# Frontier AI Technical Handbook

Generative Models, World Models, Reasoning & Post-Training,
Systems Engineering, MoE, Multimodal, and Test-Time Compute

Qi Guo

https://qig.github.io

2025–2026

# Contents

## Module 1: The Generative Engine (Enhanced)

**Core Focus:** From Diffusion Foundations to Flow Matching; Solving the Context Bottleneck.

### 1.0 Diffusion Foundations: The Theory You Must Know Before Flow Matching

**1.0.0 The ELBO Foundation: From VAEs to Diffusion Models Why This Matters:** The Evidence Lower Bound (ELBO) is the unifying principle behind both VAEs and diffusion models. Understanding ELBO deeply reveals that diffusion models are a special case of hierarchical VAEs — this connection is worth understanding deeply.

**The Evidence Lower Bound (ELBO):**

For any latent variable model with observed $x$ and latent $z$:

$$\log p(x) = \underbrace{\mathbb{E}_{q(z|x)}\left[\log \frac{p(x,z)}{q(z|x)}\right]}_{\text{ELBO } \mathcal{L}(x)} + \underbrace{D_{\text{KL}}(q(z|x)\|p(z|x))}_{\geq 0}$$

Since KL divergence is non-negative: $\log p(x) \geq \mathcal{L}(x)$. Maximizing the ELBO simultaneously:

1. Maximizes the **reconstruction term** $\mathbb{E}_{q(z|x)}[\log p_\theta(x|z)]$ — decoder should reconstruct $x$ from $z$.
2. Minimizes the **KL regularization** $D_{\text{KL}}(q_\phi(z|x)\|p(z))$ — encoder should produce $z$ close to the prior.

**VAE (Kingma & Welling, 2013):**

- **Encoder:** $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma^2_\phi(x)I)$ — amortized inference via neural network.
- **Decoder:** $p_\theta(x|z)$ — generates data from latent code.
- **Prior:** $p(z) = \mathcal{N}(0, I)$.
- **Reparameterization trick:** $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$ — enables gradient flow through sampling.
- **VAE ELBO:**

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[\log p_\theta(x|z)\right] - D_{\text{KL}}(q_\phi(z|x)\|p(z))$$

**DDPM as a Hierarchical VAE:**

DDPM can be viewed as a $T$**-step hierarchical VAE** where the "latents" are $x_1, x_2, \ldots, x_T$:

- **Encoder (forward process):** $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$ — **fixed**, not learned. Each step adds Gaussian noise.
- **Decoder (reverse process):** $p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$ — **learned** neural network.
- **Prior:** $p(x_T) = \mathcal{N}(0, I)$ — pure noise.

The DDPM Variational Lower Bound decomposes as:

$$\mathcal{L}_{\text{VLB}} = \underbrace{D_{\text{KL}}(q(x_T|x_0)\|p(x_T))}_{\mathcal{L}_T \text{ (prior matching)}} + \sum_{t=2}^{T} \underbrace{D_{\text{KL}}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t))}_{\mathcal{L}_{t-1} \text{ (denoising matching)}} - \underbrace{\log p_\theta(x_0|x_1)}_{\mathcal{L}_0 \text{ (reconstruction)}}$$

- $\mathcal{L}_T$: Constant (forward process is fixed, no learnable parameters).
- $\mathcal{L}_{t-1}$: Each term compares the true denoising posterior $q(x_{t-1}|x_t, x_0)$ (Gaussian, tractable) with the learned $p_\theta(x_{t-1}|x_t)$. This is where training happens.
- $\mathcal{L}_0$: Reconstruction at the finest level.

**The Simplified Loss Connection:**

Ho et al. showed that the $\mathcal{L}_{\text{simple}} = \mathbb{E}\|\epsilon - \epsilon_\theta(x_t, t)\|^2$ objective is a **reweighted** version of $\mathcal{L}_{\text{VLB}}$ with the SNR-dependent prefactor dropped. This reweighting emphasizes perceptual quality over exact likelihood.

**VAE vs. Diffusion — Key Differences and Commonalities:**

| Aspect | VAE | Diffusion (DDPM) |
|---|---|---|
| Latent structure | Single latent $z$ | Hierarchy $x_1, \ldots, x_T$ |
| Encoder | Learned $q_\phi(z \mid x)$ | Fixed $q(x_t \mid x_{t-1})$ |
| Decoder | Learned $p_\theta(x \mid z)$ | Learned $p_\theta(x_{t-1} \mid x_t)$ |
| Prior | $\mathcal{N}(0, I)$ | $\mathcal{N}(0, I)$ at $t = T$ |

| Aspect | VAE | Diffusion (DDPM) |
|---|---|---|
| Training objective | ELBO (reconstruction + KL) | Reweighted VLB (simplified to MSE) |
| Latent dimension | Low-dimensional bottleneck | Same dimension as data |
| Generation quality | Blurry (single-step decode) | High quality (iterative refinement) |
| Inference cost | Single forward pass | $T$ forward passes (10-1000 steps) |
| Posterior collapse | Common failure mode | Not applicable (encoder is fixed) |
| Likelihood | Tractable bound (loose) | Tractable bound (tighter with $T$ terms) |

**Why Diffusion Outperforms VAE:**

1. **No amortization gap:** VAE's $q_\phi(z|x)$ is a single neural network approximating all posteriors — this approximation is lossy. Diffusion's forward process is *exact* (no learning needed for the encoder).
2. **Hierarchical refinement:** $T$ denoising steps allow progressive detail addition — coarse structure first, fine details last. VAE must generate everything in one shot.
3. **Same-dimensional latent:** VAE compresses to a bottleneck, losing information. Diffusion latents are the same size as data — no information bottleneck.
4. **Stable training:** No posterior collapse, no KL vanishing, no decoder ignoring latents.

**Latent Diffusion Models (Rombach et al., 2022) — The Best of Both Worlds:**

- First train a VAE to compress images: $z = \text{Enc}(x)$, $\hat{x} = \text{Dec}(z)$.
- Then train a diffusion model in the *latent space* of the VAE.
- This combines VAE's compression efficiency with diffusion's generation quality. **Stable Diffusion, FLUX, DALL-E 3** all use this architecture.

**Think Deeper:** "Derive the DDPM variational lower bound starting from $\log p(x_0)$. Show how the simplified loss $\|\epsilon - \epsilon_\theta\|^2$ relates to each $\mathcal{L}_{t-1}$ term, and explain what information is lost by dropping the SNR weighting." (Answer: Each $\mathcal{L}_{t-1} = D_{\text{KL}}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t))$. Since both are Gaussian, this KL has closed form proportional to $\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\epsilon - \epsilon_\theta\|^2$. The SNR-dependent prefactor $\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}$ upweights low-noise (detail) steps. Dropping it with $\mathcal{L}_{\text{simple}}$ uniformly weights all noise levels, which empirically improves FID but worsens NLL.)

