Pós-Graduação em Inteligência Artificial Generativa e LLMs

PUC-Rio

Deep Generative Learning

Notas de Aula

Vítor Wilher

DGL · 2025.2

Versão 1.0

Resumo. Modelos generativos profundos: autoencoders variacionais, redes adversariais, modelos de difusão e aprendizado por reforço.

Palavras-chave: GAN; VAE; difusão; RLHF

Notas de estudo elaboradas a partir do material das aulas.

Índice

1 Visão Geral da Disciplina

A disciplina **Deep Generative Learning (DGL)** percorre o caminho que vai dos primeiros modelos generativos profundos até o estado da arte atual em geração de imagens, vídeo e alinhamento de modelos de linguagem. O ponto de partida é conceitual: entender a diferença entre uma IA que **classifica, prevê e detecta padrões** e uma IA que **cria novos dados**. Como formulado nos slides, "não é mais sobre entender o mundo, é sobre recriá-lo". Essa mudança de objetivo exige uma ideia central: em vez de escrever regras explícitas sobre como um rosto ou um dígito deve ser, o modelo **aprende a distribuição dos dados** e passa a amostrar novos pontos dessa distribuição.

A partir dessa base, a disciplina constrói uma progressão histórica e técnica bem definida. Primeiro vêm as **GANs** (2014), que resolvem o problema do realismo por competição entre gerador e discriminador, mas herdam instabilidade de treinamento e falta de controle. Em seguida vêm os **VAEs**, que sacrificam parte do realismo em troca de um **espaço latente organizado, contínuo e navegável**. A síntese das duas ideias aparece nos **Modelos de Difusão**, que tratam a geração como um processo iterativo de remoção de ruído e passam a dominar o estado da arte a partir de 2020. Os **Diffusion Transformers (DiT)** completam esse arco substituindo a U-Net por um Transformer puro, o que abre caminho para multimodalidade nativa e escalabilidade previsível.

A segunda metade da disciplina muda o eixo: sai da geração de imagens e entra em **Aprendizado por Reforço (RL)** como instrumento de alinhamento. Começa pelos fundamentos clássicos (Processo de Decisão de Markov, Q-Learning, exploration versus exploitation), passa por métodos policy-based (Policy Gradients, Actor-Critic, PPO) e chega ao **RLHF** e suas alternativas modernas (**DPO** e **GRPO**), incluindo a extensão para modelos multimodais (**MM-RLHF**). O fecho da disciplina trata de **Foundation Models para séries temporais** e de arquiteturas de **RAG** e SLMs aplicados a consultas SQL.

O fio condutor entre todos esses temas é sempre o mesmo: **como transformar um problema de geração difícil em uma sequência de problemas menores e tratáveis**, e **como injetar controle** (condicionamento, preferência humana, recompensa) num processo que, por natureza, é probabilístico.

2 SML sql

Material de slides não disponível para esta aula.

O tópico de **SML sql** aparece na ementa oficial associado ao trabalho com **Small Language Models (SLMs)** e geração de consultas SQL, e o material de apoio da disciplina indica notebooks dedicados a esse tema (`slm_rag_sql.ipynb` e `masterclass_rag_sql_slm.ipynb`), compartilhados com a aula de RAG. O deck de slides correspondente foi registrado no material sob o título "Engenharia de Sistemas RAG Modernos", mas o texto do arquivo não está disponível no contexto fornecido.

O que o restante do material da disciplina permite afirmar sobre o tema é o seguinte:

- **Text-to-SQL como caso de uso de LLM/SLM**: o slide sobre engenharia de prompts para domínios técnicos descreve explicitamente o **Tool Calling (Function Calling)** como o mecanismo que "permite que o LLM execute scripts Python para consultar bancos de dados

industriais". A geração de SQL se encaixa nesse padrão: o modelo não responde direto, ele produz uma consulta que é executada por um sistema determinístico.

- **Por que SLMs**: o material da aula sobre sistemas industriais registra que "LLMs na nuvem são muito lentos para controle em tempo real (necessidade de SLMs no Edge)". A mesma lógica de custo e latência favorece modelos menores para tarefas fechadas e repetitivas como tradução de linguagem natural para SQL.
- **Saída estruturada**: o material recomenda "JSON Output: forçar o LLM a responder em formato estruturado" e "parsing estruturado (JSON), testes automatizados" como parte do pilar de validação. Consultas SQL geradas por modelo devem passar pelo mesmo tipo de guardrail.

Um esqueleto ilustrativo do padrão descrito no material:

```
# Padrao: LLM/SLM gera a consulta, o sistema executa de forma deterministica
schema = "tabela vendas(data DATE, sku TEXT, quantidade INT, valor NUMERIC)"

prompt = f"""Voce e um gerador de SQL. Use apenas o schema abaixo.
Schema: {schema}
Responda SOMENTE com JSON: {{"sql": "..."}}
Pergunta: qual o total vendido por SKU no ultimo mes?"""

# resposta esperada do modelo (estruturada, validavel)
# {"sql": "SELECT sku, SUM(valor) FROM vendas
#          WHERE data >= date('now','-1 month') GROUP BY sku"}
```

O ponto pedagógico, coerente com o restante da disciplina, é que o modelo generativo cuida da parte ambígua (interpretar a pergunta) e o mecanismo determinístico cuida da parte que não admite erro (o cálculo).

3 IA Generativa: GAN e VAE

3.1 O que é IA Generativa

A definição adotada nos slides é direta: uma tecnologia que **aprende padrões complexos a partir de grandes volumes de dados** e, a partir deles, **cria conteúdo novo e original** — textos, imagens, áudios, código e vídeos. Os casos de uso citados incluem criação de conteúdo em escala, geração de código, design e prototipagem rápida, e síntese de informação (resumir, traduzir, extrair insights).

Um ponto histórico importante nos slides é que a IA Generativa **não começou com o ChatGPT**. Em **2019**, o engenheiro **Phillip Wang** lançou o site *This Person Does Not Exist*, que gerava um rosto humano hiper-realista completamente falso a cada atualização de página, usando o **StyleGAN** da NVIDIA. O StyleGAN trouxe controle sem precedentes sobre estilo em diferentes escalas (textura, forma e identidade).

3.2 O desafio central: gerar algo "real"

Os slides isolam três dificuldades que tornam a geração um problema qualitativamente diferente da classificação:

- **Sem regras explícitas**: ninguém escreve "uma face tem dois olhos, um nariz".
- **Sem rótulo de geração**: os dados não vêm com instruções de como criá-los.
- **Apenas exemplos reais**: o modelo aprende só vendo o que existe.

A solução conceitual é **aprender a distribuição**. Se o modelo entende como os dados se distribuem no espaço, ele consegue **amostrar novos pontos** dessa distribuição, ou seja, gerar novos exemplos válidos.

3.3 GAN: a ideia que mudou tudo

Em **2014**, **Ian Goodfellow** — doutorando na Université de Montréal, orientado por **Yoshua Bengio** e autor do "Deep Learning Book" — propôs as **Generative Adversarial Networks**. Segundo o relato dos slides, a ideia surgiu em uma conversa e foi implementada na mesma noite: "Fui pra casa e implementei na mesma noite. Funcionou de primeira." **Yann LeCun** classificou a proposta como "uma das ideias mais importantes dos últimos 10 anos em Machine Learning".

A GAN é **teoria dos jogos aplicada a IA**: dois modelos em competição direta.

- **Gerador (G)**: aprende a criar dados falsos convincentes. Análogo a "uma equipe de falsificadores, tentando produzir uma moeda falsa que não seja detectada".
- **Discriminador (D)**: aprende a distinguir real de falso. Análogo "à polícia, tentando detectar a moeda falsa".

A estrutura é a de um **jogo minimax de dois jogadores**: o treinamento do gerador tenta **maximizar a possibilidade de o discriminador cometer um erro**. Um detalhe fundamental destacado nos slides é que **o gerador nunca vê dados reais diretamente** — ele aprende apenas pelo feedback do discriminador. Como as GANs tentam replicar uma distribuição de probabilidade, a função de custo deve refletir a distância entre a distribuição dos dados gerados e a distribuição dos dados reais.

O aprendizado é **implícito e não supervisionado**: não há labels, não há anotações, e o modelo não aprende regras ("rosto tem dois olhos simétricos") mas sim **estruturas estatísticas profundas** que definem o que é "real".

3.4 Limitações das GANs

Os slides listam três problemas centrais:

1. **Treinamento instável**: gerador e discriminador precisam evoluir em sincronia.
2. **Colapso de modo**: o gerador aprende a enganar com poucas amostras repetidas, ignorando diversidade.
3. **Difícil de controlar**: você gera dados, mas não determina exatamente o que sairá.

O problema do controle é aprofundado: o espaço latente **existe, mas é desorganizado**. Não há relação clara entre regiões do espaço e tipos de imagem gerada, não se sabe o que cada dimensão do vetor representa, e não é possível pedir de forma confiável "gere um 3 parecido com esse". A **Conditional GAN** é apresentada como solução parcial, que exige rótulos e é frágil.

O estudo de caso proposto: **treinar uma GAN para transformar um ruído gerado com distribuição uniforme em uma imagem de um dígito de 0 a 9** (notebook `DCGAN.ipynb`).

3.5 VAE: aprender a representar em vez de apenas imitar

A pergunta que motiva o **Variational Autoencoder** é: "e se o espaço latente fosse estruturado?". A distinção conceitual nos slides é elegante:

- **GAN** aprende a **imitar** (tão bem que engana um discriminador).
- **VAE** aprende a **representar** (encontrar a estrutura oculta nos dados).

O VAE combina compressão com estrutura probabilística. Ele **comprime**, **organiza** e **reconstrói**. A intuição central do espaço latente:

- **Imagens viram pontos**: cada exemplo se torna uma coordenada em um espaço de baixa dimensão.
- **Proximidade é similaridade**: imagens parecidas ficam próximas.
- **Contínuo e navegável**: é possível mover-se pelo espaço e gerar qualquer imagem nele.

A arquitetura é um **funil inteligente**: `Encoder -> Latente -> Decoder`. O encoder comprime a imagem em sua essência e o decoder reconstrói a partir dessa representação, forçando o modelo a aprender o que realmente importa. Um ponto notável enfatizado nos slides é que **o modelo nunca recebeu rótulos** — a estrutura emerge sozinha: um "3" fica perto de outros "3" mesmo com escritas diferentes, regiões do espaço fazem sentido semântico e as fronteiras entre classes são suaves.

3.6 Não é um ponto, é uma distribuição

A inovação técnica do VAE é que o encoder **não mapeia uma imagem para um ponto fixo**, mas para uma **distribuição gaussiana** no espaço latente, caracterizada por:

- **Média (μ)**: o centro da distribuição, onde a imagem "provavelmente" está.
- **Variância (σ)**: a incerteza, o quanto a representação é espalhada ao redor da média.

Em vez de "essa imagem É esse ponto", o VAE aprende "essa imagem VIVE nessa região".

3.7 A função de perda do VAE

O treinamento equilibra **duas forças opostas e complementares**:

1. **Reconstrução**: a imagem reconstruída deve ser parecida com a original (erro pixel a pixel).
2. **Organização (divergência KL)**: a distribuição aprendida deve ser próxima de uma gaussiana padrão, o que garante a estrutura do espaço.

Os slides descrevem o treinamento como "uma negociação constante entre essas duas forças".

3.8 Geração controlada e interpolação

Com o espaço latente organizado, gerar passa a ser um procedimento de três passos:

1. **Escolher um ponto**: selecionar coordenadas no espaço latente.
2. **Passar pelo Decoder**: o decoder transforma esse ponto em pixels.
3. **Obter uma imagem com sentido**: a saída reflete a região escolhida.

