, dobra os dígitos a cada passo: 3, 8, 19, 41. E a de Ramanujan, o matemático indiano, que eu chamo de rei: 7 dígitos por termo, 48 em seis. Agora explorar não é só ir mais fundo em Leibniz: é puxar outra alavanca para ver o que ela rende. Na simulação, com o mesmo usuário, em 20 rodadas o agente puxou Ramanujan 16 vezes, Gauss-Legendre 3 e Leibniz 1, e terminou com 61, 19 e 0 dígitos certos. Ele não leu as fórmulas; ele ouviu você. Uma observação honesta: com nota pela distância, Gauss-Legendre e Ramanujan empatam em 1,00 depois do primeiro termo; o agente ficou com Ramanujan porque a regra gulosa desempata pelo primeiro máximo. Para separar os dois, a nota teria de ser por dígito. C de contextual: antes de puxar, você vê um dado da rodada, por exemplo manhã ou noite. A Wikipedia chama isso de "a useful generalization of the multi-armed bandit" (Wikipedia contributors 2026b). No código, cada letra é um tipo: M e A são a lista de braços, `Arm`, e C é o tipo do contexto `c` em `RewardModel c b`. Esse é o ambiente, o que paga. A escolha fica em outro arquivo, `DecisionRule`, de propósito. O exemplo de π não tem contexto, por isso ele é um MAB e não um CMAB. Uma observação sobre o nome: a palavra bandit assusta, mas ela só diz que a máquina está contra você. A palavra que importa é contextual.

Sem contexto, a melhor alavanca é uma só para o mundo inteiro. Com contexto, a melhor alavanca muda a cada rodada, e a política precisa aprender uma função: do contexto para a alavanca. No LinUCB o contexto é um vetor de números, e a função é linear (Li et al. 2010). Na sorveteria o contexto tem dois valores, manhã e noite, e a função é uma tabela.

Tela. As letras C, M, A, B aparecem uma a uma; cada uma vira uma peça do desenho: uma alavanca (Leibniz), três alavancas (entram Gauss-Legendre e Ramanujan), um relógio ao lado. A soma de Ramanujan cola em π no primeiro termo; a de Gauss-Legendre dobra os dígitos a cada passo; a de Leibniz oscila.

Código. `haskell/src/Pi.hs`: `leibnizS`, `gaussLegendreS`, `ramanujanS`, `sqrtRat`; a simulação é `multiArmed` em `haskell/app/Video.hs`. Z3: `ramTerm_pos` e `ramSum_lt_succ`, cada termo de Ramanujan é positivo e as somas parciais crescem; `ramApprox_anti`, a aproximação desce para π por cima. O ambiente com contexto do bloco 2 é `rate` e `shop` em `haskell/app/Video.hs`.

1.5 Modelo e regra, separados (6:30 a 7:30)

Fala. O repositório divide o agente em duas camadas. O modelo de recompensa guarda uma crença e, para cada sabor, dá duas coisas: a média prevista e uma largura, a incerteza. A regra de decisão vê só essas estimativas. Ela nunca vê o histórico nem o contexto. Essa fronteira é um teorema do repositório, `policy_factors`: dois históricos com a mesma crença dão a mesma decisão. Por isso você troca o modelo sem mexer na regra, e troca a regra sem mexer no modelo. O modelo deste vídeo é tabular: uma soma e uma contagem por sabor e por contexto; a média é soma sobre contagem; a largura é 1 sobre contagem mais 1. Em Haskell o repositório tem três modelos: Beta-Bernoulli, regressão linear bayesiana, e um invólucro para qualquer modelo que você já tenha, desde que ele devolva média e desvio. Regra de decisão é a mesma para os três. Isso é o que a fronteira compra.

Tela. Duas caixas: "modelo" à esquerda, "regra" à direita, ligadas só por uma seta "(média, largura) por sabor". O histórico entra só na caixa da esquerda.

Desmos. `desmos/1-5-width.txt`: a largura `1/(n+1)` e a largura real `1/sqrt(n)` no mesmo gráfico, com `n` de 1 a 100. As duas caem; a do Z3 é racional, para os exemplos serem exatos.

Código. Z3: `z3-scala/src/main/scala/bandit/domain/Bandit.scala` (as duas camadas, `estimates` e as regras), `z3-scala/src/main/scala/bandit/domain/Examples.scala` (`update`, `estimate`, `posterior`, `policy`) e `z3-scala/src/main/scala/bandit/domain/Theorems.scala` (`policy_factors`). Haskell: `haskell/src/ContextualBandit.hs`, os mesmos nomes; o modelo tabular do vídeo é `tabular` em `haskell/app/Video.hs`.