### 1.0.1 The Forward & Reverse SDE Framework

- **Core Idea:** Gradually destroy data with noise (forward process), then learn to reverse it (reverse process).
- **Forward SDE (data -> noise):**

$$dx_t = f(x_t, t)\,dt + g(t)\,dW_t$$

  - $f(x_t, t)$: drift coefficient (deterministic decay toward zero).
  - $g(t)$: diffusion coefficient (noise injection rate).
  - $W_t$: standard Wiener process (Brownian motion).
- **Reverse SDE (noise -> data, Anderson 1982):**

$$dx_t = \left[f(x_t, t) - g(t)^2 \nabla_{x_t} \log p_t(x_t)\right] dt + g(t)\,d\bar{W}_t$$

  - The **score function** $\nabla_{x_t} \log p_t(x_t)$ is the only unknown — this is what we train a neural network $s_\theta(x_t, t)$ to approximate.
- **Probability Flow ODE (deterministic counterpart):**

$$dx_t = \left[f(x_t, t) - \frac{1}{2} g(t)^2 \nabla_{x_t} \log p_t(x_t)\right] dt$$

  - Same marginals $p_t(x)$ as the SDE but **no stochasticity** — enables exact likelihood computation via the instantaneous change-of-variables formula and deterministic sampling.

### 1.0.2 DDPM (Ho et al., 2020)

- **The Landmark Paper:** Made diffusion models practical for image generation.

- **Forward Process (discrete):** A Markov chain adding Gaussian noise over $T$ steps:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1},\, \beta_t I)$$

  - Noise schedule $\{\beta_t\}_{t=1}^T$, typically $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$, $T = 1000$.
  - Closed-form jump to any $t$: $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0,\, (1-\bar{\alpha}_t)I)$, where $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$.
  - Reparameterization: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$.
- **Reverse Process (learned):**

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t),\, \sigma_t^2 I)$$

- **Training Objective (simplified):**

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t, x_0, \epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

  - Sample $t \sim \text{Uniform}(1, T)$, sample $\epsilon \sim \mathcal{N}(0, I)$, construct $x_t$, predict $\epsilon$.
  - This is a **reweighted variational bound** — dropping the SNR-dependent weighting simplifies training and improves sample quality.
- **Three Equivalent Parameterizations:**

| Parameterization | Network predicts | Relationship |
|---|---|---|
| $\epsilon$-prediction | noise $\epsilon$ | $\epsilon_\theta(x_t, t) \approx \epsilon$ |
| $x_0$-prediction | clean data $x_0$ | $x_{0,\theta} = (x_t - \sqrt{1-\bar{\alpha}_t}\, \epsilon_\theta)/\sqrt{\bar{\alpha}_t}$ |
| $v$-prediction | velocity $v$ | Interpolation; better for high-res and video |

- $v$-prediction (Salimans & Ho, 2022): $v = \sqrt{\bar{\alpha}_t}\, \epsilon - \sqrt{1-\bar{\alpha}_t}\, x_0$. This is the bridge to Flow Matching — it directly predicts the direction of travel.
- **Think Deeper:** "Derive the relationship between $\epsilon$-prediction and score matching. Show that $\epsilon_\theta(x_t, t) = -\sqrt{1-\bar{\alpha}_t}\, s_\theta(x_t, t)$." (Answer: By Tweedie's formula, $\nabla_{x_t} \log p(x_t) = \frac{\sqrt{\bar{\alpha}_t} x_0 - x_t}{1-\bar{\alpha}_t} = \frac{-\epsilon}{\sqrt{1-\bar{\alpha}_t}}$.)

#### 1.0.3 DDIM (Song et al., 2020)

- **Key Innovation:** Non-Markovian reverse process that **shares the same training objective** as DDPM but allows deterministic, fewer-step sampling.
- **DDIM Update Rule:**

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \underbrace{\left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\, \epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}\right)}_{\text{predicted } x_0} + \sqrt{1-\bar{\alpha}_{t-1} - \sigma_t^2} \cdot \epsilon_\theta(x_t, t) + \sigma_t\, \epsilon_t$$

  - When $\sigma_t = 0$: fully **deterministic** (ODE). Enables: exact inversion, latent interpolation, and semantic editing.
  - When $\sigma_t = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}\sqrt{\beta_t}$: recovers DDPM (stochastic).
- **Strided Sampling:** Use a subsequence $\{t_1, t_2, ..., t_S\} \subset \{1, ..., T\}$ with $S \ll T$ (e.g., $S = 50$ instead of $T = 1000$). Same trained model, 20x faster inference.
- **Think Deeper:** "DDIM is often called the 'probability flow ODE' of DDPM. Make this connection precise." (Answer: In the continuous-time limit, DDIM with $\sigma_t = 0$ converges to the probability flow ODE $dx = [f(x,t) - \frac{1}{2} g^2(t) \nabla_x \log p_t(x)]dt$.)

#### 1.0.4 Score Matching & Score-Based Models (Song & Ermon, 2019-2021)

- **Score Function:** $s(x) = \nabla_x \log p(x)$ — the gradient of the log-density.
- **Denoising Score Matching (DSM):**

$$\mathcal{L}_{\text{DSM}} = \mathbb{E}_{t, x_0, x_t}\left[\|s_\theta(x_t, t) - \nabla_{x_t} \log p(x_t|x_0)\|^2\right]$$

  - For Gaussian perturbation: $\nabla_{x_t} \log p(x_t|x_0) = \frac{x_0 - x_t}{\sigma_t^2}$. No need to know $p(x)$!
- **Unified Framework (Song et al., 2021 "Score-Based Generative Modeling through SDEs"):**
  - DDPM, SMLD (Score Matching with Langevin Dynamics), and continuous-time diffusion are all instances of the same SDE framework with different choices of $f$ and $g$.
  - **VP-SDE** (Variance Preserving, = DDPM): $f(x,t) = -\frac{1}{2}\beta(t)x$, $g(t) = \sqrt{\beta(t)}$.
  - **VE-SDE** (Variance Exploding, = SMLD): $f(x,t) = 0$, $g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$.

#### 1.0.5 EDM & EDM2 (Karras et al., 2022 & 2024)

- **EDM ("Elucidating the Design Space"):** The definitive engineering paper for diffusion. Systematically disentangles design choices.
- **Key Contributions:**
  - **Preconditioning:** Decomposes the network as:

$$D_\theta(x;\sigma) = c_{\text{skip}}(\sigma) \cdot x + c_{\text{out}}(\sigma) \cdot F_\theta(c_{\text{in}}(\sigma) \cdot x;\, c_{\text{noise}}(\sigma))$$

    * $c_{\text{skip}}, c_{\text{out}}, c_{\text{in}}, c_{\text{noise}}$ are analytically derived scaling functions of $\sigma$ that ensure the network $F_\theta$ always operates on unit-variance inputs and produces unit-variance outputs, regardless of noise level.
    * This simple insight dramatically improves training stability.
  - **Noise Schedule:** Log-normal distribution $\ln\sigma \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ during training, with $P_{\text{mean}} = -1.2$, $P_{\text{std}} = 1.2$ as defaults.
  - **Loss Weighting:** $\lambda(\sigma) = (c_{\text{out}}(\sigma))^{-2}$ — derived to equalize per-sigma gradient magnitudes.
  - **Continuous-time formulation:** $\sigma(t) = t$ (noise level = time), simplifying the ODE to:

$$dx = \frac{x - D_\theta(x;\sigma)}{\sigma} d\sigma$$

- **EDM2 (Karras et al., 2024):** Scaling EDM to larger models.
  - **Weight normalization:** Forces all weight matrices to have unit spectral norm, controlling gradient magnitudes across depths.
  - **Adaptive loss weighting:** Adjusts $\lambda(\sigma)$ during training based on observed loss-per-sigma statistics.
  - **Dropout scheduling:** Applies dropout only in late training to combat memorization without hurting early learning.
  - Result: First diffusion model to achieve **FID 1.58** on ImageNet 512x512 (class-conditional).