A propriedade mais poderosa é a **interpolação**: ao mover suavemente de um ponto A a um ponto B, a imagem gerada se transforma de forma gradual e coerente — um "5" vira um "9" passando por formas intermediárias plausíveis. Os slides afirmam que isso é "impossível de fazer com GANs de forma confiável" e que a transformação suave prova que o espaço é contínuo e organizado, sem saltos bruscos.

3.9 Síntese: GAN, VAE e Diffusion

O quadro comparativo apresentado ao final da aula:

Modelo	Contribuição principal
GAN	Realismo: aprende a imitar muito bem, mas sem estrutura
VAE	Estrutura: aprende a organizar e representar
Diffusion	Os dois: realismo de GAN + estrutura de VAE

O estudo de caso da segunda parte: **treinar um VAE para gerar imagens de dígitos de 0 a 9** (notebook `VAE.ipynb`).

4 Diffusion Transformers

4.1 Da convolução à atenção

A aula (Prof. Manoela Kohler) abre com um recap: GANs geram com alta qualidade mas com treinamento instável; VAEs têm espaço latente bem organizado mas imagens borradas. A tese central é que **arquiteturas específicas estão sendo substituídas por arquiteturas universais**. A linha de evolução: CNNs (padrões locais), GANs (geração adversarial), VAEs (geração probabilística), Diffusion (remoção de ruído) e Transformers (modelagem global e unificada).

Os componentes do Transformer original relembrados nos slides: **Encoder** (transforma a entrada em representações contextuais ricas), **Decoder** (gera a saída token a token) e **Positional Encoding** (injeta informação de ordem, compensando a natureza não-sequencial do self-attention).

4.2 Imagem como sequência: o Vision Transformer

O **ViT** reimagina como máquinas enxergam:

- **Patches**: a imagem é dividida em blocos regulares; cada patch vira um token de entrada.
- **Positional Encoding**: tokens recebem informação de posição espacial.
- **Attention Global**: relações de longa distância são capturadas diretamente, sem o viés local das convoluções.

A frase-síntese dos slides: "uma imagem é apenas uma frase de patches".

4.3 Por que abandonar a U-Net

A **U-Net** dominou o backbone da difusão porque a arquitetura encoder-decoder com **skip connections** permite operar em múltiplas escalas simultaneamente, capturando detalhes finos e contexto geral. Mas os slides listam quatro limitações fundamentais:

- **Localidade**: convoluções capturam apenas vizinhanças; relações distantes se perdem.
- **Contexto global**: dificuldade em modelar dependências de longa distância.
- **Multimodalidade**: integração com texto não é nativa, requer adaptações complexas — descritas como "um remendo".
- **Escalabilidade**: mais dados e mais compute ajudam, mas a arquitetura limita a captura de relações globais.

Uma tentativa intermediária registrada nos slides foi **substituir a U-Net por MLPs**, tratando a imagem como vetor plano ou patches discretos, motivada por simplificar a arquitetura e remover vieses convolucionais. Os problemas encontrados foram decisivos: não escala com resoluções maiores, tem dificuldade em capturar a estrutura espacial, custo computacional alto e performance significativamente inferior. A conclusão registrada: "MLPs são simples, mas não entendem bem o espaço".

4.4 A arquitetura DiT

O **Diffusion Transformer** substitui a U-Net por um **Transformer puro** dentro do pipeline de difusão. O fluxo tem quatro etapas:

1. **Latent Space**: imagem comprimida pelo encoder do VAE.
2. **Patchificação**: o latente é dividido em patches, que viram tokens.
3. **Transformer**: blocos de atenção processam os tokens.
4. **Previsão de ruído**: a saída é decodificada para o ruído estimado.

O detalhe crítico é que **todo o processamento ocorre no espaço latente**, comprimido pelo encoder do VAE, o que torna o DiT drasticamente mais eficiente do que operar no espaço de pixels. O **timestep embedding** injeta informação sobre quanto denoising ainda resta, guiando o modelo em cada passo do processo reverso.

O DiT foi proposto no artigo *Scalable Diffusion Models with Transformers*, por pesquisadores da **UC Berkeley e NYU**, e rapidamente adotado pela indústria, incluindo a Meta.

4.5 O que muda na prática

Antes (U-Net)	Depois (DiT)
Convolução (receptive field local)	Attention global entre todos os tokens
Hierarquia piramidal encoder-decoder	Sequência plana de blocos uniformes
Skip connections	Removidas; fluxo linear entre blocos
Condicionamento via cross-attention	Condicionamento via AdaLN

Os slides registram que o condicionamento via normalização adaptativa "resultou em melhor performance que o uso de cross attention".

4.6 AdaLN-Zero: o coração do controle

O **Adaptive Layer Normalization** é o mecanismo que injeta condicionamento no DiT. O raciocínio é construído em etapas:

1. **LayerNorm padrão**: normaliza para estabilizar o treinamento e depois aplica uma transformação afim com parâmetros treináveis γ (escala) e β (deslocamento), que ficam **fixos após o treinamento**.
2. **AdaLN**: os parâmetros γ e β passam a ser **gerados por uma MLP** a partir de um embedding de contexto (a condição, como timestep ou texto). A normalização das ativações passa a depender do contexto.
3. **O problema**: injetar o condicionamento diretamente pode levar à instabilidade — o condicionamento domina cedo demais no processo de denoising, o que pode fazer o treinamento explodir ou degradar a performance.
4. **A solução (AdaLN-Zero)**: **inicializar γ e β como zero**. No início, o bloco de Transformer atua como **função identidade** e o condicionamento é ignorado, prevenindo instabilidade. Durante o treinamento, os parâmetros são ajustados progressivamente, até que o condicionamento passe a guiar todo o processo de denoising.

A analogia dos slides: "é como dar uma nova informação para alguém: primeiro ele ignora, depois começa a considerar, até que aquilo passa a influenciar tudo que ele faz". O AdaLN-Zero é aplicado **em cada bloco Transformer, antes das camadas principais** (atenção e feedforward), modulando diretamente suas ativações.

Uma nota de precisão dos slides: não é que cross-attention "não exista" no DiT, mas que **no DiT original ela não é necessária**. Modelos modernos usam **ambos** os mecanismos, com papéis complementares — **AdaLN** para controle global (modulação adaptativa abrangente) e **cross-attention** para alinhamento fino (foco preciso em regiões específicas).

4.7 MMDiT: a próxima fronteira

O **MMDiT** (usado no Stable Diffusion 3.5) integra texto e imagem numa arquitetura Transformer, **mantendo processamento específico por modalidade** em partes do bloco e promovendo comunicação entre elas por **atenção conjunta**. Texto e imagem são convertidos em sequências de tokens compatíveis, mas preservam características próprias de cada modalidade. Na atenção conjunta:

- cada palavra pode "olhar" para partes da imagem;
- cada parte da imagem pode "olhar" para palavras;
- e também há atenção interna imagem-imagem e texto-texto.

4.8 Estado da arte e limitações

Os modelos citados: **DiT** (o paper original que demonstrou Transformers superando U-Nets em benchmarks de geração de imagem), **Stable Diffusion 3** (adota DiT com MMDiT), e **Sora, Imagen Video e Veo3** (geração de vídeo da OpenAI e Google, ambos transformer-based diffusion em escala). A tendência registrada é clara: "todo o estado da arte está convergindo para diffusion + transformer".

As vantagens: modelagem global, integração com texto, melhor scaling (performance cresce previsível e consistentemente com dados) e arquitetura simples (menos componentes especializados). As limitações:

- **Custo computacional**: treinamento e inferência demandam hardware de ponta.
- **Atenção quadrática**: complexidade $O(n^2)$ em relação ao número de tokens.
- **Treinamento pesado**: modelos maiores exigem datasets massivos.
- **Ainda em evolução**: melhores práticas de arquitetura e scaling ainda estão sendo descobertas.

Os slides mencionam **Flash Attention** e **sparse attention** como área ativa de pesquisa para reduzir o custo quadrático sem perder qualidade.

O material de apoio da aula inclui ainda dois artigos que aprofundam a eficiência de DiTs: **Q-DiT**, sobre quantização pós-treinamento (Post-Training Quantization) para Diffusion Transformers, que identifica variância espacial em pesos e ativações e variância temporal nas ativações como fontes de degradação; e **DC-DiT** (Dynamic Chunking Diffusion Transformer), que substitui a patchificação estática por um mecanismo aprendido de compressão adaptativa, gastando menos tokens em regiões uniformes e em timesteps ruidosos, e mais tokens em regiões ricas em detalhe e nos estágios finais do denoising.

5 Modelos de Difusão

5.1 O que se quer de um modelo generativo ideal

A aula (Prof. Manoela Kohler) parte das limitações já conhecidas e enuncia três requisitos: **alta qualidade** (imagens realistas, detalhadas e coerentes), **estabilidade** (treinamento robusto, sem colapso de modo) e **controle** (capacidade de guiar a geração com prompts, classes ou condicionamentos).

A ideia que satisfaz os três é reformular a pergunta: **e se gerar fosse um processo?** Em vez de gerar a imagem de uma só vez, algo muito complexo, a geração é dividida em **muitos passos pequenos e simples**. Os Diffusion Models tratam a geração como um **processo iterativo**, não como uma transformação direta.

5.2 História breve

- **2015** — *Deep Unsupervised Learning using Nonequilibrium Thermodynamics* (Sohl-Dickstein, Weiss, Maheswaranathan e Ganguli): a referência mais antiga para Diffusion Models. À época, GANs dominavam o estado da arte e os modelos de difusão não recebiam atenção. A ideia essencial, inspirada em física estatística de não-equilíbrio, é **destruir sistemática e lentamente a estrutura de uma distribuição de dados** por meio de um processo iterativo de difusão forward, e então **aprender um processo reverso** que restaura essa estrutura. O artigo usa uma cadeia de Markov cujos passos individuais têm probabilidade analiticamente avaliável, de modo que a cadeia completa também pode ser avaliada; o treinamento consiste em **estimar pequenas perturbações a um processo de difusão**, o que é mais tratável do que descrever a distribuição inteira com uma única função potencial não normalizável.
- **2020** — *Denoising Diffusion Probabilistic Models* (DDPM), de Jonathan Ho, Ajay Jain e Pieter Abbeel (UC Berkeley): o paper que trouxe os Diffusion Models para o centro das atenções, acumulando mais de 20.000 citações e levando a difusão ao estado da arte em geração de imagens.

5.3 Forward e reverse process

Forward process (destruir): em cada passo, adiciona-se uma pequena quantidade de **ruído gaussiano** à imagem. Após centenas de passos, a imagem original é completamente destruída, restando apenas ruído puro. Este processo é **fixo e matemático** — não precisa ser aprendido.

A pergunta central: "se eu sei destruir, consigo reconstruir?". Como o processo de destruição é controlado e bem definido matematicamente, se o modelo aprender como ele funciona, talvez consiga **reverter cada passo**.

Reverse process (reconstruir): o modelo aprende a executar o caminho inverso — dado um estado ruidoso, ele prediz como era o estado um passo antes. Repetindo isso dezenas ou centenas de vezes, a imagem se revela progressivamente.

5.4 O insight principal

A frase que resume a aula: **"Diffusion não aprende a gerar, aprende a remover ruído."** Em vez de modelar diretamente a distribuição complexa das imagens, o modelo resolve um problema muito mais simples em cada etapa: **estimar e subtrair o ruído presente**.

Por que essa decomposição funciona:

- Gerar uma imagem realista do zero é extremamente difícil; remover uma pequena quantidade de ruído de uma imagem levemente corrompida é simples.
- O problema complexo é dividido em **centenas de subproblemas triviais**.
- Cada passo é fácil de aprender com gradiente descendente.
- O modelo **acumula pequenas melhorias** até obter um resultado excelente.