1.6 Explorar e aproveitar (7:30 a 8:30)

Fala. Aproveitar é escolher o sabor com a maior média. Explorar é escolher outro, para aprender. Quem só aproveita pode ficar preso: depois de três clientes, a média de um sabor bom pode estar baixa demais, e ele nunca mais sai da gaveta. Quem só explora perde venda nos sabores ruins. Três regras equilibram isso. A gulosa não explora. A ε-gulosa sorteia um sabor qualquer com probabilidade ε e, no resto, é gulosa. A UCB soma à média um bônus de incerteza, β vezes a largura: ela é otimista com o que conhece pouco. Com β igual a zero, a UCB vira a gulosa. O repositório prova isso: `ucb_zero`. Há uma quarta regra, a amostragem de Thompson: em vez da média, ela sorteia uma média plausível da crença de cada sabor e escolhe a maior. Um sabor com crença larga sorteia valores altos de vez em quando, e por isso é explorado. No repositório ela está em Haskell, `thompsonArm`, porque precisa de um modelo que saiba sortear.

Tela. Três colunas com as mesmas estimativas; a seta de cada regra aponta para um sabor; a da UCB sobe a barra da incerteza.

Desmos. `desmos/1-6-ucb.txt`: três pontos (média, largura) e a pontuação `m + b·w` de cada um; o controle `b` vai de 0 a 3 e o sabor vencedor troca na tela.

Código. Z3: `greedy`, `epsGreedy`, `ucb` em `z3-scala/src/main/scala/bandit/domain/Bandit.scala`, `ucb_zero` em `z3-scala/src/main/scala/bandit/domain/Theorems.scala`. Haskell: os mesmos nomes em `haskell/src/ContextualBandit.hs`; o spec check de `haskell/app/Main.hs` confere os exemplos de `z3-scala/src/main/scala/bandit/domain/Examples.scala` `[0, 1, 0]` de manhã, `[0, 0, 1]` de noite e `[1/30, 14/15, 1/30]` para $\varepsilon = 1/10$.

1.7 O exemplo complexo: a Netflix (8:30 a 10:00)

Fala. Do exemplo simples para o complexo. Em dezembro de 2017 a Netflix publicou "Artwork Personalization at Netflix" (Dezeen 2017; Hoffmann 2026a). O problema: para cada título, há várias imagens de capa; qual mostrar para cada assinante? Monte as peças. O contexto é o histórico do assinante e o país. Os braços são as imagens. A recompensa é a reprodução: o assinante deu play ou não. Tony Jebara contou, em 2018, que o sistema recebe cerca de 20 milhões de pedidos de imagem por segundo no pico (Jebara 2018). É o mesmo ciclo do π, com três diferenças. O contexto existe, e é grande. A recompensa não é uma pessoa dizendo "está bom": é uma pessoa agindo, sem saber que está respondendo. E a escala obriga o modelo a generalizar: não dá para ter uma tabela por assinante. No exemplo de π você era a recompensa e sabia disso. Na Netflix, você é a recompensa e não sabe. A estrutura de decisão é a mesma. O que muda é de onde vem o sinal, e esse é o fio que leva até o bloco 3.

Tela. Um título, três capas. Um assinante, o histórico dele como uma fita de ícones, o país. A seta amarela escolhe uma capa; o play aparece ou não; a barra azul daquela capa, para aquele perfil, sobe ou desce. Em canto, "20 M pedidos/s".

Código. O modelo que generaliza é `bayesLinear` em `haskell/src/Models.hs`: a média é um produto escalar entre o contexto e os pesos do braço. O modelo tabular do vídeo (`haskell/app/Video.hs`) é a mesma interface com um modelo menor. A Netflix: a nota `contextual-bandits/README.pt-BR.md`, seção 8, parágrafo 1.7 (Hoffmann 2026a).