#### 1.0.6 Popular Samplers: A Practical Guide

- **Euler (1st-order ODE):** $x_{t-1} = x_t + (t_{i-1} - t_i) \cdot \frac{x_t - D_\theta(x_t, \sigma_t)}{\sigma_t}$. Fast but noisy at low step counts.
- **Heun (2nd-order, "Improved Euler" / EDM default):** Predictor-corrector. Two model evaluations per step but far more accurate. EDM's recommended sampler.
  - Predict: $\hat{x}_{t-1}$ via Euler. Correct: average the derivatives at $t$ and $t-1$.
- **DPM-Solver / DPM-Solver++ (Lu et al., 2022):** Exponential integrator exploiting the semi-linear structure of the diffusion ODE. Achieves high quality in 10-20 steps.
  - DPM-Solver++: Formulated in the $x_0$-prediction space, more numerically stable.
- **UniPC (Zhao et al., 2023):** Unified predictor-corrector framework. Combines benefits of DPM-Solver++ and multi-step methods. State-of-the-art at 5-10 steps.
- **Ancestral Samplers (stochastic):** Add noise at each step (like DDPM). Better diversity but slower convergence. Variants: Euler-a, DPM2-a.
- **Karras Noise Schedule (EDM):** $\sigma_i = (\sigma_{\max}^{1/\rho} + \frac{i}{N-1}(\sigma_{\min}^{1/\rho} - \sigma_{\max}^{1/\rho}))^\rho$, with $\rho = 7$. Concentrates steps at low noise levels where detail emerges.
- **Practical Heuristic (2025):**

| Steps | Recommended Sampler |
|---|---|
| 1-4 | Consistency Model / Distilled (not a classical sampler) |
| 5-10 | UniPC or DPM-Solver++ |
| 10-25 | DPM-Solver++ or Heun (EDM) |

| Steps | Recommended Sampler |
|---|---|
| 25-50 | Heun (EDM) or Euler-a for diversity |
| 50+ | Diminishing returns; use fewer steps with better sampler |

- **Think Deeper:** "Explain the relationship between ODE solvers for diffusion and classical numerical methods. Why does Heun's method (RK2) work better than Euler for diffusion sampling, and what is the error order?" (Answer: Euler is $O(h)$ local error, $O(h^0)$ global. Heun is $O(h^2)$ local, $O(h)$ global. For diffusion ODEs, the curvature of the trajectory near $t = 0$ (data distribution) is high, so higher-order methods dramatically reduce discretization artifacts at low step counts.)

#### 1.0.7 Classifier-Free Guidance (CFG) (Ho & Salimans, 2022)

- **The Most Important Trick in Production Diffusion.**
- **Idea:** Train a single model with both conditional and unconditional objectives (randomly drop conditioning with probability $p_{\text{uncond}} \approx 0.1$). At inference, extrapolate away from the unconditional prediction:

$$\hat{\epsilon}_\theta(x_t, t, c) = \epsilon_\theta(x_t, t, \emptyset) + w \cdot (\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \emptyset))$$

  - $w = 1$: standard conditional generation. $w > 1$: amplifies conditioning signal. Typical: $w \in [5, 15]$.
  - **Geometric Interpretation:** Moves the sample toward high $p(c|x)$ regions in data space. Equivalent to sampling from $\tilde{p}(x|c) \propto p(x|c) \cdot p(c|x)^{w-1}$.
- **Trade-off:** Higher $w$ -> better prompt adherence but lower diversity and potential artifacts (oversaturation). This is why modern models (FLUX, SD3) explore **guidance distillation** to bake CFG into the model itself.
- **CFG in Flow Matching:** Same formula applies to the velocity field: $\hat{v}_\theta(x_t, t, c) = v_\theta(x_t, t, \emptyset) + w \cdot (v_\theta(x_t, t, c) - v_\theta(x_t, t, \emptyset))$.
- **Think Deeper:** "CFG doubles inference cost (two forward passes). How does guidance distillation eliminate this? What loss function would you use?" (Answer: Train student $s_\theta$ to match $\hat{\epsilon}_{\text{teacher}}(x_t, t, c; w)$ with a single forward pass. Loss: $\|s_\theta(x_t, t, c) - \hat{\epsilon}_{\text{teacher}}\|^2$. FLUX uses this approach.)

#### 1.0.8 Self-Attention & Cross-Attention in Generative Models

- **Self-Attention (Vaswani et al., 2017):** Each token attends to all other tokens in the same sequence.

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad Q = XW_Q,\ K = XW_K,\ V = XW_V$$

  - Complexity: $O(N^2 \cdot d)$ time, $O(N^2)$ memory. This is the bottleneck for long sequences (video, high-res images).
  - **In DiT/UNet:** Self-attention over spatial tokens lets the model learn global structure (object coherence, lighting consistency).
- **Cross-Attention:** Query from one modality, Key/Value from another.

$$\text{CrossAttn}(Q_x, K_c, V_c) = \text{softmax}\left(\frac{Q_x K_c^T}{\sqrt{d_k}}\right)V_c$$

  - $Q_x$: from image/video latents. $K_c, V_c$: from text encoder (CLIP/T5).
  - **This is how text controls image generation.** Each spatial position queries the text to decide "what should I generate here?"
- **Multi-Head Attention (MHA):** Split $d$ into $h$ heads of dimension $d_k = d/h$. Each head learns different attention patterns (one for color, one for edges, one for semantics, etc.).
- **Softmax Complexity & Optimization:**
  - Naive: $O(N^2)$ memory for the attention matrix. For $N = 16384$ (512x512 image at patch-2): 1GB just for attention weights in FP32.
  - **FlashAttention:** Tiled computation in SRAM. Never materializes the $N \times N$ matrix. $O(N)$ memory, same $O(N^2)$ FLOPs but 2-4x wall-clock speedup due to memory hierarchy.

  - **Linear Attention** (Katharopoulos et al.): Replace softmax with kernel $\phi$: Attn $= \phi(Q)(\phi(K)^T V)$. $O(N \cdot d^2)$ — linear in $N$ but worse quality.
  - **Sliding Window / Sparse Attention:** Restrict attention to local windows + global tokens. Reduces to $O(N \cdot w)$ where $w$ is window size.
- **Think Deeper:** "In a DiT with both self-attention and cross-attention layers, how do gradients from the diffusion loss flow back to the text encoder? Derive the gradient path and explain why CLIP text encoders are often frozen." (Answer: Gradients flow through $\frac{\partial \mathcal{L}}{\partial K_c} = \text{softmax}^T \cdot \frac{\partial \mathcal{L}}{\partial \text{output}} \cdot V_c^T$ and similarly for $V_c$. Freezing text encoder prevents catastrophic forgetting of the text representation and stabilizes training.)