A analogia usada nos slides é a do **restaurador de fotografias antigas**: ele não conserta tudo de uma vez, trabalha em camadas progressivas — remove uma mancha, suaviza um arranhão, reconstrói

detalhes gradualmente. O Diffusion Model faz o mesmo, com ruído gaussiano e em sentido inverso ao da degradação.

Visualmente, nos primeiros passos do reverse process surgem apenas **formas grosseiras** (contornos vagos e manchas de cor); conforme os passos avançam, **texturas, bordas e cores precisas** se consolidam.

5.5 Por que Diffusion vence GAN na prática

GANs sofrem com o **equilíbrio frágil** entre gerador e discriminador. Diffusion Models têm um **objetivo de treinamento bem definido**: minimizar o erro de predição do ruído. Isso torna o processo muito mais **estável e reproduzível**.

5.6 Stable Diffusion: a conexão com o VAE

Executar difusão diretamente no espaço de pixels é caro demais. O **Stable Diffusion** resolve isso operando no **espaço latente comprimido do VAE**:

1. O **VAE comprime** a imagem numa representação latente menor.
2. A **difusão opera nesse espaço latente**, o que é muito mais rápido.
3. O **VAE decodifica** o resultado de volta para pixels.

Os slides explicam as métricas relevantes:

- **f** é o **fator de compressão**. Se a imagem original é 256x256, um `f=4` resulta num latente 64x64.
- **PSNR** (Peak Signal-to-Noise Ratio) mede quão próxima a imagem gerada está da original.
- **R-FID** (Reconstruction Fréchet Inception Distance) mede a distância entre distribuições de imagens reais e geradas.

A arquitetura simplificada do Stable Diffusion tem três blocos principais: **codificação de texto**, **difusão no espaço latente** e **reconstrução via VAE**. A **Text Conditioned U-Net** pode incorporar o texto por diferentes mecanismos — concatenação, modulação das ativações ou **atenção cruzada** — para guiar a geração no espaço latente.

As tarefas suportadas listadas nos slides: **Text-to-Image**, **prompts negativas**, **Image-to-Image**, geração **condicionada à classe**, **Super Resolution**, **Layout-to-Image** e **Inpainting**. O ano de **2022** é apontado como o ponto de inflexão que levou os modelos de difusão do laboratório para o uso real.

5.7 Limitações

1. **Velocidade de inferência**: gerar uma imagem requer dezenas a centenas de passos de denoising, tornando o processo significativamente mais lento que GANs.
2. **Custo computacional**: cada passo envolve uma passagem completa pela rede neural.

3. **Número de passos**: reduzir drasticamente os passos compromete a qualidade. Técnicas como **DDIM** (Denoising Diffusion Implicit Models) mitigam isso, mas há um trade-off inevitável. A observação técnica dos slides: no diffusion clássico (DDPM) o processo é **estocástico**, e por isso precisa de muitos passos; o **DDIM transforma o processo em determinístico (ou quase)**.

A conclusão da aula: os Diffusion Models funcionam melhor porque **tornam o problema mais simples**. Essa decomposição inteligente é o que garante estabilidade, qualidade e flexibilidade.

Os estudos de caso: **DDPM** (notebook `MNIST_Diffusion.ipynb`) e **Stable Diffusion** (notebook `Stable_Diffusion.ipynb`).

6 Introdução a Reinforcement learning

6.1 Contextualização

A aula (Ph.D. Evelyn Batista, ICA/PUC-Rio) cobre três blocos: Introdução a Reinforcement Learning, Q-Learning e RLHF com PPO. As aplicações citadas incluem computação gráfica, robótica, drones, softwares inteligentes e múltiplos agentes. Um exemplo industrial concreto: na Fanuc, **um robô usa deep reinforcement learning para escolher um dispositivo de uma caixa e colocá-lo em um recipiente**; se ele é bem-sucedido ou falha, memoriza o objeto, ganha conhecimento e treina para fazer o trabalho com grande velocidade e precisão.

6.2 RL versus aprendizado supervisionado

Aprendizado por Reforço	Aprendizado supervisionado
Aprendizado a partir da interação	Aprendizado a partir de padrões entrada-saída
Baseado em tentativa e erro	Baseado em minimizar um erro
Existe busca (exploration) no espaço	Busca limitada aos valores dos padrões
Orientado a objetivo	Orientado a aproximação de função

Os slides observam que o aprendizado por reforço é **um modelo importante de como nós (e todos os animais) aprendemos**: elogios dos pais, notas na escola e salário no trabalho são exemplos de recompensas. Problemas de **atribuição de crédito** e dilemas de **exploration-exploitation** surgem todos os dias, e jogos são ótimos exemplos para experimentar novas abordagens.

6.3 Processo de Decisão de Markov

A formalização padrão é o **Processo de Decisão de Markov (MDP)**. São ditos "de Markov" porque obedecem à **propriedade de Markov**: o efeito de uma ação em um estado depende apenas da ação e do estado atual do sistema, **e não de como o processo chegou a tal estado**. São chamados "de decisão" porque modelam a possibilidade de um agente interferir periodicamente no sistema executando ações.

Traduzido para linguagem operacional:

- O ambiente está em certo **estado**.
- O agente pode executar certas **ações** no ambiente.
- Essas ações às vezes resultam em uma **recompensa**.
- As ações transformam o ambiente e conduzem a um **novo estado**.
- As regras sobre como se escolhem essas ações são chamadas de **política**.

Um episódio do processo (por exemplo, um jogo) forma uma **sequência finita de estados, ações e recompensas**.

6.4 Elementos de um sistema RL

- **Agente**: o aprendiz inserido no ambiente, que toma ações que o mudam. Busca atingir um objetivo enquanto interage com um ambiente desconhecido e incerto.
- **Estado**: a condição atual do ambiente, especificada por um conjunto de variáveis adequadas ao problema. Deve prover ao agente a informação de quais ações podem ser executadas e ser suficiente para que ele decida (satisfazendo a propriedade de Markov).
- **Ambiente / modelo do ambiente**: imita o comportamento do ambiente. Dados um estado e uma ação, o modelo antecipa o próximo estado e o ganho. Modelos são usados para **planejamento**.

6.5 Recompensa futura com desconto

Dada uma série do processo de decisão de Markov, é fácil calcular a recompensa total de um episódio. Mas **o ambiente é estocástico**, e por isso é comum usar uma **recompensa futura com desconto**. O fator de desconto γ tem interpretação direta:

- $\gamma = 0$: confia-se apenas nas recompensas imediatas.
- $\gamma = 1$: considera-se o ambiente determinístico, e as mesmas ações sempre resultarão nas mesmas recompensas.

6.6 Q-Learning

A pergunta que o Q-Learning responde: **como estimar a pontuação no final do jogo, se conhecemos apenas o estado atual e a ação, e não as ações e recompensas que vêm depois?**

Os elementos são: `s` (estado atual), `a` (ação), `r` (recompensa) e `s'` (próximo estado). A função **Q** representa a **recompensa futura máxima para este estado e ação**, decomposta em: a **recompensa imediata**, mais a **recompensa futura máxima para o próximo estado**, ponderada pelo **fator de desconto**.

Uma vez que se tenha a função Q "mágica", a resposta se torna simples: **escolher a ação com o Q-valor mais alto**. A **política** π é a regra de como se escolhe uma ação em cada estado — é a "estratégia" do agente.

A ideia principal é que a função Q pode ser **aproximada iterativamente usando a equação de Bellman**. No caso mais simples, Q é implementada como uma **tabela (matriz)**, com **estados como linhas e ações como colunas**, inicializada em zero e progressivamente preenchida.

A **taxa de aprendizagem** α determina em que medida as informações recém-adquiridas substituirão as antigas:

- $\alpha = 0$: o agente não aprende nada.
- $\alpha = 1$: o agente considera apenas a informação mais recente.

6.7 Exploration versus exploitation

O agente deve aprender, por interação, qual política melhor o leva ao objetivo. Como não possui conhecimento sobre todas as variáveis envolvidas, é necessário que ele **explore**, ou seja, busque políticas alternativas àquelas que já utiliza e compare seus resultados com os já aprendidos.

Os slides apontam também o salto para **Deep Q Network (DQN)**, quando o espaço de estados é grande demais para uma tabela. O exercício prático usa o ambiente **FrozenLake** do Gymnasium (`gymlibrary.dev`), com o notebook `1_Q_Learning.ipynb`.

```python
# Esqueleto ilustrativo de Q-Learning tabular
import numpy as np

n_estados, n_acoes = 16, 4
Q = np.zeros((n_estados, n_acoes))
alpha, gamma, epsilon = 0.1, 0.99, 0.1

# a cada transicao observada (s, a, r, s_linha):
#   alvo = r + gamma * np.max(Q[s_linha])
#   Q[s, a] = Q[s, a] + alpha * (alvo - Q[s, a])
```

6.8 RLHF e PPO

RLHF significa *Reinforcement Learning from Human Feedback*: uma técnica usada para treinar e ajustar modelos de IA, especialmente LLMs, para responder de forma mais útil, segura e alinhada ao que humanos esperam. A ideia geral tem quatro etapas:

1. **O modelo aprende inicialmente com textos**: é treinado para prever palavras e gerar respostas coerentes.
2. **Humanos avaliam respostas**: para uma mesma pergunta, o modelo gera várias respostas e pessoas escolhem quais são melhores.
3. **Cria-se um modelo de recompensa**: a IA aprende a prever que tipo de resposta os humanos preferem.
4. **O modelo é ajustado com aprendizado por reforço**: passa a gerar respostas tentando maximizar essa recompensa.

Em termos simples: "RLHF é como ensinar um modelo não apenas a falar, mas a responder de um jeito que humanos consideram melhor".

Por que **PPO** em LLMs: o fluxo é `Prompt -> Modelo gera resposta -> Modelo de recompensa avalia -> PPO ajusta o modelo`. O PPO é útil porque (1) trabalha bem com políticas probabilísticas, (2) lida melhor com espaços de ações enormes, como vocabulários de tokens, (3) evita mudanças bruscas no modelo e (4) é mais estável que policy gradient puro. Em vez de aprender

uma tabela Q para cada token em cada contexto, o modelo **ajusta diretamente sua distribuição de probabilidade sobre tokens**.

6.9 Q-Learning versus PPO

O **Q-Learning** é um método **value-based**: aprende "se estou no estado `s` e tomo a ação `a`, qual é o retorno esperado?" e depois escolhe a ação de maior valor. O **PPO** é um método **policy-based**, mais especificamente **Actor-Critic**: aprende diretamente "se estou no estado `s`, qual a probabilidade de escolher cada ação?".

O exemplo comparativo dos slides:

- **Q-Learning** aprende `Q(posicao, cima) = 2`, `Q(posicao, baixo) = -1`, `Q(posicao, direita) = 5`, `Q(posicao, esquerda) = 0` e escolhe *direita*, por ter o maior valor.
- **PPO** aprende `pi(cima) = 10%`, `pi(baixo) = 5%`, `pi(direita) = 75%`, `pi(esquerda) = 10%`. Se ir para a direita deu boa recompensa, o PPO **aumenta um pouco** essa probabilidade — mas não deixa virar 99% de uma vez.

A pergunta que define o PPO: "como posso aumentar a chance das ações boas e diminuir a chance das ações ruins **sem mudar a política de forma brusca demais**?". Essa última parte é o coração do algoritmo.

O ciclo básico do PPO: (1) o agente interage com o ambiente, (2) coleta estados, ações e recompensas, (3) avalia se as ações foram boas ou ruins, (4) atualiza a política e (5) **limita o tamanho da atualização**. Os elementos:

- **Actor e Critic**: o Ator escolhe a ação; o Crítico avalia a situação, estimando o valor de um estado ("o quão promissor é estar nesse estado?").
- **Recompensa**: o sinal que diz se o agente foi bem ou mal.
- **Vantagem**: mede se uma ação foi **melhor ou pior do que o esperado**.