2 Teste A/B (10:00 a 20:00)

2.1 O que é um teste A/B (10:00 a 11:30)

Fala. A Wikipedia define: "'A/B testing' is a shorthand for a simple randomized controlled experiment" (Wikipedia contributors 2026a). Você tem K variantes. Cada visitante cai numa variante por sorteio, com a mesma probabilidade. Depois de um tempo, a equipe olha os números, escolhe a melhor variante e manda todo mundo para ela. Duas fases: sortear, depois decidir. Em inglês, *explore-then-commit*. O teste A/B tem três virtudes que explicam por que ele dominou a web por vinte anos. Ele é simples de montar: um sorteio e uma contagem. Ele é simples de ler: a diferença entre as médias é o efeito. E ele é simples de defender: a alocação é fixa, e ninguém mexeu nela durante o teste. Guardem essas três virtudes. O bandit troca as duas primeiras por velocidade e paga na terceira.

Tela. Uma fila de visitantes dividida em três caminhos iguais; no fim, um portão fecha dois caminhos.

Desmos. `desmos/2-1-ab.txt`: duas Bernoulli com `p_A` e `p_B` em controles; `n` visitantes por variante; o intervalo de confiança das duas médias encolhe quando `n` cresce.

Código. Z3: `uniform(K)` em `z3-scala/src/main/scala/bandit/domain/Bandit.scala`; a invariante R3, `etc` em `z3-scala/src/main/scala/bandit/domain/Video.scala`, com `etc_before`: antes da troca, a alocação é `uniform`. Haskell: `etc` em `haskell/src/Video.hs`.

2.2 O teste A/B é uma regra de bandit (11:30 a 13:00)

Fala. Agora a ponte. A ε-gulosa com ε igual a 1 ignora o modelo e sorteia cada sabor com probabilidade 1 sobre K. Isso é o período de teste de um teste A/B/n. Não é uma analogia: é o teorema `epsGreedy_one_is_ab_test`, provado no repositório. E a segunda fase, decidir e ficar na melhor, é a ε-gulosa com ε igual a 0, a gulosa. Então um teste A/B é uma política de bandit com um ε que pula de 1 para 0 num dia marcado. Eu gosto desse jeito de ver, porque ele transforma uma discussão de time, "A/B ou bandit?", numa escolha de um número: como ε deve cair ao longo do tempo. De 1 para 0 num salto, é o teste A/B. Constante em 0,1, é a ε-gulosa. Caindo devagar, é um esquema intermediário. A pergunta deixa de ser qual ferramenta e passa a ser qual curva.

Tela. Um controle deslizante de ε. Em 1, as três barras iguais. Em 0, uma barra só. O título: "teste A/B = ε de 1 para 0".

Desmos. `desmos/2-2-epsilon.txt`: a mistura `e/K + (1-e)·[a = melhor]` para três braços; o controle `e` de 0 a 1 é a cena inteira.

Código. Z3: `epsGreedy_one_is_ab_test` ($\varepsilon = 1$ é o sorteio uniforme) em `z3-scala/src/main/scala/bandi`
`epsGreedy_zero` ($\varepsilon = 0$ é a gulosa), `etc_after` e `etc_valid` em `z3-scala/src/main/scala/bandit/domain/`
Haskell: `etc 10 3` dá `[1/3, 1/3, 1/3]` e `etc 10 10` dá `[0, 1, 0]`, conferidos contra os exemplos do Z3 no spec check.

2.3 O custo da espera (13:00 a 15:00)

Fala. O que o teste A/B custa? Durante toda a fase de sorteio, dois terços dos clientes veem um sabor pior. Esse custo tem nome: arrependimento. É a soma, rodada a rodada, da média do melhor sabor menos a média do sabor escolhido. Tony Jebara, da Netflix, chamou o ciclo "coletar dados por meses, treinar, depois testar" de "a massive amount of regret" (Jebara 2018). Para medir, eu simulei uma sorveteria: três sabores, uma vitrine com espaço para um, e taxas secretas, as do repositório. De manhã, o sabor 1 vende para 12 % dos clientes; de noite, o sabor 2 vende para 15 %. A política não conhece essas taxas; eu conheço porque escrevi a simulação. Com 5.000 clientes, metade na fase de sorteio, o teste A/B acumula 129 de arrependimento. A ε-gulosa com ε igual a 0,1 acumula 76. A UCB com β igual a 2 acumula 20. Mesma sorveteria, mesmo modelo, só a regra mudou. Por que o teste A/B custa tanto? Porque ele é cego durante metade do tempo, por decisão. Na rodada 100 ele já sabe que o sabor 0 vende mal de manhã, e continua mandando um terço dos clientes para ele até a rodada 2.500. A ε-gulosa manda só um trinta avos. A UCB manda enquanto a largura daquele sabor ainda é grande, e para quando a largura encolhe. A diferença de 129 para 20 é a diferença entre ignorar e usar a informação que já chegou.