#### 1.0.9 Exposure Bias & Distribution Drift in Diffusion

- During training, the model sees $x_t$ sampled from the true forward process $q(x_t|x_0)$. During inference, it sees $\hat{x}_t$ from its own imperfect predictions. **Small errors compound across steps.**
  - Formally: training distribution $q(x_t)$ vs. inference distribution $\hat{p}(x_t) = \int p_\theta(x_t|x_{t+1})\hat{p}(x_{t+1})dx_{t+1}$.
  - The mismatch $D_{\text{KL}}(\hat{p}(x_t)\|q(x_t))$ accumulates over $T$ steps — **this is why diffusion has a train-test gap despite operating on noise.**
- **Why "Diffusion Already Has Noise" Doesn't Save You:**
  - True: the forward process adds noise, so the model is trained on noisy inputs.
  - But: $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ assumes the *correct* $x_0$. At inference, the denoised estimate $\hat{x}_0$ has error $\delta$, so the actual input is $\hat{x}_t = \sqrt{\bar{\alpha}_t}(\hat{x}_0 + \delta) + \sqrt{1-\bar{\alpha}_t}\epsilon$. The $\delta$ term is **not noise the model was trained to handle** — it's structured prediction error.
  - This compounds: error at step $t$ shifts $\hat{x}_{t-1}$ further from the training distribution, causing larger error at $t-1$, etc.
- **Mitigations:**
  - **Input Perturbation (Ning et al., 2023):** Add small noise to $x_t$ during training to simulate inference errors: $x_t' = x_t + \sigma_{\text{pert}} \cdot \eta$. Closes the gap without changing the architecture.
  - **Self-Play / Rollout Training:** Sample from the model's own predictions during training. Expensive but directly addresses distribution shift.
  - **Flow Matching reduces drift:** Straighter trajectories -> fewer steps -> less compounding. This is a key practical advantage.
- **Think Deeper:** "Given infinite GPUs, how would you train a diffusion model that has zero train-test gap?" (Answer: Train with online rollouts — at each training step, generate full trajectories from noise using current model parameters, then compute loss against ground truth. This makes training distribution = inference distribution. Equivalent to "self-play" for diffusion. Cost: $T$x more expensive per step, but parallelizable across GPUs. Combined with flow matching (fewer steps) and real-time inference via consistency distillation, this is the ultimate pipeline.)

#### 1.0.10 Data Imbalance in Diffusion Training

- Real datasets are long-tailed. ImageNet has "golden retriever" (1300 images) vs. "platypus" (few). Text-to-image datasets: "woman, portrait" appears 100x more than "isometric factory, blueprint style."
  - Diffusion models **memorize common modes** and **fail on rare concepts**. This is why your DALL-E prompt works for "cat" but not "axolotl wearing a top hat."
- **Noise-Level Imbalance:** Different noise levels $\sigma$ contribute differently to image quality.
  - High $\sigma$: learns global structure (easy, low-frequency). Low $\sigma$: learns fine details (hard, high-frequency).
  - Standard uniform $t$-sampling overweights easy noise levels. EDM's log-normal schedule (Section 1.0.5) partially addresses this.
  - **Min-SNR Weighting (Hang et al., 2023):** $\lambda(t) = \min(\text{SNR}(t), \gamma)/\text{SNR}(t)$. Clips the weight at high SNR to prevent detail-level gradients from dominating.
- **Class/Concept Imbalance Strategies:**
  - **Aspect-Ratio Bucketing + Caption Rebalancing:** Group images by content, upsample rare concepts.
  - **Classifier-Free Guidance as Implicit Rebalancer:** CFG upweights $p(c|x)$ — rare concepts with distinctive features actually benefit *more* from high guidance.
  - **Synthetic Data Augmentation:** Use a teacher model to generate more samples for rare concepts, then retrain. Risk: feedback loops (Section 1.5).
- **Think Deeper:** "You're training a video diffusion model and notice it generates photorealistic humans but fails

on industrial machinery. Diagnose and fix." (Answer: Data imbalance. Fix: (1) rebalance caption sampling, (2) targeted synthetic data from 3D engines like Blender for machinery, (3) multi-stage training — pretrain on balanced data, finetune on target domain with higher learning rate.)

#### 1.0.11 Forward KL vs. Reverse KL in Diffusion & Distillation

- **Forward KL (Mode-Covering):**

$$D_{\mathrm{KL}}(p_{\mathrm{data}} \| p_\theta) = \mathbb{E}_{x \sim p_{\mathrm{data}}} \left[ \log \frac{p_{\mathrm{data}}(x)}{p_\theta(x)} \right]$$

  - Penalizes $p_\theta(x) = 0$ where $p_{\mathrm{data}}(x) > 0$ -> model tries to **cover all modes**. Leads to blurry/averaged outputs.
  - **Used in:** Standard diffusion training (reconstruction loss is forward KL), VAEs.
- **Reverse KL (Mode-Seeking):**

$$D_{\mathrm{KL}}(p_\theta \| p_{\mathrm{data}}) = \mathbb{E}_{x \sim p_\theta} \left[ \log \frac{p_\theta(x)}{p_{\mathrm{data}}(x)} \right]$$

  - Penalizes $p_\theta(x) > 0$ where $p_{\mathrm{data}}(x) = 0$ -> model avoids generating **unrealistic** samples. Leads to sharp but potentially mode-dropped outputs.
  - **Used in:** GAN-like training, DMD (Distribution Matching Distillation), adversarial diffusion distillation.
- **In Distillation (Consistency Models / DMD / DMD2):**
  - **Consistency Distillation (Song et al., 2023):** Forward KL-style. Student learns to map any $x_t$ to $x_0$ directly. Loss: $\|f_\theta(x_{t_{n+1}}) - f_{\theta^-}(x_{t_n})\|^2$. Mode-covering but may be blurry.
  - **DMD (Yin et al., NeurIPS 2024 Oral):** Minimizes approximate KL whose gradient is the difference of two score functions:

$$\nabla_\theta D_{\mathrm{KL}} \approx \mathbb{E}\left[ (s_{\mathrm{fake}}(G_\theta(z)) - s_{\mathrm{real}}(G_\theta(z))) \cdot \nabla_\theta G_\theta(z) \right]$$

    - $s_{\mathrm{real}}$: pretrained diffusion teacher (frozen). $s_{\mathrm{fake}}$: trained on student's outputs. One-step generation matching multi-step teacher quality.
  - **DMD2:** Eliminates the regression loss from DMD, adds two-time-scale update rule for stability. Achieves **FID 1.28** on ImageNet with one-step generation.
- **The "Infinite GPU" Design Question:** "Given unlimited compute, design a training + deployment pipeline for a diffusion model with real-time inference."
  - Training: Flow Matching + online rollout training (Section 1.0.9) + EDM2 preconditioning on $N$ GPUs.
  - Distillation: DMD2 to one-step generator (reverse KL for sharpness).
  - Serving: One-step model behind continuous batching system (Section 3.8).
  - Result: Real-time generation at ~30fps.
- **Think Deeper:** "Forward KL produces blurry samples; Reverse KL drops modes. DMD uses both via a two-score approach. Derive why the gradient of the KL between student and teacher distributions can be estimated using only score functions." (Answer: $\nabla_\theta D_{\mathrm{KL}}(p_\theta \| p_{\mathrm{data}}) = \mathbb{E}_{x \sim p_\theta}[(\nabla_x \log p_\theta(x) - \nabla_x \log p_{\mathrm{data}}(x)) \nabla_\theta x]$. The score of $p_{\mathrm{data}}$ is approximated by the pretrained teacher; the score of $p_\theta$ is approximated by a "fake critic" trained on student outputs.)