O exemplo do robô que aprende a andar: ele move a perna para frente, recebe recompensa positiva, e o PPO calcula "essa ação foi melhor do que o esperado? Sim. Então aumente a chance dela acontecer novamente" — mas **com limite**: não aumente de 20% para 95% em uma atualização.

Aplicado a LLMs, o exemplo dos slides usa o prompt "explique fotossíntese para uma criança". A resposta A ("fotossíntese é quando a planta usa luz do sol para produzir seu alimento") recebe recompensa alta e o PPO aumenta a probabilidade de respostas parecidas. A resposta B ("fotossíntese é um conceito altamente complexo envolvendo mecanismos bioquímicos…") recebe recompensa menor e o PPO reduz a probabilidade de respostas inadequadas ao objetivo.

As referências indicadas incluem o paper do DQN, os artigos da Nature e do arXiv sobre RL profundo, *Deep Learning with Keras* (Gulli e Pal) e *Reinforcement Learning: An Introduction* (Sutton e Barto).

7 Sistemas de Geração de Imagens Baseados em Difusão

7.1 Objetivo da aula

Esta é a aula aplicada da disciplina (Prof. Heitor Teixeira de Azambuja), voltada a **usar** sistemas de geração baseados em difusão e a **avaliar criticamente** seus resultados. O roteiro cobre introdução, DiT versus U-Net, funcionamento, principais modelos no mercado, plataformas online, geração de vídeos e atividades práticas.

7.2 DiT versus U-Net: o salto tecnológico

- **U-Net (antigo)**: baseada em convoluções; ótima para detalhes locais, mas limitada em entender relações de longa distância na imagem.
- **DiT (novo)**: baseado em atenção; permite que cada parte da imagem "veja" todas as outras, resultando em **composições mais coerentes** e **melhor seguimento de prompts complexos**.
- **Escalabilidade**: quanto mais poder computacional se dá ao DiT, melhor a imagem fica — ao contrário das U-Nets, que tendem a estagnar.

A ideia central: **tratar pedaços da imagem (patches) como "palavras" em uma frase**, aplicando o mecanismo de atenção para entender o contexto global da cena.

7.3 Funcionamento: self-attention aplicado a imagens

A analogia dos slides: "é como ler uma frase inteira para entender o significado de uma palavra, em vez de ler letra por letra". O pipeline:

- **Tokenização (patches)**: a imagem é dividida em pequenos quadrados (por exemplo, 16x16 pixels); cada quadrado é um token.
- **Processamento Transformer**: os blocos de atenção analisam a relação entre esses patches durante os passos de difusão.
- **Matriz de afinidade**: o modelo calcula o quanto o "Patch A" (por exemplo, um olho) é relevante para o "Patch B" (por exemplo, a pata de um gato), para garantir que façam sentido juntos.
- **Globalidade**: isso resolve o problema de "mãos com 6 dedos" ou objetos flutuando, porque o modelo tem uma **visão panorâmica** da obra durante todo o processo.
- **Denoising progressivo**: o ruído é removido gradualmente até a imagem final emergir, guiada pelo texto.

Os três componentes da atenção, na formulação didática dos slides:

- **Query (consulta)**: o que este patch está procurando?
- **Key (chave)**: o que este patch tem a oferecer?
- **Value (valor)**: a informação real que será passada adiante se houver um "match".

O benefício direto para o entendimento de texto: a **cross-attention** cruza as palavras do prompt diretamente com os patches da imagem, garantindo que "um astronauta montando um cavalo" não vire apenas um astronauta e um cavalo separados.

7.4 Principais modelos no mercado

- **FLUX.1**: atualmente o rei do realismo e da renderização de texto (letras e frases corretas).
- **Stable Diffusion 3 (SD3)**: o primeiro grande modelo comercial a adotar DiT, equilibrando velocidade e qualidade artística.
- **PixArt-alfa**: focado em eficiência, gerando imagens de alta resolução com menos custo computacional.
- **Sora (OpenAI)**: utiliza DiT para criar vídeos realistas, provando que a tecnologia funciona além de imagens estáticas.

Plataformas indicadas para testes: **Shakker.ai** e **SeaArt.ai** (realismo, bons para FLUX.1), **Civitai** (comunidade e ajustes finos de modelos) e **Fal.ai** e **Hugging Face** (acesso direto aos modelos mais novos).

7.5 Geração de vídeos

A "grande virada de 2024/2025" foi aplicar DiT em vídeos: a IA trata o vídeo como um **"cubo" de patches (espaço + tempo)**. Os principais geradores citados:

1. **Luma Dream Machine**: um dos mais consistentes; usa arquitetura de difusão para manter a física e a coerência entre frames.
2. **Kling AI**: capaz de gerar vídeos de até 10 a 20 segundos com movimentos complexos.
3. **Runway Gen-3 Alpha**: focado em controle cinematográfico; usa difusão multimodal que entende comandos de câmera (por exemplo, "pan left", "zoom in").
4. **Hailuo AI (MiniMax)**: impressionante na criação de seres humanos e expressões faciais.

7.6 Atividades: prompts como testes de hipótese

A parte mais didática da aula transforma cada prompt em um **teste de uma capacidade arquitetural específica**:

1. **Relação espacial** — "Uma maçã vermelha equilibrada em cima de um cubo azul que está dentro de uma caixa de vidro". Se a imagem sair correta, é o mecanismo de atenção garantindo que a maçã "saiba" onde o cubo está.
2. **Renderização de texto** — um letreiro de neon com um texto específico numa rua chuvosa. Antigamente as IAs escreviam "hieróglifos"; modelos DiT conseguem soletrar. Verificar se todas as letras estão presentes e se a luz do texto reflete corretamente no ambiente.
3. **Anatomia e interação complexa** — close-up extremo de duas mãos fazendo um gesto entrelaçado. Contar os dedos: o DiT usa atenção global para entender que uma mão pertence a um braço e a outra ao outro, evitando fusão de dedos.
4. **Coerência e transparência** — um cubo de vidro transparente sobre uma mesa de madeira, com uma floresta em miniatura e um dragão dentro. O modelo consegue manter o dragão dentro do cubo lidando com a transparência?
5. **Estilo e materialidade** — macrofotografia de um bordado, observando fios individuais e trama do tecido. Testa a capacidade de gerar padrões repetitivos e texturas táteis sem borrões.

6. **Coerência temporal (vídeo)** — uma pessoa servindo café, o líquido respingando, e então bebendo e sorrindo, em plano contínuo. O café continua sendo café? A xícara muda de formato? Isso testa se o Transformer mantém a "memória" dos frames anteriores.

Uma nota prática relevante dos slides: **para a maioria dos modelos de ponta o inglês é preferível**. Modelos como FLUX.1 e DALL-E 3 já entendem prompts em português razoavelmente bem, pois foram expostos a múltiplas línguas durante o treinamento, mas em português o modelo pode se confundir com palavras de múltiplos significados ou falhar em detalhes específicos, recorrendo a conceitos mais genéricos. Os slides observam ainda que o DALL-E 3 supostamente **não usa DiT**, acreditando-se que utilize uma abordagem mais tradicional baseada em U-Net.

7.7 Conclusões

- **Fim da "sopa de pixels"**: menos distorções em mãos, rostos e textos.
- **Multimodalidade**: modelos que entendem imagem, vídeo e áudio no mesmo "cérebro".
- **Acessibilidade**: mesmo sendo modelos pesados, otimizações permitem que rodem em hardware comum cada vez mais rápido.

8 Reinforcement Learning

8.1 A jornada do RL em modelos de linguagem

Esta aula estrutura o RL aplicado a LLMs em três módulos: fundamentos (filosofia do RL, ciclo agente-ambiente, MDP, política e recompensa), algoritmos e agentes (Q-Learning, Deep RL, Policy Gradients, Actor-Critic e PPO) e RL em LLMs (RLHF, DPO, agentes autônomos e raciocínio).

A linha do tempo apresentada:

- **2017**: introdução do PPO.
- **2019-2020**: primeiras aplicações em modelos de linguagem.
- **2022**: adoção em larga escala com InstructGPT e ChatGPT.
- **2023-2024**: explosão de métodos alternativos (DPO, GRPO).
- **2025**: integração com modelos multimodais e raciocínio reforçado.

8.2 A premissa filosófica

O pilar conceitual é a **Hipótese da Recompensa**: "todos os objetivos e valores podem ser descritos pela maximização de uma recompensa cumulativa esperada". Isso significa que **qualquer objetivo pode ser traduzido em um número que o agente tenta maximizar**.

Os **três pilares do RL**:

- **Política (π)**: a estratégia do agente. Mapeia estados para ações — "dado este estado, qual ação devo tomar?".
- **Função de Valor (V)**: avalia o quão bom é estar em um estado — "qual é minha recompensa futura esperada?".
- **Recompensa (R)**: o feedback imediato do ambiente — "isso foi bom ou ruim?".

O **dilema fundamental** permanece: explorar demais significa nunca aproveitar o aprendido; explotar demais significa ficar preso em soluções subótimas.

8.3 Métodos: value-based, policy-based e híbrido

Método 1 — Value-based (Q-Learning): cria uma tabela onde cada célula representa o valor de tomar uma ação específica em um estado específico. A limitação: funciona bem em ambientes pequenos, mas falha quando há milhões de estados (por exemplo, pixels de uma tela). A solução é o **DQN**, que usa uma rede neural profunda para **aproximar a função Q**, permitindo lidar com espaços de estados imensos como imagens de videogames.

Método 2 — Policy Gradients: ajustam diretamente as probabilidades. A **regra de ouro** é aumentar a probabilidade se a ação teve recompensa e diminuir se houve punição. A vantagem é lidar bem com **ações contínuas e políticas estocásticas**.

Método 3 — Actor-Critic: o Ator decide a ação (policy) enquanto o Crítico avalia a qualidade da ação (value). Juntos são **mais estáveis e eficientes** que cada abordagem isolada.

PPO é descrito como o algoritmo padrão da indústria (usado pela OpenAI) e como uma versão melhorada do Actor-Critic. **O segredo do PPO** é limitar mudanças rápidas com **clipping**, para preservar o que o agente já aprendeu.

Marcos históricos citados: **AlphaGo (2016)**, que derrotou o campeão mundial de Go — jogo com mais configurações possíveis que átomos no universo observável; **AlphaZero (2017)**, que aprendeu xadrez do zero jogando contra si mesmo por 4 horas e superou o melhor motor de xadrez do mundo; e aplicações em **robótica e navegação** (robôs que andam, braços robóticos que manipulam objetos, balões estratosféricos que navegam).

8.4 O problema: supervisão não é suficiente

LLMs pré-treinados **apenas preveem a próxima palavra**. Eles podem gerar textos tóxicos, alucinar ou simplesmente não responder ao que o usuário pediu. A limitação é estrutural: **o aprendizado supervisionado ensina o modelo a imitar dados da internet, mas não ensina o que é "bom", "seguro" ou "útil"**.

8.5 RLHF: o pipeline completo que criou o ChatGPT

Três etapas transformam um LLM pré-treinado em um assistente alinhado:

Etapa 1 — Supervised Fine-Tuning (SFT): pega o modelo base e o treina com milhares de exemplos de alta qualidade escritos por humanos. O objetivo é transformar um "completador de textos" em um assistente que responde perguntas no formato correto (`Prompt -> Resposta`).

Etapa 2 — Reward Model: treina-se um segundo modelo para agir como juiz humano. Humanos recebem duas respostas do LLM e escolhem a melhor; o Reward Model aprende a **prever qual resposta um humano preferiria**, dando uma nota (recompensa) para qualquer texto.