Tela. Quatro curvas de arrependimento acumulado subindo; a do teste A/B sobe reta até a metade e depois quase para; a da UCB fica embaixo.

Desmos. `desmos/2-3-regret.txt`: o arrependimento esperado do teste A/B em forma fechada, `(T/2)·g + 0`, contra o da ε-gulosa, `T·e·g`, com `g` o salto médio e `e` em controle; mostra em que ε as duas retas se cruzam.

Código. `abVsBandit` em `haskell/app/Video.hs`, média de 20 sementes (`results.md`); a sorveteria é `rate` e `shop`, com as taxas de `haskell/app/Main.hs` (`ctr`). A Netflix: a nota do repositório, seção 4.1 (Hoffmann 2026a), e a palestra de Jebara (Jebara 2018).

2.4 O contexto personaliza o teste (15:00 a 16:30)

Fala. Um teste A/B acha uma melhor variante para todo mundo. Mas na sorveteria o melhor sabor de manhã não é o melhor de noite. Um teste A/B que ignora a hora escolhe um sabor só e perde dos dois lados: 179 de arrependimento, contra 129 do teste que separa manhã e noite. Um teste A/B por segmento é um caso particular de bandit contextual. O bandit contextual acha a melhor variante para cada contexto, e o contexto pode ser um vetor com dezenas de dimensões, como em Li, Chu, Langford e Schapire, em 2010, no LinUCB (Li et al. 2010). Com um vetor, não dá mais para ter uma tabela por contexto: há contextos demais, e cada um aparece uma vez. O modelo precisa generalizar entre contextos parecidos. É aí que entra a regressão: a média do sabor é um produto escalar entre o contexto e um vetor de pesos do sabor, e a largura vem da covariância desses pesos. A regra de decisão não muda uma linha.

Tela. Duas vitrines, manhã e noite, com o sabor vencedor diferente em cada uma. A linha "sem contexto" com um sabor só nas duas.

Código. `abVsBandit` em `haskell/app/Video.hs`, política "A/B sem contexto": o mesmo `etc` sobre um modelo com um único contexto `()`.

2.5 O que o teste A/B ainda faz melhor (16:30 a 18:00)

Fala. Então o teste A/B morreu? Não. A alocação fixa e aleatória dá uma estimativa simples e sem viés do efeito. O bandit muda a alocação enquanto aprende; a média simples dos registros fica com viés. Para avaliar um bandit *offline*, você registra a probabilidade de cada escolha e usa *replay* ou ponderação pelo inverso da propensão. Li e colegas argumentam, em 2010, que "any bandit algorithm can be reliably evaluated offline using previously recorded random traffic" (Li et al. 2010). A Netflix ainda aprovou o sistema de imagens com um teste A/B antes de ligar para todos (Jebara 2018). O bandit substituiu o ciclo lento dentro do teste, não o teste. O *replay* funciona assim: você tem um registro de rodadas com escolhas uniformes. Para avaliar uma política nova, você percorre o registro; quando a política nova escolhe o mesmo braço que o registro, a rodada conta; quando não, a rodada é descartada. A média das rodadas que contaram é uma estimativa sem viés do que a política nova faria. O preço é jogar fora K menos 1 de cada K rodadas.

Tela. Dois gráficos: "alocação fixa, média sem viés" e "alocação que muda, média com viés", e a correção por propensão.

Desmos. `desmos/2-5-propensity.txt`: a média simples e a média ponderada por `1/p` de uma amostra em que o braço bom foi escolhido com probabilidade `p`; o controle `p` mostra o viés aparecer e a correção o desfazer.

Código. `policy` em `haskell/src/ContextualBandit.hs` (e em `z3-scala/src/main/scala/bandit/domain`) devolve a distribuição inteira, não só o braço: essa distribuição é a propensão que o *replay* precisa registrar.