### 1.1 From Diffusion to Flow Matching

- Why 2025 is the year of Flow Matching (Rectified Flow) over DDPM.
  - **Concept:** Instead of de-noising a curved trajectory, we learn a velocity field $v_t(x)$ that moves data straight from noise distribution $\pi_0$ to data distribution $\pi_1$.
  - **ODE Solver:** $dx_t = v_t(x_t)dt$.
  - **Target:** Minimize $\mathbb{E}_{t,x_1,x_0}||v_t(x_t) - (x_1 - x_0)||^2$. The target velocity is simply the straight line.
  - **Why it supersedes DDPM:**
    - Straighter trajectories -> fewer ODE steps needed (10-20 vs. 50-1000).
    - Simpler training objective (no noise schedule engineering, no preconditioning gymnastics).
    - Mathematically equivalent to diffusion in the limit, but practically 10-100x faster inference.
    - All major 2025 models (SD3, FLUX, HunyuanVideo, Open-Sora 2.0) use Flow Matching.

  - **Connection to Section 1.0:** Flow Matching's velocity field $v_t$ is the continuous-time generalization of $v$-prediction. The probability flow ODE from DDIM/Score-SDE *is* a flow — Flow Matching just learns a straighter one.
- **Practical Application:** Flow Matching enables real-time video editing with linear interpolation paths — critical for interactive generative tools.

### 1.2 Architecture: Diffusion Transformers (DiT)

- **Why Transformers replaced UNets:** UNets have fixed resolution (downsampling path hardcodes spatial structure), limited scaling behavior, and inductive biases (locality) that restrict global coherence. DiT treats generation as a sequence modeling problem — and Transformers scale.
- **Patchification:** Input latent $z \in \mathbb{R}^{T\times H\times W\times C}$ is split into non-overlapping patches of size $(p_t, p_h, p_w)$. Each patch is linearly projected to a token of dimension $d$:

$$N_{\text{tokens}} = \frac{T}{p_t} \times \frac{H}{p_h} \times \frac{W}{p_w}, \quad x_i = W_{\text{proj}} \cdot \text{flatten}(\text{patch}_i) + b$$

  - Typical: $p = 2$ for images (DiT-XL/2), $(1, 2, 2)$ for video. A 256x256 image in latent space (32x32x4) with $p = 2$ yields $16 \times 16 = 256$ tokens.
- **AdaLN-Zero Conditioning:** Timestep $t$ and class/text conditioning enter via **Adaptive Layer Normalization**:

$$\text{AdaLN}(h, t) = \gamma(t) \cdot \text{LayerNorm}(h) + \beta(t)$$

  where $\gamma(t), \beta(t)$ are predicted from the timestep/condition embedding via an MLP. "Zero" initialization: $\gamma$ initialized to 1, $\beta$ to 0, and an additional scale parameter $\alpha$ initialized to 0 applied before the residual connection — so the DiT block starts as an identity function. This dramatically stabilizes training.
- **Scaling Properties:** DiT scales predictably. DiT-XL/2 (675M params) achieved FID 2.27 on ImageNet 256x256 class-conditional. FID improves log-linearly with model FLOPs — no plateau observed up to XL scale.
- **RoPE for 2D/3D:** Standard 1D RoPE applies rotation matrices to (query, key) pairs: $\text{RoPE}(x, m) = x \cdot e^{im\theta}$. For images, factorize into separate frequencies per axis:

$$\text{RoPE}_{2D}(x, h, w) = x \cdot e^{i(h\cdot\theta_h + w\cdot\theta_w)}$$

  This naturally handles arbitrary aspect ratios and resolutions — the frequency per axis is independent. For video: add a temporal frequency $\theta_t$.
- **MM-DiT (SD3 / FLUX):** Two separate token streams (text + image) with independent linear projections, but **joint self-attention** over concatenated streams. Each stream has its own AdaLN parameters. This allows text-image cross-attention implicitly through the shared attention matrix, while maintaining modality-specific normalization.
- **Think Deeper:** "DiT-XL/2 uses 675M parameters for 256x256. For 1024x1024 (16x more patches), how does compute scale, and what architectural modifications would you make?" (Answer: Naive: $O(N^2)$ attention scales $16^2 = 256$x in FLOPs — prohibitive. Solutions: (1) increase patch size to $p = 4$ (same token count), (2) use windowed/local attention for early layers + global attention for later layers, (3) progressive resolution training. SD3/FLUX use $p = 2$ with FlashAttention and train at multiple resolutions.)

### 1.3 Tokenization: The Discrete-Continuous Gap

- The quality ceiling of any latent generative model is set by its tokenizer. A lossy tokenizer destroys detail that no amount of diffusion/flow training can recover.
- **VQ-VAE (van den Oord et al., 2017):** Encoder outputs continuous $z_e$; quantize to nearest codebook entry:

$$z_q = \arg\min_{e_k \in \mathcal{C}} \|z_e - e_k\|_2$$

  - **Loss:** $\mathcal{L} = \|x - \hat{x}\|^2 + \|\text{sg}[z_e] - z_q\|^2 + \beta\|z_e - \text{sg}[z_q]\|^2$ (reconstruction + codebook + commitment).
  - **Codebook collapse:** In practice, most codes go unused (only 10-20% of a 8192-size codebook is active). Solutions: EMA updates, codebook reset, increased commitment loss $\beta$.