Etapa 3 — RL puro (PPO): usa-se PPO para treinar o LLM a gerar respostas que o Reward Model avalie com notas altas. O ciclo é: `LLM gera resposta -> Reward Model dá nota -> PPO ajusta o LLM para gerar respostas parecidas com as que ganharam notas altas`.

O mapeamento entre RLHF e o RL clássico, explicitado nos slides:

Elemento do RL	Correspondente em RLHF
Agente	O LLM
Ambiente	O usuário (prompt) e o Reward Model
Estado	O contexto da conversa até agora
Ação	Gerar a próxima palavra (token)
Recompensa	A nota dada pelo Reward Model

8.6 Além do PPO

DPO (Direct Preference Optimization) otimiza o LLM **diretamente com preferências humanas**, sem modelo de recompensa separado nem PPO.

Agentes autônomos: LLMs modernos vão além de gerar texto — planejam tarefas, usam ferramentas externas (calculadora, navegador, código) e mantêm memória de conversas anteriores.

RL para raciocínio: modelos como o OpenAI o1 usam RL não apenas no treinamento, mas **durante a inferência** (quando a pergunta é feita). Na **Cadeia de Pensamento (Chain of Thought)**, o modelo é recompensado por explorar diferentes caminhos de raciocínio, corrigir seus próprios erros e chegar à resposta certa antes de mostrá-la ao usuário.

8.7 Perguntas em aberto

Os slides fecham com três questões honestas:

- **Quem define os valores?** Quando dizemos "valores humanos", quais culturas e vozes estão incluídas?
- **Reward Hacking**: como evitar atalhos que maximizam recompensa sem resolver o objetivo real?
- **Eficiência de dados**: RL requer muito ensaio; como acelerar o aprendizado para ser mais próximo do humano?

Recursos recomendados: *Reinforcement Learning: An Introduction* (Sutton e Barto), OpenAI Gym / Gymnasium para prática, e o paper *Training language models to follow instructions with human feedback* (InstructGPT/RLHF).

9 RL human feed back

9.1 Três blocos de conteúdo

Esta aula reúne três decks complementares: um sobre **automação com prompts** (casos reais e arquiteturas escaláveis), um sobre **RL em modelos multimodais** e um sobre **RLHF, PPO, DPO e GRPO**.

9.2 Automação com prompts: o paradoxo moderno

O **paradoxo** enunciado nos slides: LLMs permitem automação sem código tradicional, mas **exigem arquiteturas de orquestração e design de prompts muito mais sofisticados para garantir determinismo**. A transição de scripts determinísticos para sistemas probabilísticos requer novas abordagens de engenharia de software.

Os **quatro pilares da automação com prompts**:

- **Engenharia de Prompts**: design estruturado (Few-shot, CoT, ReAct).
- **Orquestração**: pipelines de dados (RAG, LangChain, LlamaIndex).
- **Validação**: guardrails, parsing estruturado (JSON), testes automatizados.
- **Monitoramento**: observabilidade de LLMs, detecção de drift e alucinação.

9.2.1 Caso 1: relatórios automáticos

Os desafios: **consistência** (manter tom e formato em execuções diferentes), **precisão numérica** (LLMs são notoriamente ruins em cálculos matemáticos complexos) e **integração de dados** (unir dados estruturados com narrativas não estruturadas).

O pipeline end-to-end proposto tem cinco etapas: (1) **coleta** via APIs/SQL, (2) **processamento** com cálculos determinísticos em Python/Pandas, (3) **síntese** — o LLM recebe **dados pré-calculados** e gera a narrativa, (4) **validação** de formato e consistência, e (5) **distribuição** (PDF/email automático). O ponto arquitetural essencial é que o cálculo **nunca** é delegado ao LLM.

As técnicas de prompt indicadas: **Data-to-Text Templates** (injetar JSONs estruturados no prompt), **Role-Playing** (definir personas específicas, como "Analista Financeiro Sênior"), **Constraints rígidas** (instruções explícitas sobre o que NÃO gerar, como "não invente métricas") e **Few-Shot Learning** (fornecer exemplos de relatórios anteriores bem-sucedidos).

Os desafios residuais: **alucinação de dados** (o LLM inventar números para justificar uma narrativa; solução: RAG estrito e validação cruzada), **model drift** (atualizações na API do LLM mudam o estilo; solução: versionamento de prompts e testes de regressão) e **context window** (relatórios anuais excedem o limite de tokens; solução: sumarização hierárquica com Map-Reduce).

O caso real citado é o **fechamento contábil mensal de uma multinacional**: um pipeline extrai dados do ERP, calcula variações percentuais em Python, e usa um LLM para redigir a "Análise de Variação" (MD&A), destacando anomalias e sugerindo planos de ação.

9.2.2 Caso 2: geração de código

Envolve fluxos complexos: geração de testes unitários, refatoração de código legado (por exemplo, COBOL para Java), geração de documentação técnica e criação de boilerplate. O pipeline de CI/CD aumentado tem cinco passos: **especificação** (issue do Jira), **geração** (LLM usa contexto do repositório), **teste automático** (execução imediata de testes unitários), **feedback loop** (se falhar, o erro volta para o LLM corrigir) e **integração** (PR automático para revisão humana).

Prompt engineering para código: **Repository Context** (incluir AST ou assinaturas de funções relevantes no prompt), **Step-by-Step** (pedir ao LLM que planeje a arquitetura antes de escrever o código) e **Style Guidelines** (injetar regras de linting, como PEP 8, no prompt).

Os riscos: **vulnerabilidades** (LLMs podem reproduzir código inseguro presente nos dados de treino), **débito técnico** (geração rápida pode levar a arquiteturas mal planejadas) e **alucinação de APIs** (o modelo inventa bibliotecas ou métodos que não existem).

O caso real é o **GitHub Copilot Workspace**: o sistema analisa a issue, busca arquivos relevantes no repositório via RAG, propõe um plano de ação, gera o código, executa testes em sandbox e apresenta um Pull Request completo.

9.2.3 Caso 3: revisão de contratos

O pipeline: **OCR/extração** (PDF para texto estruturado), **segmentação** (divisão do contrato em cláusulas lógicas), **análise** (classificação de risco por cláusula usando RAG) e **relatório** (dashboard com red flags e sugestões de redigitação).

Prompt engineering para análise legal: **RAG jurídico** (injetar jurisprudência e políticas internas no prompt), **JSON output** (forçar resposta estruturada, por exemplo `{"clausula": "...", "risco": "Alto", "motivo": "..."}`) e **few-shot com casos limites** (mostrar exemplos de cláusulas ambíguas e como interpretá-las).

Os desafios: **responsabilidade legal** (quem é culpado se a IA aprovar uma cláusula prejudicial? — human-in-the-loop é obrigatório), **nuances linguísticas** (termos jurídicos variam drasticamente por jurisdição) e **falsos positivos** (excesso de alertas causa fadiga nos advogados revisores).

O caso real de compliance: uma **fusão e aquisição exigindo revisão de 10.000 contratos de fornecedores**. O pipeline processa todos os PDFs, extrai cláusulas de rescisão e mudança de controle, e gera uma planilha consolidada destacando apenas os **5% de contratos que apresentam risco real** para a operação.

9.2.4 Casos 4 e 5: engenharia offshore e sistemas industriais

Em **equipes globais distribuídas** (Brasil, Índia, EUA), as barreiras são comunicação, fuso horário e cultura. A arquitetura proposta é um **knowledge graph dinâmico**: ingestão (Slack, Jira, Confluence, transcrições de Zoom), vetorização (embeddings do conhecimento da empresa), agente de suporte (bot que responde dúvidas técnicas em tempo real) e tradução automática (adaptação de PRs e issues para o idioma local). O caso real é o **onboarding de 50 desenvolvedores offshore em um mês**, com um agente treinado em todo o histórico do repositório que gera tutoriais passo a passo baseados no estado atual do código.

Os desafios: **perda de contexto tácito** (conhecimento não documentado não pode ser acessado pelo LLM), **latência de atualização** (a base RAG precisa acompanhar os commits) e **segurança de dados** (vazamento de propriedade intelectual em prompts para APIs públicas).

Em **sistemas industriais**, a integração de LLMs com **OT (Operational Technology)** é apresentada como a fronteira da automação. O pipeline edge-to-cloud: sensores (edge) coletam vibração e temperatura, um gateway agrega e filtra ruído, a análise no cloud/LLM interpreta anomalias cruzando dados de sensores com manuais de manutenção, e a ação gera ordens de serviço automáticas.

Técnicas específicas: **Time-Series to Text** (converter dados de sensores em descrições textuais antes de enviar ao LLM), **Tool Calling / Function Calling** (permitir que o LLM execute scripts Python para consultar bancos industriais) e **manuais técnicos no contexto** (RAG focado em PDFs de especificações de engenharia).

O caso real de manutenção preditiva: uma turbina eólica apresenta vibração anômala; o sistema detecta, o LLM consulta o manual e o histórico, e gera um relatório para o técnico ("vibração no rolamento X sugere desgaste; recomenda-se inspeção visual e substituição da peça Y; procedimento na página 42 do manual").

Os desafios críticos: **latência** (LLMs na nuvem são lentos demais para controle em tempo real, o que exige SLMs no Edge), **segurança física** (uma alucinação não pode resultar em comando perigoso para uma máquina) e **conformidade** (necessidade de auditoria estrita de todas as decisões geradas por IA).

A conclusão do bloco: a automação com prompts está evoluindo de pipelines estáticos para **Sistemas Multi-Agentes**, com agentes especializados colaborando entre si — um analisa dados, outro escreve código, um terceiro revisa a segurança. O papel do engenheiro humano muda de "executor" para **"orquestrador e revisor de sistemas autônomos"**.

9.3 RLHF, PPO, DPO e GRPO

9.3.1 Componentes de RL aplicados a LLMs

O mapeamento apresentado nos slides:

- **Agente**: o modelo de linguagem que toma decisões.
- **Ambiente**: contexto e prompts do usuário.
- **Ação**: tokens gerados pelo modelo.
- **Recompensa**: sinal avaliando a qualidade da resposta.
- **Política**: distribuição de probabilidade sobre ações.
- **Valor**: estimativa de recompensa futura esperada.

9.3.2 As cinco etapas do treinamento com RLHF

1. **Pré-treinamento do LM base** em dados de linguagem natural.
2. **Coleta de preferências**: humanos comparam pares de respostas.
3. **Treinamento do Reward Model (RM)**: o modelo aprende a prever preferências humanas.
4. **Otimização com RL**: ajuste fino do LM usando recompensas do RM.

5. **Modelo alinhado**: resultado final que segue instruções e valores humanos.

O **reward model** é treinado em pares de respostas nas quais anotadores humanos indicam qual é melhor, e aprende a **generalizar** preferências para novas respostas — fornecendo sinais contínuos de qualidade **sem necessidade de anotação humana adicional a cada iteração**.

9.3.3 Desafios do RLHF tradicional

- **Instabilidade de treinamento**: a otimização pode divergir ou colapsar.
- **Reward hacking**: o modelo aprende a explorar falhas no reward model em vez de melhorar a qualidade real.
- **Custo computacional**: requer múltiplos modelos em memória simultaneamente (Policy, Reference, Reward, Value).
- **Qualidade do reward model**: erros na avaliação propagam-se e amplificam-se na otimização.
- **Divergência de KL**: o modelo pode se afastar demais da política original, perdendo capacidades gerais.