2.6 Resumo do bloco (18:00 a 20:00)

Fala. O teste A/B é um bandit com três escolhas fixas: sem contexto, alocação uniforme, e uma decisão só, no fim. O bandit contextual relaxa as três. Ele usa o contexto, muda a alocação a cada rodada e decide sempre. O preço é a avaliação *offline*, que fica mais difícil. No Brasil, a globo.com usa um serviço de bandit desde 2017 para "escolher uma foto" e "a melhor posição para colocar publicidade", nas palavras de Tiago Motta (Motta 2024). Na próxima parte, o bandit sai do site e entra no agente. Antes, uma regra prática. Use o teste A/B quando a decisão é uma só e precisa ser defendida: um preço, uma mudança de marca, algo que vai para um comitê. Use o bandit quando a decisão se repete milhares de vezes por dia e o custo de errar em cada uma é pequeno: uma imagem, uma posição, uma frase de abertura. E use os dois quando puder: o bandit por dentro, o teste A/B por fora, como a Netflix. Essa combinação é a que a maioria das equipes acaba adotando, porque ela fica com a velocidade de um e a defesa do outro. E para fechar o bloco, cinco fatos sobre tudo isso estão provados pelo Z3, por refutação. Toda regra devolve uma distribuição de probabilidade. A ε-gulosa dá a cada braço pelo menos ε sobre K. A gulosa e a UCB escolhem um máximo de verdade. A ε-gulosa com ε igual a 1 é o teste A/B. E o lema do otimismo, `ucb_gap`: se o modelo é calibrado, o braço da UCB perde no máximo 2 β vezes a largura para o melhor. A parte probabilística da cota de arrependimento está fora do que o Z3 decide e fica aberta. Eu digo isso para separar o que este vídeo mostra, a parte determinística e executável, do que ele não mostra.

Tela. Tabela de três linhas: contexto, alocação, decisão; coluna A/B e coluna CMAB. Depois, a tabela de teoremas do README, uma linha por vez.

Código. A tabela é o parágrafo 2.6 da nota `contextual-bandits/README.pt-BR.md` (Hoffmann 2026a). Os teoremas estão em `z3-scala/src/main/scala/bandit/domain/Theorems.scala`, provados para até cinco braços e todo valor real das estimativas. O Python do Manim não prova nada; ele só desenha o que o Z3 provou.

3 RSI, automelhoria recursiva (20:00 a 30:00)

3.1 Três palavras, uma por vez (20:00 a 21:30)

Fala. *Improvement*: o agente escolhe melhor hoje do que ontem. *Self-improvement*: o próprio agente faz essa mudança, sem ninguém de fora ajustar nada. *Recursive self-improvement*: a mudança é no mecanismo que faz as mudanças. A recursão é um dos conceitos mais interessantes da natureza, e do próprio ser humano: nós aprendemos a aprender. A Anthropic define, no ensaio "When AI builds itself", um sistema que "could fully autonomously design and develop its successor", e diz: "This is called recursive self-improvement" (Anthropic Institute 2026). Eles também dizem que ainda não chegaram lá e que isso "is not inevitable". Eu separo as três palavras porque elas são três exigências diferentes. Melhoria exige uma métrica: sem ela, mudar não é melhorar. Automelhoria exige um laço fechado: a métrica volta para dentro do agente, sem uma pessoa no meio. Recursão exige que o laço tenha acesso ao próprio laço: o que ele muda inclui o modo de mudar. Cada exigência tem um lugar no código que vem a seguir.

Tela. Três círculos concêntricos: melhoria, automelhoria, automelhoria recursiva. Cada palavra acende um anel.

Código. Z3, `z3-scala/src/main/scala/bandit/domain/Video.scala`, invariante R4: `outer` e `innerEps`, a mesma `epsGreedy` nos dois níveis; `rsi_valid` e `outer_explores`. Haskell: `outer` e `inner` em `haskell/src/Video.hs`.

3.2 O menor ciclo recursivo que cabe numa tela (21:30 a 24:00)

Fala. Vamos construir o menor exemplo honesto. O agente de dentro é a ε-gulosa da sorveteria. O agente de fora é outro bandit, e os braços dele são valores de ε: 1, 0,5, 0,1 e 0,01. A cada episódio, o de fora escolhe um ε, o de dentro vende 500 sorvetes com esse ε, e o ganho médio volta como recompensa para o de fora. Depois de 200 episódios, o de fora escolheu ε igual a 0,5 em 168 deles, com ganho médio de 0,099 por cliente. O ε igual a 0,01 ficou com 9 episódios e 0,057. O agente de fora aprendeu como o de dentro deve aprender. Nada aqui é novo: é a mesma `epsGreedy` aplicada duas vezes. Esse é o ponto. Agora as três exigências. A métrica é o ganho médio por cliente. O laço fechado é o `update` do agente de fora, alimentado pelo resultado do de dentro, sem ninguém escolher o ε à mão. E a recursão: o de fora muda o parâmetro que define como o de dentro explora. Dá para subir mais um nível, com um terceiro bandit escolhendo o ε do segundo. Em algum nível isso para de valer a pena, porque cada nível custa episódios inteiros do nível de baixo. Essa conta, quanto custa aprender a aprender, é o problema central da RSI. Um detalhe do resultado: ε igual a 0,5 ganhou, e não 0,1. Com 500 clientes por episódio e um modelo que começa do zero, explorar muito ainda compensa. Com 50.000 clientes, o vencedor seria outro. O agente de fora aprende o ε certo para o horizonte que ele tem. É por isso que ninguém fixa o ε de cabeça.