- **MagViT-v2 (Yu et al., 2024):** Lookup-Free Quantization (LFQ). Instead of a learned codebook, use a fixed binary code: each dimension is quantized to $\{-1, +1\}$. With $d$ dimensions, the implicit codebook has $2^d$ entries.
  - **Advantage:** No codebook collapse (all codes are used by construction), no EMA, no auxiliary losses. Training is simpler and more stable.
  - Used in Emu3, VideoPoet. State-of-the-art video tokenizer.
- **FSQ (Mentzer et al., 2024):** Project each dimension of $z$ to a small integer set (e.g., $\{-2, -1, 0, 1, 2\}$) via rounding. With $d$ dimensions and $L$ levels per dimension: implicit codebook size = $L^d$.
  - Example: $d = 6$, levels $[8, 6, 5, 5, 5, 5]$ gives $8 \times 6 \times 5^4 = 30{,}000$ codes.
  - **Straight-Through Estimator (STE):** Forward pass uses round($z$); backward pass passes gradients through as if rounding were identity. Same trick as VQ-VAE but without the codebook lookup.
- **Continuous vs. Discrete Latents:**

| Aspect | Discrete (VQ/FSQ) | Continuous |
|---|---|---|
| Information per token | $\log_2$(codebook size) bits | Float32: 8192 bits/dim |
| Generation method | Autoregressive (next-token) | Diffusion / Flow Matching |
| Quality ceiling | Limited by codebook | Unlimited (continuous) |
| Training simplicity | Simple (cross-entropy) | Complex (noise schedules) |

- **Compression Ratios:** For video, the VAE compresses spatially (8x downsampling per axis typical) and temporally (4x with causal 3D-VAE). A 256x256x16-frame clip: raw = $256^2 \times 16 \times 3 \approx 3.1M$ values $\rightarrow$ latent = $32^2 \times 4 \times 4 \approx 16K$ values. ~**200x compression.**
- **Causal 3D-VAE:** For autoregressive video generation, the temporal convolutions must be causal (only look at past frames). Replaces standard 3D Conv with causal padding: pad $(k_t - 1, 0)$ in the temporal dimension. Used in Open-Sora, CogVideoX.
- **Think Deeper:** "You observe that your video DiT produces sharp individual frames but temporal flickering. Where in the pipeline is the likely failure — tokenizer or DiT — and how do you diagnose it?" (Answer: If the VAE can reconstruct ground-truth video without flicker (encode $\rightarrow$ decode), the tokenizer is fine and the DiT's temporal attention is the issue. If the VAE *itself* introduces flicker, the 3D-VAE's temporal compression is too aggressive or the temporal convolutions are not properly causal. Diagnosis: compute per-frame PSNR of VAE reconstructions vs originals and check temporal variance.)

### 1.4 Context Expansion: Beyond FlashAttention

- Standard DiT fails at long-form video (1min+) due to quadratic memory growth. A 1-min video at 24fps with 512x512 patches produces ~1.5M tokens. Even FlashAttention (which solves *memory*) still requires $O(N^2)$ *compute*.
- **FlashAttention Recap (Dao et al., 2022-2024):**
  - Standard attention materializes the $N \times N$ attention matrix — $O(N^2)$ memory. FlashAttention tiles the computation into SRAM-sized blocks, never materializing the full matrix.
  - **Memory:** $O(N)$ instead of $O(N^2)$. **Speed:** 2-4x faster via reduced HBM reads (IO-aware algorithm). **Exact:** No approximation — mathematically identical output.
  - FlashAttention-3 (2024): Leverages H100 TMA (Tensor Memory Accelerator) and FP8, reaching ~75% of theoretical peak FLOPs. Asynchronous softmax with pipelining.
- **Ring Attention (Liu et al., ICLR 2024):**
  - **Problem:** Even with FlashAttention, a single GPU cannot hold all KV blocks for 1M+ tokens.
  - **Algorithm:** Distribute the sequence across $P$ GPUs. Each GPU holds $N/P$ query blocks. KV blocks circulate in a ring:
    1. Each GPU computes attention for its Q blocks against its *local* KV block.
    2. Send KV block to the next GPU in the ring (async), receive from the previous.
    3. Accumulate attention outputs using the online softmax trick (numerically stable incremental softmax).
    4. After $P$ rounds, every GPU has attended to all KV blocks.
  - **Communication:** Each GPU sends/receives $O(N/P \cdot d)$ per round, $P$ rounds total = $O(N \cdot d)$ total. Overlapped with compute — if compute $\geq$ communication, the overhead is near-zero.
  - **Causal masking:** In causal attention, the lower-triangular mask means some KV blocks have zero contribution to certain Q blocks. Ring Attention skips these blocks entirely — saves up to ~50% of compute for

causal models.

- **Context Parallelism (CP):** The production-grade implementation of Ring Attention in frameworks like Megatron-Core and PyTorch.
  - **Dynamic CP (NVIDIA, 2025):** Handles variable-length sequences within a batch. Pads sequences to minimize waste, then distributes across the CP dimension. Reduces padding overhead from ~40% to ~5% for mixed-length batches.
  - CP composes with TP, PP, DP, EP — forming **5D parallelism**: DP × TP × PP × CP × EP.
- **Sequence Parallelism vs. Context Parallelism:** Sequence parallelism (Megatron-SP) shards the LayerNorm/Dropout activations across the sequence dimension but does *not* shard attention. CP shards attention itself. They are complementary: SP reduces activation memory; CP enables arbitrarily long sequences.
- **RoPE Extrapolation — The Length Generalization Problem:**
  - RoPE encodes position $m$ as a rotation: $f(x, m) = x \cdot e^{im\theta}$, with frequencies $\theta_j = 10000^{-2j/d}$.
  - Training at context length $L_{\text{train}}$, the model never sees positions $m > L_{\text{train}}$. At inference with $m \gg L_{\text{train}}$, high-frequency components rotate to unseen angles — attention patterns degrade.
  - **Position Interpolation (Chen et al., 2023):** Scale positions: $m' = m \cdot \frac{L_{\text{train}}}{L_{\text{target}}}$. All positions stay within the trained range. Requires brief fine-tuning (~1000 steps).
  - **YaRN (Peng et al., 2023):** NTK-aware interpolation — applies different scaling to different frequency bands. Low frequencies (which carry long-range position info) are interpolated more aggressively; high frequencies (local structure) are left unchanged. No fine-tuning needed for moderate extrapolation (4-8x).
  - **Practical:** Llama 3 trained at 8K, extended to 128K via progressive interpolation + continued pretraining. Gemini 3 natively supports 1M context.
- **Think Deeper:** "You need to train a video DiT on 2-minute clips (2880 frames, ~3M tokens). You have 256 H100 GPUs. Design the parallelism strategy for the attention computation." (Answer: CP = 32 (each GPU handles ~94K tokens of Q), TP = 8 (within node for attention head parallelism), DP = 1 (256/32/8 = 1). Ring Attention across the 32 CP ranks with FlashAttention-3 locally. Communication: each ring step sends $94K \times d_{\text{head}}$ bytes — at 900 GB/s NVLink within nodes and 400 Gb/s InfiniBand across, compute dominates. For causal DiT: skip ~50% of KV blocks due to causal mask.)

### 1.5 Data Synthesis & Labeling

- **LMM-based Video Captioning** (e.g., Gemini 3 Flash labeling Sora data at scale).
- **Concept:** "Recaptioning" at scale. High-quality, dense temporal descriptions are the *only* reason modern video models follow prompts well.
- **Think Deeper:** "How do you handle the 'Feedback Loop' bias if you train a video model on data captioned by another model?"

---

## Module 2: The Physical World & Evaluation

**Core Focus:** Moving from "Generative Art" to "World Simulation."

### 2.1 World Models: From Passive Video to Interactive Environments

- World models have moved from "predict the next frame" to "generate controllable, interactive environments."