9.3.4 PPO

O **mecanismo de clipping** limita a razão de probabilidade entre a nova e a antiga política a um intervalo (tipicamente **0.8 a 1.2**), prevenindo ajustes agressivos demais. As vantagens listadas: estabilidade, eficiência (menos sensível a hiperparâmetros que TRPO), escalabilidade, implementação simples e resultados empiricamente comprovados. As desvantagens: requer reward model, custo computacional (múltiplos forward/backward passes), sensibilidade a hiperparâmetros e dependência da qualidade das recompensas.

9.3.5 DPO

O **DPO** elimina a necessidade de um reward model explícito. Usa a **relação de Bradley-Terry** entre preferências e recompensas para derivar uma função de perda que otimiza a política diretamente. A ideia central: se temos pares `(resposta preferida, resposta não preferida)`, é possível calcular a perda sem treinar um reward model separado, permitindo otimização end-to-end mais simples e eficiente.

Vantagens: sem reward model, custo reduzido, otimização direta, estabilidade (menos hiperparâmetros) e escalabilidade para pesquisadores com recursos limitados. Desvantagens: convergência mais lenta, sensibilidade à qualidade dos dados de preferência, menos explorado em produção e complexidade teórica para debugging.

9.3.6 GRPO

O **GRPO (Group Relative Policy Optimization)** combina ideias do DPO com otimização em grupo. Em vez de comparar pares individuais, **agrupa múltiplas respostas e calcula preferências relativas dentro do grupo**. Isso permite que o modelo aprenda não apenas qual resposta é

melhor, mas **compreender gradações de qualidade**, reduzindo ruído de comparações individuais e melhorando generalização. Sua eficiência de memória vem de **dispensar a rede crítica**.

Vantagens: melhor captura de preferências, estabilidade aprimorada (menos sensível a outliers), eficiência comparável ao DPO e flexibilidade no tamanho de grupo. Desvantagens: menos maduro, implementação mais complexa, hiperparâmetros adicionais (tamanho de grupo e ponderação) e necessidade de dados agrupados.

9.3.7 Comparação e critérios de escolha

Aspecto	PPO	DPO	GRPO
Reward Model	Necessário	Não	Não
Custo computacional	Alto	Baixo	Médio
Estabilidade	Muito alta	Média	Alta
Convergência	Rápida	Lenta	Média
Maturidade	Muito alta	Média	Baixa

O guia prático de seleção: escolher **PPO** se há recursos computacionais abundantes, equipe experiente e necessidade de máxima estabilidade em produção; **DPO** se busca eficiência, há dados de preferência de qualidade e é tolerável uma convergência mais lenta; **GRPO** se quer um balanço entre eficiência e estabilidade, com dados estruturados em grupos.

Por cenário: startups e pesquisa (DPO, pelo custo-benefício), produção em larga escala (PPO, pela estabilidade comprovada), recursos limitados (DPO), qualidade máxima (PPO) e inovação/experimentação (GRPO, fronteira de pesquisa).

Quanto à convergência, os slides registram: PPO converge de forma estável mas mais lenta (cerca de 80 iterações até o platô), DPO converge rápido mas com pequenas oscilações (cerca de 75 iterações) e GRPO converge de forma mais rápida e suave (cerca de 70 iterações). A trajetória típica é a mesma nos três: ganho rápido nas primeiras 20 iterações, crescimento mais lento entre 20 e 50, e platô com refinamento fino a partir de 50.

9.3.8 RL em arquiteturas multimodais

Modelos multimodais combinam processamento de imagens com geração de texto. O RL amplifica esses modelos ao permitir **otimização conjunta de compreensão visual e resposta textual**: o modelo aprende não apenas a gerar texto bom, mas a gerar texto **especificamente apropriado para o conteúdo visual fornecido**.

Os componentes da arquitetura:

- **Encoder visual**: extrai características de imagens usando CNN ou Vision Transformer.
- **Encoder de texto**: processa prompts e instruções.
- **Fusion layer**: combina informações visuais e textuais numa representação unificada.
- **LLM base**: gera respostas coerentes e contextualmente apropriadas.
- **Reward model multimodal**: avalia qualidade considerando o contexto visual.

O fluxo de dados: `Imagem + Texto -> Encoders -> Fusion -> LLM -> Resposta`, e depois `Resposta -> Reward Model -> Sinal de Recompensa -> Atualização`.

O **desafio único** deste cenário é a **alucinação visual**: descrever elementos que não existem na imagem. Os demais desafios listados: alinhamento visual-textual, construção de um reward multimodal, custo computacional (múltiplos encoders aumentam o overhead), necessidade de anotadores que entendam contexto visual, e **divergência de modalidades** (o modelo pode focar excessivamente em uma modalidade).

As inovações do **MM-RLHF** registradas nos slides: um **dataset de 120 mil pares de preferência anotados por humanos**, avaliação em **8 dimensões diferentes** de qualidade visual e textual, e um **Critique-Based Reward Model** que **gera explicações antes de pontuar**.

As aplicações práticas: assistentes visuais, geração de legendas, análise de documentos (combinando visão de layouts com compreensão textual), educação visual e acessibilidade (descrever imagens de forma otimizada para usuários com deficiência visual).

Os desafios de fronteira até 2028 apontados: **reward hacking**, **custo computacional** (RL é significativamente mais caro que SFT), **distribuição de dados** (dados fora-de-distribuição prejudicam generalização), **escalabilidade** (difícil aplicar em modelos com trilhões de parâmetros), **alinhamento multiobjetivo** (equilibrar múltiplos valores humanos conflitantes) e **interpretabilidade**.

A recomendação de fechamento: "comece experimentando DPO para eficiência ou PPO para estabilidade, e evolua conforme suas necessidades crescem".

10 Foundation Models

10.1 Estrutura da aula

A aula magna sobre **Foundation Models para Séries Temporais** é dividida em quatro blocos: fundamentos (contextualização, desafios de séries temporais e limites do Deep Learning tradicional), arquiteturas (Transformers, mecanismos de atenção, Informer, Autoformer, PatchTST e FEDformer), modelos (TimesFM, Chronos, MOIRAI, Tiny Time Mixers e Lag-Llama, com estudo comparativo) e produção (pipeline, fine-tuning, Edge AI, monitoramento de drift e cases industriais). O foco de aplicação é indústria, energia e óleo e gás.

10.2 Fundamentos: decomposição e estacionariedade

Uma **série temporal** é uma sequência de pontos de dados coletados em intervalos de tempo sucessivos e tipicamente uniformes. Seus componentes clássicos:

- **Tendência**: a direção de longo prazo da série.
- **Sazonalidade**: flutuações periódicas e previsíveis que se repetem em intervalos regulares (picos diários de energia, vendas de natal).
- **Ciclo**: oscilações de médio e longo prazo sem período fixo, associadas a ciclos econômicos ou de mercado.

- **Ruído / resíduo**: flutuações aleatórias e sem padrão — a incerteza física inevitável de qualquer sistema.

A decomposição pode ser **aditiva** ($Y_t = T_t + S_t + I_t$), quando a sazonalidade é constante e independente da tendência, ou **multiplicativa** ($Y_t = T_t \times S_t \times I_t$), quando a sazonalidade é proporcional à magnitude da tendência.

Uma série é **estacionária** quando suas propriedades estatísticas (média, variância e autocovariância) são constantes ao longo do tempo. Séries não-estacionárias precisam ser diferenciadas para que modelos clássicos como ARIMA funcionem. A **ACF (Autocorrelation Function)** mede a correlação linear entre o valor atual e seus valores passados, ajudando a identificar padrões sazonais e memória de curto e médio prazo.

O argumento central da aula: **modelos clássicos exigem testes estatísticos manuais rigorosos** (ADF para estacionariedade, análise de ACF/PACF para as ordens `p`, `d`, `q`). Os **Foundation Models eliminam essa etapa de pré-processamento manual**, porque seus mecanismos de atenção aprendem dinâmicas não-estacionárias diretamente em escala.

10.3 A revolução paradigmática

A evolução histórica é apresentada em cinco eras: métodos estatísticos clássicos (ARIMA, ETS), Deep Learning inicial (LSTMs e GRUs, que permitiram modelar relações não-lineares complexas), Transformers especializados (Informer, Autoformer), modelos de pré-fundação (DeepAR, TFT) e, hoje, **Foundation Models** treinados em bilhões de pontos temporais de múltiplos domínios.

A mudança de paradigma: em vez de treinar um modelo específico do zero para cada novo conjunto de dados, usam-se **modelos pré-treinados em escala massiva** que realizam previsões de alta qualidade de forma imediata — **zero-shot forecasting**.

O **gargalo do ciclo tradicional** que isso resolve: coletar dados históricos suficientes, tratar valores nulos, treinar um modelo específico, validar, ajustar hiperparâmetros e colocar em produção — e recomeçar do zero para **cada novo sensor, SKU ou domínio**.

Dimensão	Paradigma tradicional	Foundation Models
Abordagem	Um modelo por dataset	Modelo único pré-treinado
Tempo de deployment	Semanas ou meses	Minutos (inferência direta)
Dados escassos	Falha ou overfitting severo	Excelente via zero-shot
Manutenção	Altíssima (retreinar sempre)	Baixíssima (modelo estático ou fine-tuning leve)

10.4 Por que Transformers dominam a previsão moderna

Os desafios de adaptação do Transformer a séries temporais:

- **Ausência de vocabulário natural**: diferente de NLP, séries temporais são formadas por **valores contínuos e numéricos**, sem uma "gramática" explícita óbvia.
- **Complexidade temporal e escala**: variações de escala extremas, tendências não-estacionárias e sazonalidades sobrepostas exigem que a arquitetura seja **invariante a translações e escalas**.

- **O gargalo** $O(L^2)$: a auto-atenção padrão calcula a relação entre todos os pontos da série, e o custo cresce quadraticamente com o comprimento da sequência.

As vantagens: **processamento paralelo** (ao contrário de LSTMs sequenciais, processa todo o histórico simultaneamente) e **dependências de longo alcance** (a atenção conecta diretamente pontos distantes no tempo, sem perda de gradiente).

A **revolução do patching** resolve vários desafios de uma vez: dividir a série em subsegmentos contíguos **reduz o comprimento da entrada de L para L/P** (onde P é o tamanho do patch), mitigando o custo quadrático, e permite **foco local e global** simultaneamente — correlações locais dentro do patch e dependências globais através da atenção entre patches.

10.5 Arquiteturas especializadas

Encoder-Decoder: o Encoder processa a série histórica e aprende uma representação comprimida (vetor de contexto) que captura padrões essenciais — tendência, sazonalidade, ciclos, anomalias. O Decoder recebe essa representação e gera previsões futuras de forma **autoregressiva**, com cada previsão condicionando a próxima.

Informer: resolve o gargalo $O(L^2)$ introduzindo o **ProbSparse Attention**, que seleciona apenas as chaves e queries mais dominantes com base numa medida de esparsidade probabilística, reduzindo a complexidade para $O(L \log L)$. Ideal para cenários industriais com dados de alta frequência (sensores a 1 Hz ou mais) e horizontes de previsão muito longos.

PatchTST: aplica o princípio do Vision Transformer a séries temporais — divide a entrada em **patches contíguos**, cada um linearizado e projetado em um embedding. Isso reduz o comprimento da sequência de milhares para centenas, permitindo **atenção completa sem mecanismos esparsos**. Ao agrupar pontos vizinhos, o modelo induz um **inductive bias local correto** (pontos próximos no tempo são altamente correlacionados). Descrito nos slides como "uma das arquiteturas mais fortes e influentes atualmente".

FEDformer (Frequency Enhanced Decomposition Transformer): em vez de realizar as operações de atenção no domínio do tempo, projeta as séries para o **domínio da frequência**. Usando a Transformada Rápida de Fourier (FFT) ou wavelets, aplica **atenção de frequência esparsa** nos componentes mais relevantes. Excelente para sinais industriais de alta oscilação — vibrações de máquinas, flutuações de pressão em tubulações e comportamento oscilatório de sistemas elétricos.