Tela. Dois ciclos desenhados, o pequeno dentro do grande. A tabela de ε aparece preenchida ao vivo.

Código. `rsiLoop` em `haskell/app/Video.hs`: `episode` é o agente de dentro, o laço com `outer 0.2` é o de fora. A tabela está em `results.md`.

3.3 Onde o bandit entra na RSI de verdade (24:00 a 26:30)

Fala. Nos sistemas reais, o laço de fora escolhe entre coisas maiores que um ε. Lembrem do π: lá o agente escolhia entre duas fórmulas e a recompensa era você. Aqui o agente escolhe entre mudanças em si mesmo. O artigo SIA, de 2026, descreve um "Feedback-Agent" que a cada passo decide entre editar o *scaffold*, os prompts e as ferramentas, ou atualizar os pesos do modelo (Bhavsar 2026). O "Feedback-Agent" também escolhe entre algoritmos como PPO, GRPO e DPO (Bhavsar 2026). A recompensa vem de um avaliador determinístico. Quando a recompensa estaciona, ele troca de tipo de mudança. Isso é uma escolha entre braços com uma recompensa observada só para o braço escolhido: a estrutura de um bandit, mesmo quando o artigo não usa a palavra. A Anthropic relata, no mesmo ensaio, laços de experimento com objetivo fixo. Num deles, a aceleração de um código de treino foi de cerca de 3 vezes, em maio de 2025, para cerca de 52 vezes, em abril de 2026 (Anthropic Institute 2026). E diz que "direction-setting was the only meaningful role a human played" num estudo de pesquisa aberta. Vamos olhar a estrutura de recompensa com cuidado, porque é nela que a RSI vive ou morre. No SIA, a recompensa é um avaliador determinístico: um teste que passa ou falha, um tempo que cai ou não. Isso é o que torna o laço fechado possível. Se a recompensa fosse uma opinião humana a cada passo, o laço teria uma pessoa dentro e deixaria de ser automelhoria. Mas um avaliador determinístico é um alvo fixo, e um alvo fixo pode ser mirado. A Anthropic relata uma revisão automática de código que teria pegado cerca de um terço dos erros por trás de incidentes passados (Anthropic Institute 2026). Um terço. Os outros dois terços são o lembrete de que a recompensa mede uma parte do que importa, não o todo.

Tela. O diagrama do laço de fora: braços "editar harness", "atualizar pesos"; a recompensa voltando de um avaliador.

Código. A estrutura é a de `rsiLoop` em `haskell/app/Video.hs`; os braços mudam de nome. ⚠ Eu li o SIA por uma resenha (Bhavsar 2026), não pelo PDF.

3.4 Crença atual contra crença nova, de novo (26:30 a 28:00)

Fala. Voltamos ao gancho, e ao π. No exemplo de π a recompensa era o seu input, explícito. Em produção, você sempre vai querer comparar a sua crença atual com uma crença nova, a do usuário. O modelo Beta-Bernoulli faz isso com duas contagens por opção: sucessos e fracassos. Cada sinal do usuário soma 1 numa delas. A crença atual dizia 0,67 para a resposta curta e 0,50 para a longa. Quatro sinais a favor da longa e dois contra a curta, e a crença nova diz 0,57 e 0,62. A escolha mudou. Hoje as interfaces agênticas conseguem ler esse sinal direto da conversa. É isso que faz o CMAB ser o mecanismo certo para 2026: ele foi feito para decidir com recompensa parcial e para mudar de ideia com evidência. Dois detalhes do exemplo. Primeiro: a crença nova não apagou a antiga. Ela somou seis sinais a doze. Com a crença antiga mais forte, a mesma evidência não mudaria a escolha. Isso é o que impede o agente de virar com cada vento. Segundo: a largura caiu dos dois lados. Mais evidência, menos incerteza, mesmo quando a média vai na direção contrária. A UCB usa essa largura; a gulosa não. Para um agente que conversa com uma pessoa, eu prefiro a UCB: ela tenta a resposta longa algumas vezes antes de desistir dela.