#### 2.1.1 Genie / Genie 2 / Genie 3 (DeepMind, 2024-2026)

- **Genie (ICML 2024):** The first generative interactive environment model. Three components:
  - **Spatiotemporal Video Tokenizer:** Encodes video frames into discrete latent tokens using a VQ-VAE with 3D convolutions.
  - **Latent Action Model (LAM):** Learns action representations from *unlabeled* video. No ground-truth actions needed — infers what "action" caused frame $t$ to become frame $t+1$ via a VQ bottleneck between consecutive frames.
  - **Dynamics Model:** Autoregressive transformer predicting next-frame tokens conditioned on past frames + latent action. Enables frame-by-frame controllable generation.

  - **Key Insight:** Trained on 200K+ hours of Internet video (platformer games). Learns physics, gravity, collision — all from observation alone.
- **Genie 2 (Dec 2024):** Scaled to a **foundation world model.**
  - **Architecture:** Autoregressive latent diffusion model. Replaces discrete token prediction with continuous latent diffusion for higher fidelity.
  - **Scale:** Trained on large-scale video datasets (not just games). Generates photorealistic 3D environments from a single image prompt.
  - **Capabilities:** Consistent object permanence, realistic lighting, water/smoke physics, character animation. Supports diverse action spaces (walking, jumping, camera rotation).
  - Classifier-free guidance on the latent action for controllability.
- **Genie 3 (Aug 2025 research, Jan 2026 public):** The state-of-the-art interactive world model.
  - **Real-time generation** at 24fps, 720p resolution, minutes of consistent interaction.
  - **Text-to-world:** Generate a full interactive environment from a text description ("a medieval castle with a drawbridge over a moat").
  - **Physics simulation:** Simulates gravity, collisions, fluid dynamics, object interactions — not scripted, but learned from data.
  - **Breakthrough consistency:** Maintains scene coherence over extended interaction (previous models degraded after seconds).
  - DeepMind positions this as a stepping stone toward AGI — a model that understands "how the world works."

#### 2.1.2 Other World Model Paradigms

- **Autoregressive Video (Sora, Open-Sora):** Generate video frame-by-frame or chunk-by-chunk. High fidelity but **not interactive** — the user cannot act in the world.
- **JEPA (LeCun, V-JEPA 2):** Joint Embedding Predictive Architecture. Predicts in latent space, ignoring pixel-level noise. Good for representation learning but currently **not generative** — learns to understand, not to create.
- **NVIDIA Cosmos:** Foundation world models for physical AI. Focus on robotics sim-to-real transfer.
- **Key Comparison:**

| Model | Interactive | Real-time | Learned Physics | Generative |
|---|---|---|---|---|
| Genie 3 | Yes | Yes (24fps) | Yes | Yes |
| Sora / Open-Sora | No | No | Partial | Yes |
| V-JEPA 2 | No | N/A | Yes | No (embeddings) |
| Cosmos | Partial | Yes | Yes | Yes |

- **Practical Application:** The Genie paradigm is directly applicable to physics simulation — learn a latent action model from unlabeled physics data, then use a dynamics model to predict "what happens next" interactively.
- **Think Deeper:** "Genie learns actions from unlabeled video via a VQ bottleneck. Why does this work? What prevents the action space from collapsing to trivial representations?" (Answer: The VQ bottleneck forces the model to learn discrete, information-limited action codes. Combined with the dynamics model's need to predict the next frame *conditioned on the action*, the action must capture the delta between frames. Without the bottleneck, the dynamics model could ignore the action and just auto-regress — the bottleneck prevents this by limiting what information the action can carry.)

### 2.2 Sim-to-Real & Differentiable Physics

#### 2.2.1 The Sim-to-Real Gap

- Training in simulation is cheap (parallel, safe, unlimited data). Deploying in reality fails because simulators approximate physics imperfectly.
- **Two sources of gap:**
  - **Appearance gap:** Rendered images lack real-world texture, lighting, shadows, sensor noise.
  - **Dynamics gap:** Simulated friction, contact, deformation, aerodynamics differ from reality.

- **Formal framing:** Policy $\pi$ trained in simulator MDP $\mathcal{M}_{\text{sim}} = (\mathcal{S}, \mathcal{A}, T_{\text{sim}}, R)$ must transfer to real MDP $\mathcal{M}_{\text{real}} = (\mathcal{S}, \mathcal{A}, T_{\text{real}}, R)$ where $T_{\text{sim}} \neq T_{\text{real}}$. The performance drop $J(\pi, \mathcal{M}_{\text{real}}) - J(\pi, \mathcal{M}_{\text{sim}})$ is bounded by the total variation between transition distributions:

$$|J_{\text{real}} - J_{\text{sim}}| \leq \frac{2\gamma R_{\max}}{(1-\gamma)^2} \max_s D_{\text{TV}}(T_{\text{sim}}(\cdot|s,a) \| T_{\text{real}}(\cdot|s,a))$$

#### 2.2.2 Domain Randomization (DR)

- **Core Idea:** Instead of making the simulator match reality, randomize simulation parameters so the policy becomes robust to *any* plausible environment, including the real one.

$$\pi^* = \arg\max_{\pi} \ \mathbb{E}_{\xi \sim p(\xi)} \left[ J(\pi, \mathcal{M}_\xi) \right]$$

  where $\xi$ are randomized physics/visual parameters: friction $\mu \sim U[0.5, 1.5]$, mass $m \sim U[0.8m_0, 1.2m_0]$, joint damping, actuator delay, camera pose, lighting, texture.
- **Visual DR:** Randomize colors, textures, lighting, camera FOV, backgrounds. The policy learns features invariant to visual appearance — only geometry and motion matter.
- **Physics DR:** Randomize friction, restitution, mass, center-of-mass offset, motor strength, observation noise, action delay.
- **Landmark result — OpenAI Rubik's Cube (2019):** A dexterous robot hand (Shadow Hand) solved a Rubik's cube in the real world. Trained entirely in simulation with massive DR: randomized cube size, friction of each face, finger pad friction, gravity direction, observation latency, actuator noise. The policy saw ~$10^{10}$ randomized environments during training.
- **Automatic DR (ADR):** Start with narrow parameter ranges; widen them if the policy maintains performance above a threshold. Curriculum over randomization breadth.

$$\text{If } J(\pi) > \tau: \quad \xi_{\text{range}} \leftarrow \xi_{\text{range}} \cdot (1 + \alpha)$$

- **Limitations:**
  - Over-randomization produces overly conservative policies (sacrifice optimality for robustness).
  - DR assumes the real world is "within the randomization envelope" — fails for systematic biases (e.g., real actuators with nonlinear backlash not modeled at all).
  - Cannot close the gap for phenomena absent from the simulator (flexible cables, deformable cloth, fluids).