Modelagem multivariada: em ambientes industriais reais, pressão, temperatura, vazão e vibração não existem no vácuo — são registradas simultaneamente por múltiplos sensores acoplados ao mesmo equipamento. Capturar **correlações cruzadas** é fundamental. Os desafios: correlações cruzadas (mudanças em uma variável precedem alterações em outras), **escalas heterogêneas** (unidades completamente distintas, exigindo normalização robusta) e **assincronia** (sensores com frequências de amostragem diferentes ou falhas de leitura).

10.6 Os cinco Foundation Models

Google TimesFM: modelo de **200 milhões de parâmetros**, treinado em **100 bilhões de pontos temporais** reais e sintéticos. Adota estrutura **decoder-only** (semelhante ao GPT em NLP). A inovação-chave é que **o comprimento do patch de saída é maior que o de entrada**,

permitindo previsões de horizonte longo com pouquíssimos passos autoregressivos. Foi pré-treinado com modelos estatísticos (que ensinam a "gramática" temporal) e dados reais (Google Trends, Wikipedia). Desenvolvido pelo Google Research e aceito no ICML 2024.

Na avaliação sobre o **Monash Forecasting Archive**, o TimesFM em modo zero-shot supera não apenas métodos estatísticos clássicos, mas também **modelos de deep learning especializados treinados diretamente nos dados**:

Modelo	Treinamento	MAE relativo	Zero-shot
ARIMA	Por dataset	1.00 (baseline)	Não
DeepAR	Por dataset	0.92	Não
PatchTST	Por dataset	0.88	Não
Google TimesFM	Nenhum	0.87	Sim

Amazon Chronos: em vez de modificar a arquitetura do Transformer para lidar com números contínuos, **tokeniza a série temporal** para que ela se comporte exatamente como texto. Isso permite treinar arquiteturas de linguagem padrão (como o T5) diretamente em dados temporais, aproveitando todas as inovações desenvolvidas para LLMs. Os passos: **mean scaling** (dividir a série pela média absoluta para remover a escala), **quantização em bins** (mapear valores contínuos para intervalos discretos, criando um "vocabulário" de números) e **tokens especiais** (PAD para valores ausentes e EOS para fim de sequência, idênticos aos de NLP).

O Chronos é disponibilizado sob licença **Apache 2.0**, tornando-se um dos modelos mais reprodutíveis e estendidos pela comunidade e o principal baseline para novas pesquisas. Para treinar em escala, os autores desenvolveram o **KernelSynth**, técnica que usa **Processos Gaussianos com kernels compostos aleatoriamente** para gerar milhões de séries temporais sintéticas realistas.

Salesforce MOIRAI: projetado especificamente para a complexidade e heterogeneidade dos dados industriais e de IoT. Treinado no dataset **LOTSA (Large-scale Open Time Series Archive)**, que reúne mais de **27 bilhões de pontos temporais**. Seus três pilares:

- **Any-Variate**: suporta qualquer número de variáveis de entrada num único modelo.
- **Any-Frequency**: lida nativamente com dados amostrados em segundos, minutos, horas ou dias, eliminando reamostragem forçada que destrói padrões de alta frequência.
- **Any-Horizon**: prevê horizontes curtos ou extremamente longos com o mesmo checkpoint.

O **MOIRAI 2.0** acrescenta **Any-Variate Attention** (trata qualquer número de variáveis como canais independentes, aprendendo correlações cruzadas dinâmicas sem explosão de parâmetros), **patching flexível por canal** (cada canal com seu próprio tamanho de patch) e **suporte nativo a quantis**: em vez de prever um único valor futuro, gera previsões para múltiplos quantis simultaneamente (P10 para cenário pessimista e estoque mínimo de segurança, P50 para o cenário mais provável e planejamento diário, P90 para dimensionamento de capacidade máxima e análise de risco).

IBM Tiny Time Mixers (TTM): uma quebra de paradigma — em vez de Transformers gigantescos, adota uma arquitetura baseada em **MLP-Mixers** extremamente leve, com **menos de 1 milhão de parâmetros** (frequentemente entre 500k e 800k), atingindo ou superando a precisão de modelos 100 vezes maiores. Substitui a atenção quadrática por **projeções lineares alternadas no tempo e nos canais**, o que permite inferência em tempo real com latência na casa dos microssegundos e execução nativa em **Edge AI**. Integrado à família de modelos abertos IBM Granite, o

TTM demonstra que "o design inteligente de arquitetura pode superar a força bruta de parâmetros gigantescos".

Seu valor prático aparece em plataformas FPSO, minas remotas e plantas industriais isoladas, onde a conectividade com a nuvem é instável, cara ou inexistente. Enviar gigabytes de dados de sensores em tempo real para a nuvem é inviável — **a inteligência precisa residir na ponta (edge)**. O TTM permite inferência de milissegundos diretamente em gateways IoT e PLCs industriais, roda em CPUs x86 ou ARM comuns sem GPU, e sustenta latência de inferência abaixo de 5 ms para detecção de anomalias em tempo real.

Lag-Llama: foundation model de código aberto focado em **previsão probabilística** de séries univariadas. Em vez de fornecer apenas uma previsão pontual determinística, **estima os parâmetros de uma distribuição de probabilidade completa** (como Student's t ou Normal) para cada ponto no tempo. Usa **lags** (valores passados) como principal feature de entrada, de forma autoregressiva. É essencial para cenários onde o custo de subestimar ou superestimar é assimétrico.

A justificativa operacional é forte: em indústrias de alto risco, **saber a margem de erro de uma previsão é mais importante do que o número em si**. O exemplo dos slides: "saber que há 10% de chance de faltar energia é muito mais útil para um hospital do que receber uma previsão pontual de que o consumo estará dentro da média". Casos ilustrativos:

- **Supply chain**: em vez de "demanda média de 500 unidades" (com 50% de chance de falta de estoque), "90% de probabilidade da demanda ser menor que 650 unidades", levando à decisão de estocar 650 para garantir nível de serviço de 90%.
- **Geração eólica**: em vez de "geração esperada de 40 MW" (com risco de multas se a geração real for menor), "95% de certeza de gerar pelo menos 25 MW", levando à decisão de comprometer 25 MW no mercado de curto prazo com risco quase zero de multa.
- **Manutenção**: em vez de "falha prevista para daqui a 30 dias", "probabilidade de falha supera 15% no dia 20", levando ao agendamento preventivo para o dia 18.

10.7 Guia de seleção

Requisito-chave	Recomendação
Baixa latência e Edge AI	IBM TTM
Dados multivariados e IoT	Salesforce MOIRAI
Previsão probabilística	Lag-Llama
Zero-shot de alta precisão	Google TimesFM
Pesquisa e reprodutibilidade	Amazon Chronos

A **regra de ouro** enunciada duas vezes nos slides: **sempre comece avaliando o modelo em modo zero-shot**. Se a precisão for insuficiente, avance para estratégias de fine-tuning leve antes de considerar o treinamento de um modelo do zero. E sempre comece com o modelo mais simples e leve que atenda aos requisitos mínimos de infraestrutura.

10.8 Da ideia à produção

O ciclo de vida de um projeto real vai muito além de chamar uma API: envolve **ingestão e limpeza** (coleta de dados de sensores e tratamento de valores faltantes), **alinhamento temporal** (sincronização de múltiplas variáveis numa grade temporal unificada) e **inferência e monitoramento**. O alerta dos slides: "a maior parte do esforço em projetos reais de séries temporais está na engenharia de dados e infraestrutura".

10.8.1 Estratégias de fine-tuning

Estratégia	Custo	Volume de dados	Cenário recomendado
Zero-shot	Nenhum	Nenhum	Novos sensores, SKUs recentes
Linear Probing	Muito baixo	Centenas de pontos	Escalas ou distribuições muito diferentes do pré-treino
PEFT / LoRA	Baixo	Milhares de pontos	Melhor custo-benefício para especialização industrial
Full Fine-tuning	Muito alto	Milhões de pontos	Domínios exóticos e proprietários com abundância de dados

O desafio central do fine-tuning é **adaptar o modelo sem causar o esquecimento catastrófico** dos padrões gerais aprendidos. A vantagem é a **eficiência de dados**: como o modelo já possui representações ricas, o ajuste exige uma fração minúscula dos dados necessários para treinar do zero.

10.8.2 Monitoramento de drift

Diferente de domínios estáticos, o mundo real das séries temporais está em constante mudança: comportamentos de consumidores mudam, sensores se desgastam e o ambiente econômico evolui. Dois fenômenos:

- **Data drift**: mudança na distribuição estatística das variáveis de entrada (a temperatura média ambiente sobe devido ao verão). Métricas de detecção: teste de Kolmogorov-Smirnov, Population Stability Index (PSI) e distância de Wasserstein. Ações corretivas: normalização dinâmica, alinhamento de escala e fine-tuning com dados recentes.
- **Concept drift**: mudança na relação entre entrada e saída (a mesma pressão agora causa uma vibração diferente devido ao desgaste físico). Métricas: ADWIN (Adaptive Windowing), DDM (Drift Detection Method) e monitoramento de erro (MAE/MAPE). Ações: retreinamento completo, adaptação de pesos ou fallback para modelo estatístico local.
- **Anomalias de sensores**: detectadas por taxa de valores faltantes, Z-score dinâmico e Isolation Forest; ação corretiva de imputação de dados e alerta para a equipe de manutenção física.

A regra de ouro de MLOps: "não pergunte **se** o seu modelo vai falhar, mas **quando**".

10.8.3 Cases industriais

Utility elétrica com TimesFM: uma concessionária de energia tinha dificuldade em prever a demanda de carga de curto prazo (24 horas à frente) para suas subestações regionais. Erros geravam subcontratação (multas no mercado de curto prazo) ou supercontratação (desperdício de capital), e modelos estatísticos locais falhavam em capturar frentes frias e ondas de calor. Com TimesFM em modo zero-shot integrado ao fluxo de telemetria, os resultados foram: **MAE de 4.82% para 3.15%** (queda de 34%), tempo de setup de **14 dias para imediato**, custo de despacho de backup de **$1.2M para $850k por mês** (queda de 29%) e economia anual estimada de **$4.2 milhões**.

Poços submarinos com MOIRAI: falhas em bombas centrífugas submersas causam paradas de produção que custam milhões de dólares por dia. Uma operadora integrou dados multivariados de **12 sensores físicos** (pressão de sucção, temperatura do motor, vibração e corrente elétrica) usando MOIRAI em modo zero-shot. Resultados: tempo de antecedência subiu de **24-48 horas para 14 dias**, taxa de falsos alarmes caiu de **18% para menos de 4%**, economia estimada de **$4.2M por poço** ao evitar paradas não planejadas, e deployment imediato no primeiro dia de operação do poço.

Chão de fábrica com TTM: uma multinacional de autopeças enfrentava instabilidade de rede que inviabilizava previsões baseadas em nuvem. A solução foi rodar o TTM localmente em gateways industriais simples sem GPU. Resultados: MAE de **18.4% para 11.2%** (melhoria de 39%), latência de **2.5 segundos por lote para 12 milissegundos por previsão**, custo zero de infraestrutura adicional e **100% de disponibilidade operacional** mesmo offline.

10.9 Honestidade técnica: o que Foundation Models não resolvem

Um dos blocos mais valiosos da aula é o que insiste no **pragmatismo de engenharia**: esses modelos **não são balas de prata**. Em muitos cenários industriais comuns, abordagens estatísticas clássicas ou modelos supervisionados locais ainda entregam maior precisão com uma fração do custo computacional. As limitações declaradas:

- **Alucinações numéricas**: modelos baseados em tokenização (como o Chronos) podem gerar valores fisicamente impossíveis fora do domínio de treino.
- **Caixa preta**: a falta de interpretabilidade matemática direta dificulta a auditoria regulatória e a aceitação por operadores de campo.