Tela. As duas barras do gancho, agora com as contagens α e β visíveis mudando a cada sinal.

Desmos. `desmos/3-4-beta.txt`: as duas densidades Beta(α, β) das opções; os controles `s` (sucessos) e `f` (fracassos) da opção 1 movem a curva até a média dela passar a da opção 0.

Código. `beliefs` em `haskell/app/Video.hs`, a mesma regra de `betaBernoulli` em `haskell/src/Models.hs`.

3.5 O que não é bandit, e o que pode dar errado (28:00 a 29:00)

Fala. Dois limites. Primeiro: o bandit vê a recompensa de uma rodada e não planeja a próxima. Quando a escolha de hoje muda o estado de amanhã, o problema é de aprendizado por reforço completo, e o bandit é o caso de um passo. Segundo: a estrutura de recompensa é o ponto mais frágil. Se a recompensa for fácil de enganar, o laço aprende a enganar a recompensa. O repositório tem um lema que diz o que o modelo deve à regra: incerteza calibrada. Sem isso, o otimismo da UCB vira chute.

Tela. Um aviso simples: "recompensa de um passo" e "recompensa fácil de enganar", com o lema `ucb_gap` ao lado.

Código. `z3-scala/src/main/scala/bandit/domain/Theorems.scala`, as premissas `calibrated` e `ucb_gap`. ⚠ A frase sobre enganar a recompensa é um resumo meu; o roteiro não cita um trabalho específico de *reward hacking*.

3.6 Fecho e créditos (29:00 a 30:00)

Fala. Resumo em três frases. Um CMAB é uma regra de decisão em cima de um modelo probabilístico, que escolhe com recompensa parcial. Um teste A/B é um CMAB com três coisas congeladas: sem contexto, 1 sobre K, uma decisão só. E a RSI é esse mesmo laço apontado para o próprio laço. O código de cada frase está no repositório proof, em Z3 e Haskell; o Python só desenha. Agora os créditos, porque este vídeo também roubou como um artista. O estilo é o de 3Blue1Brown, de Grant Sanderson. As animações são Manim,

a biblioteca dele, mantida pela Manim Community (Manim Community 2026). O roteiro, o código e as animações foram feitos com o Claude Code, da Anthropic, a partir dos meus prompts. O teste do Manim foi feito com o Claude Sonnet 5.5, e o roteiro e o código com o Claude Fable 5.1. A lista completa dos prompts está no repositório, e o custo total da produção está na tela. Eu revisei tudo; o que estiver errado é meu. Roubem à vontade; creditem.

Tela. Os três títulos de novo, cada um com o nome do arquivo embaixo. Depois, a tela de créditos: 3Blue1Brown · Manim Community · Claude Code (Sonnet 5.5, Fable 5.1) · custo total `[a confirmar: preencher no fim da produção, em dólares e em horas]` · "prompts: PROMPTS.md".

Código. `z3-scala/src/main/scala/bandit/domain/Video.scala`, `haskell/src/Video.hs`, `haskell/src/Pi.hs`, `haskell/app/Video.hs`, `results.md` e `PROMPTS.md`. As animações: `animation/`.

Créditos

Table 1: O que este vídeo tomou emprestado, de quem, e onde isso entra.

O quê	De quem	Como entra
Estilo visual	3Blue1Brown, Grant Sanderson	citado na fala e na tela de créditos; sem o personagem π
Animações	Manim (Manim Community) (Manim Community 2026)	`animation/timeline.py` e os takes
Gráficos interativos	Desmos, calculadora gráfica	`desmos/`
Voz	XTTS v2 (Coqui) (Coqui 2024) e OpenVoice V2 (MyShell) (MyShell AI 2024b)	`voice/narrate.py`
Roteiro, código, animações	Claude Code (Anthropic): Claude Sonnet 5.5 (teste do Manim), Claude Fable 5.1 (roteiro, Z3, Haskell)	a partir dos prompts em `PROMPTS.md`
Invariantes	Z3 (`z3-scala/src/main/scala/bandit/domain/Video.scala`, R1 a R5; `z3-scala/src/main/scala/bandit/domain/Theorems.scala`, as regras) e o espelho em Haskell	conferidas pelo spec check de `app/MyVideo.scala` com `ajust z3`
Custo total	`[a confirmar: preencher só no fim da produção]`	tela de créditos

A confirmar

- Cena 1.1: a referência de "Side Agents" da Anthropic.
- Cena 1.2: o nome do amigo e o mês de 2018.
- Cena 1.2: Renan Moreira de Oliveira foi para a Uber?
- Cena 1.4: os epítetos "príncipe dos matemáticos" (Gauss) e "rei" (Ramanujan, do autor) não têm fonte aberta neste roteiro. ⚠

- Seção Voz: a voz de referência (a do autor, ou um locutor).
- Cena 3.6: o custo total, no fim da produção.

Declaração sobre o uso de IA

Este roteiro, o código em Z3 e em Haskell que sustenta o roteiro e este documento foram escritos com o Claude Code (Anthropic), pelos modelos Claude Sonnet 5.5 e Claude Fable 5.1. Os modelos partiram dos prompts do autor, listados em `PROMPTS.md` com a hora de cada um. O autor revisou e aprovou o rascunho em 9 de novembro de 2026. Toda fonte citada foi lida antes de ser citada; `REFERENCES.md`, ao lado deste documento, diz o que foi conferido em cada uma.

Referências

Anthropic Institute. 2026. "When AI builds itself". Essay, with data of May 2026, updated 2026-09-18. https://www.anthropic.com/institute/recursive-self-improvement.

Bhavsar, Pratik. 2026. "Review of 'SIA: Self Improving AI with Harness & Weight Updates' (arXiv 2605.27276)". thersibook.com. https://thersibook.com/papers/sia.html.

Coqui. 2024. "XTTS v2, documentation". coqui-tts.readthedocs.io. https://coqui-tts.readthedocs.io/en/latest/models/xtts.html.

Dezeen. 2017. "Netflix targets different film artworks to suit users' viewing habits". https://www.dezeen.com/2017/12/20/netflix-targets-film-artwork-depending-users-viewing-habits-design-technology/.

Hoffmann, Matheus. 2026a. "Bandit contextual de múltiplos braços: o que é, como generaliza o teste A/B, e a Netflix". Note contextual-bandits/README.pt-BR.md in the repository h0ffmann/proof. https://github.com/h0ffmann/proof/blob/main/contextual-bandits/README.pt-BR.md.

———. 2026b. "proof: specifications in Lean 4, ported to Haskell". GitHub repository. https://github.com/h0ffmann/proof.

Jebara, Tony. 2018. "Artwork Personalization at Netflix". Keynote talk, AI Council. https://www.aicouncil.com/talks/artwork-personalization-at-netflix.

Li, Lihong, Wei Chu, John Langford, e Robert E. Schapire. 2010. "A Contextual-Bandit Approach to Personalized News Article Recommendation". Em *Proceedings of the 19th International Conference on World Wide Web*. https://doi.org/10.1145/1772690.1772758.

Manim Community. 2026. "Manim: a community-maintained Python library for creating mathematical animations". GitHub repository. https://github.com/ManimCommunity/manim.

Motta, Tiago Albineli. 2024. "No one cared about Multi Armed Bandit's tech". Blog post, 2024-09-28. https://tiago.rio.br/work/globocom/recommendation/no-one-cared-about-multi-armed-bandit-tech/.

MyShell AI. 2024a. "MeloTTS". GitHub repository. https://github.com/myshell-ai/MeloTTS.

———. 2024b. "OpenVoice V2, model card". Hugging Face. https://huggingface.co/myshell-ai/OpenVoiceV2.

Oliveira, Renan Moreira de. 2018. "Machine Learning na globo.com". Slides. https://www.slideshare.net/slideshow/machine-learning-na-globocom/94369887.

Rhasspy. 2023. "piper-voices: pt/pt_BR/faber/medium". Hugging Face, tag v1.0.0. https://huggingface.co/rhasspy/piper-voices/tree/v1.0.0/pt.

Wikipedia contributors. 2026a. "A/B testing". Wikipedia, The Free Encyclopedia. Read on 2026-11-09. https://en.wikipedia.org/wiki/A/B_testing.

———. 2026b. "Multi-armed bandit". Wikipedia, The Free Encyclopedia. Read on 2026-11-09. https://en.wikipedia.org/wiki/Multi-armed_bandit.

———. 2026c. "Steal Like an Artist". Wikipedia, The Free Encyclopedia. Read on 2026-11-09. https://en.wikipedia.org/wiki/Steal_Like_an_Artist.