Cenário de falha	Por que o FM falha	Alternativa recomendada
Séries curtas	Atenção exige contextos longos	ARIMA / ETS, Croston
Mudanças estruturais	Eventos de cauda única violam o pré-treino	Modelos estruturais (BSTS), regras manuais
Restrição de CPU	Transformers exigem GPUs caras	LightGBM / XGBoost, IBM TTM
Física estrita	Modelos estatísticos ignoram leis físicas	Physics-Informed Neural Networks, filtros de Kalman

A conclusão: "o papel do engenheiro de IA não é usar a tecnologia mais complexa, mas sim a mais adequada e eficiente para resolver o problema de negócio com o menor custo e risco possível".

11 RAG - Geração Aumentada por Recuperação

Material de slides não disponível para esta aula.

O deck registrado para esta aula ("Engenharia de Sistemas RAG Modernos") aparece no material sem texto extraído, e os arquivos de apoio consistem nos notebooks `slm_rag_sql.ipynb` e `masterclass_rag_sql_slm.ipynb`. O que segue é estritamente o que o restante do material da disciplina permite dizer sobre RAG, já que o tema atravessa várias aulas anteriores.

11.1 RAG como pilar de orquestração

Nos slides de automação com prompts, o **RAG** é listado como um dos quatro pilares arquiteturais, dentro da categoria **Orquestração**, ao lado de LangChain e LlamaIndex. Ele não aparece como técnica isolada, mas como o componente que **conecta o modelo generativo a uma base de conhecimento verificável**.

11.2 Funções concretas de RAG registradas no material

- **Antídoto para alucinação de dados**: no caso de relatórios automáticos, o risco identificado é "o LLM inventar números para justificar uma narrativa", e a solução prescrita é explicitamente **"RAG estrito e validação cruzada"**.
- **Classificação de risco em contratos**: no pipeline de revisão legal, a terceira etapa é "análise (LLM): classificação de risco por cláusula **usando RAG**". A técnica associada é o **RAG jurídico**, que injeta jurisprudência e políticas internas no prompt.
- **Base de conhecimento corporativa**: na arquitetura de engenharia offshore, o **knowledge graph dinâmico** é construído por ingestão (Slack, Jira, Confluence, transcrições) seguida de **vetorização** — "criação de embeddings do conhecimento da empresa" — que alimenta um agente de suporte que responde dúvidas técnicas em tempo real.
- **Manuais técnicos no contexto**: em sistemas industriais, o material descreve "RAG focado em PDFs de especificações de engenharia", e o caso da turbina eólica mostra o padrão em ação — o sistema detecta a anomalia, **o LLM consulta o manual e o histórico de manutenção**, e gera um relatório com a referência exata ("procedimento na página 42 do manual").
- **Contexto de repositório em geração de código**: o caso do GitHub Copilot Workspace descreve que "o sistema analisa a issue, **busca arquivos relevantes no repositório (RAG)**, propõe um plano de ação, gera o código, executa testes em sandbox e apresenta um Pull Request completo".

11.3 Desafios de engenharia de RAG apontados no material

Três problemas concretos aparecem nos slides:

- **Latência de atualização**: "a base de conhecimento (RAG) precisa ser atualizada em tempo real com os commits". Uma base defasada produz respostas confiantes e erradas.
- **Perda de contexto tácito**: "conhecimento que não está documentado não pode ser acessado pelo LLM". RAG só recupera o que foi escrito.

- **Segurança de dados**: "vazamento de propriedade intelectual em prompts para APIs públicas". Recuperar documentos internos e enviá-los a um provedor externo é uma decisão de arquitetura com implicações de compliance.

11.4 Padrão canônico

Reunindo os elementos que aparecem dispersos no material, o padrão de um sistema RAG moderno tal como descrito na disciplina:

```
# Padrao RAG descrito no material: vetorizacao -> recuperacao -> geracao restrita
# 1) Ingestao: documentos corporativos (PDFs tecnicos, contratos, repositorio)
# 2) Vetorizacao: embeddings armazenados em um indice vetorial
# 3) Recuperacao: os k trechos mais relevantes para a pergunta
# 4) Geracao: o LLM responde APENAS com base nos trechos recuperados

contexto = recuperar(pergunta, k=5)   # trechos do indice vetorial

prompt = f"""Responda usando SOMENTE o contexto abaixo.
Se a resposta nao estiver no contexto, diga que nao sabe.
Contexto:
{contexto}

Pergunta: {pergunta}"""
```

A instrução de recusa ("se não estiver no contexto, diga que não sabe") é a materialização direta do princípio de **constraints rígidas** enunciado nos slides de engenharia de prompts: instruções explícitas sobre o que **não** gerar.

11.5 RAG combinado com SQL e SLMs

Os notebooks de apoio da aula combinam os três elementos: **SLM**, **RAG** e **SQL**. A lógica arquitetural, coerente com o restante da disciplina, é que o RAG recupera o **esquema e a semântica do banco** (nomes de tabelas, descrições de colunas, exemplos de consultas validadas), o SLM traduz a pergunta em uma consulta e o banco executa o cálculo de forma determinística. É o mesmo princípio do pipeline de relatórios automáticos: **o LLM não calcula, o LLM redige e traduz**.

12 SLMs Multimodais na Prática: do Hugging Face ao Código

Material de slides não disponível para esta aula.

Este tópico consta da ementa oficial mas não possui deck de slides no contexto fornecido. O que o material da disciplina permite estabelecer sobre ele:

12.1 Por que SLMs

O argumento econômico e físico é construído nos slides de sistemas industriais: **"LLMs na nuvem são muito lentos para controle em tempo real (necessidade de SLMs no Edge)"**. A mesma lógica aparece de forma quantificada na aula de Foundation Models, no caso do IBM TTM: latência de **2.5 segundos por lote** com solução em nuvem contra **12 milissegundos por previsão** com modelo local, e **100% de disponibilidade operacional** mesmo com queda completa de internet.

A lição transferível é arquitetural, não específica de séries temporais: **o design inteligente de arquitetura pode superar a força bruta de parâmetros gigantescos**, e a escolha do modelo deve ser guiada por onde ele vai rodar, qual a latência aceitável e qual o custo de indisponibilidade.

12.2 Multimodalidade: a arquitetura de referência

A aula de RL multimodal fornece a arquitetura canônica de um modelo visão-linguagem, aplicável tanto a modelos grandes quanto a SLMs:

- **Encoder visual**: extrai características de imagens usando CNN ou Vision Transformer.
- **Encoder de texto**: processa prompts e instruções.
- **Fusion layer**: combina informações visuais e textuais numa representação unificada.
- **LLM base**: gera respostas coerentes e contextualmente apropriadas.

O fluxo: `Imagem + Texto -> Encoders -> Fusion -> LLM -> Resposta`.

O **desafio único** desses modelos, registrado explicitamente, é a **alucinação visual**: descrever elementos que não existem na imagem. Um risco relacionado é a **divergência de modalidades** — o modelo focar excessivamente em uma delas.

12.3 Hugging Face como ponto de acesso

O Hugging Face aparece no material como a plataforma de referência para acessar modelos abertos. Na aula de sistemas de geração de imagens, `huggingface.co/spaces` é indicado como o primeiro endereço das atividades práticas, para testar FLUX.1 [schnell] e Stable Diffusion 3, e listado como plataforma "para profissionais: acesso direto aos modelos mais novos". Na aula de Foundation Models, o material observa que implementar em produção "exige um pipeline estruturado que vai muito além de simplesmente chamar uma API ou carregar um checkpoint do Hugging Face" — reconhecendo o Hugging Face como o mecanismo padrão de distribuição de checkpoints.

12.4 Aplicações registradas

As aplicações de modelos multimodais listadas no material: assistentes visuais (responder perguntas sobre imagens), geração de legendas, análise de documentos (combinar visão de layouts com compreensão textual), educação visual e **acessibilidade** (descrever imagens de forma otimizada para usuários com deficiência visual).

13 Síntese da Disciplina

A disciplina Deep Generative Learning tem uma arquitetura interna bastante coerente, organizada em torno de duas grandes perguntas encadeadas: **como gerar?** e **como controlar o que se gera?**

A primeira metade responde à primeira pergunta por acumulação histórica. As **GANs** mostraram que é possível aprender uma distribuição complexa sem rótulos, por competição — mas ao custo de instabilidade e de um espaço latente opaco. Os **VAEs** trocaram parte do realismo por estrutura, ensinando que uma representação bem organizada permite não apenas gerar, mas **navegar** e **interpolar**. Os **Modelos de Difusão** reconciliaram as duas virtudes com uma reformulação decisiva: **não aprender a gerar, e sim aprender a remover ruído**, decompondo um problema impossível em centenas de problemas triviais com um objetivo de treinamento bem definido. Os **Diffusion Transformers** completaram a linha ao substituir a U-Net por atenção global, trazendo escalabilidade previsível, multimodalidade nativa e o mecanismo elegante do **AdaLN-Zero**, que começa neutro e aprende gradualmente a incorporar o condicionamento.

A segunda metade responde à segunda pergunta com um instrumento diferente: **Aprendizado por Reforço**. A partir do MDP, do Q-Learning e do dilema exploration-exploitation, a disciplina mostra como o mesmo formalismo se traduz para LLMs — o agente é o modelo, o ambiente é o usuário e o reward model, o estado é o contexto, a ação é o próximo token e a recompensa é a nota do juiz treinado sobre preferências humanas. O **RLHF** com **PPO** foi o pipeline que criou o ChatGPT; **DPO** o simplificou eliminando o reward model explícito; **GRPO** ganhou eficiência de memória dispensando a rede crítica e passando a comparar grupos em vez de pares. A extensão multimodal (**MM-RLHF**) adiciona encoders visuais, fusão e um reward model que avalia imagem e texto conjuntamente — enfrentando o problema específico da alucinação visual.

Há duas conexões transversais que atravessam quase todas as aulas.

A primeira é **onde reside o controle**. Na GAN, ele praticamente não existe. No VAE, ele vem da estrutura do espaço latente. Na difusão condicionada, vem do texto, injetado via cross-attention ou AdaLN. No RLHF, vem da preferência humana codificada num sinal numérico. Em cada caso, a estratégia é a mesma: **transformar uma intenção qualitativa num sinal que o gradiente possa usar**.

A segunda é o **pragmatismo de engenharia**, mais explícito nas aulas aplicadas. Difusão no espaço de pixels é caro demais, então o Stable Diffusion comprime com um VAE. Atenção quadrática é cara demais, então surgem patching, ProbSparse e Flash Attention. Transformers gigantes não cabem em um gateway industrial, então o IBM TTM usa MLP-Mixers com menos de 1 milhão de parâmetros. LLMs não sabem calcular, então o pipeline calcula em Python e o modelo apenas redige. LLMs inventam fatos, então o RAG ancora a geração em documentos recuperáveis. A aula de Foundation Models formula o princípio de maneira definitiva: **o papel do engenheiro de IA não é usar a tecnologia mais complexa, mas a mais adequada e eficiente para resolver o problema de negócio com o menor custo e risco possível**.

Fica também um conjunto de questões abertas que a disciplina não fecha, e faz bem em não fechar: quem define os "valores humanos" usados no alinhamento; como evitar **reward hacking** sem enrijecer o modelo; como equilibrar objetivos conflitantes; e como auditar decisões de modelos que permanecem, em grande medida, caixas pretas. Essas perguntas conectam o conteúdo técnico da disciplina ao debate mais amplo sobre o uso responsável de sistemas generativos.