#### 2.2.3 System Identification & Domain Adaptation

- **System Identification (SysID):** Measure real-world physics parameters and set the simulator to match.

$$\xi^* = \arg\min_{\xi} \sum_t \| s_t^{\text{real}} - f_\xi(s_{t-1}^{\text{real}}, a_{t-1}) \|^2$$

  - Bayesian SysID: maintain posterior $p(\xi|\text{data})$, train policy over posterior samples.
  - Pro: most accurate if parameters are identifiable. Con: requires real-world data collection, parameters may be non-identifiable.
- **Domain Adaptation (DA):** Transform sim observations to look like real observations (or vice versa).
  - **Visual DA:** CycleGAN / RCAN to translate sim images → real-looking images. Train policy on translated images.
  - **Latent DA:** Train a shared encoder that maps both sim and real observations to a domain-invariant latent space. Adversarial training: $\min_{\text{encoder}} \max_{\text{discriminator}} \mathcal{L}_{\text{task}} - \lambda \mathcal{L}_{\text{domain}}$.
- **Comparison:**

| Method | Real Data Needed | Accuracy | Robustness | Computational Cost |
|---|---|---|---|---|
| Domain Randomization | None | Moderate | High | Low (parallel sim) |
| System Identification | Some (calibration) | High | Low (brittle to drift) | Medium |

| Method | Real Data Needed | Accuracy | Robustness | Computational Cost |
|---|---|---|---|---|
| Domain Adaptation | Some (unpaired images) | High | Moderate | High (GAN training) |
| DR + Fine-tuning | Small amount | Highest | High | Medium |

- **Best practice (2025):** DR for pretraining robustness + small amount of real-world fine-tuning (10-100 episodes) + SysID for known measurable parameters.

#### 2.2.4 Differentiable Physics Simulation

- **Core Idea:** Make the physics simulator differentiable end-to-end so you can compute $\frac{\partial \mathcal{L}}{\partial \theta}$ through the entire trajectory, where $\theta$ are policy parameters, control parameters, or physical design parameters.
- **Lagrangian Mechanics in Autodiff:**

$$\mathcal{L}(q, \dot{q}) = T(\dot{q}) - V(q) = \frac{1}{2}\dot{q}^T M(q)\dot{q} - V(q)$$

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} - \frac{\partial \mathcal{L}}{\partial q} = \tau + J^T\lambda$$

  where $\tau$ = applied torques, $J^T\lambda$ = contact forces. In matrix form: $M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau + J^T\lambda$.
  - If every operation ($M$, $C$, $g$, $J$, contact solve) is implemented in a differentiable framework (JAX, PyTorch, Warp), then $\frac{\partial s_{t+1}}{\partial s_t}$, $\frac{\partial s_{t+1}}{\partial a_t}$ are available via autodiff.
- **Gradient-Based Trajectory Optimization:** Instead of model-free RL (sample $10^6$ trajectories), directly differentiate through the physics:

$$\frac{\partial \mathcal{L}_{\text{task}}}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial \ell_t}{\partial s_t} \prod_{k=t}^{T-1} \frac{\partial s_{k+1}}{\partial s_k} \cdot \frac{\partial s_0}{\partial \theta}$$

  - Converges in ~$10^2$ iterations vs $10^6$ episodes for PPO. **1000-10000x more sample-efficient.**
  - But: requires differentiable simulator, susceptible to exploding/vanishing gradients over long horizons.
- **Key Frameworks (2024-2026):**

| Framework | Backend | Differentiable | Rigid Body | Soft Body | Speed |
|---|---|---|---|---|---|
| Brax (Google) | JAX | Yes (autodiff) | Yes | Limited | Very fast (GPU-parallel) |
| MuJoCo MJX | JAX | Yes (autodiff) | Yes | Yes (tendon) | Fast |
| DiffTaichi (MIT) | Taichi | Yes (source-to-source AD) | Yes | Yes (MPM) | Fast |
| NVIDIA Warp | CUDA/Python | Yes (kernel-level AD) | Yes | Yes | Very fast |
| Genesis (2025) | PyTorch/Taichi | Yes | Yes | Yes (MPM, FEM, SPH) | State-of-the-art |
| NVIDIA Isaac Sim | PhysX/Omniverse | Partial | Yes | Limited | Production-grade |

- **Genesis (2025):** Unified physics engine supporting rigid body, soft body (MPM), articulated body, fluid (SPH), cloth (FEM) — all differentiable, all on GPU, all in one framework. Designed as the "physics backbone" for generative physical AI. ~10-80x faster than Isaac Sim for parallel environments.
- **Short-Horizon vs Long-Horizon Differentiation:**
  - Short horizon ($T < 100$): direct backprop through physics works well.
  - Long horizon ($T > 1000$): gradients explode/vanish (chaotic dynamics). Solutions: shooting methods (break into sub-trajectories), adjoint method (constant memory), or use short-horizon differentiable planning + RL for long-horizon.

### 2.2.5 The Contact Problem

- Contact is the fundamental challenge for differentiable physics. Rigid body contact involves **discontinuous** dynamics — a ball either touches the floor or it doesn't. There is no smooth gradient through the contact event.
- **Complementarity Formulation:**

$$0 \leq \lambda \perp \phi(q) \geq 0$$

  where $\phi(q)$ = signed distance (gap), $\lambda$ = contact force. Either $\phi > 0$ (no contact, $\lambda = 0$) or $\lambda > 0$ (contact, $\phi = 0$). The gradient $\frac{\partial \lambda}{\partial q}$ is undefined at the transition.
- **Solutions for Differentiability:**
  - **Soft/Compliant Contact:** Replace hard contact with a spring-damper: $\lambda = k \cdot \max(0, -\phi)^n + d \cdot \dot{\phi}$. Smooth but physically inaccurate for stiff contacts.
  - **Randomized Smoothing:** Convolve the contact function with Gaussian noise: $\tilde{\lambda} = \mathbb{E}_\epsilon[\lambda(q+\sigma\epsilon)]$. Provides well-defined gradients but introduces bias.
  - **Interior Point / Barrier Methods:** Replace complementarity with a log-barrier: $\lambda\phi = \mu \to 0$. Smooth path to the solution; gradients exist everywhere.
  - **Implicit Differentiation:** Solve contact as $F(q, \lambda) = 0$, differentiate implicitly: $\frac{\partial \lambda}{\partial q} = -(\frac{\partial F}{\partial \lambda})^{-1} \frac{\partial F}{\partial q}$. Exact gradients at the solution without differentiating the solver iterations.
- **Friction (Coulomb):** $\|f_t\| \leq \mu\lambda_n$ (friction cone). Non-smooth and set-valued. Practical solutions: smoothed friction cone, convex relaxation.
- **Think Deeper:** "You're training a dexterous manipulation policy that must grasp objects of varying mass and friction. Explain why model-free RL requires $\sim 10^7$ environment steps while differentiable simulation requires $\sim 10^4$, but can fail when contacts change topology (e.g., object slips). What's your hybrid strategy?" (Answer: Differentiable physics gives exact $\frac{\partial \text{reward}}{\partial \text{action}}$ within a contact mode but gradients are discontinuous across mode transitions (contact/no-contact). Strategy: use differentiable physics for short-horizon control within a contact mode, use RL for mode-switching decisions. Or: use randomized smoothing to get biased but smooth gradients, then fine-tune with RL.)

### 2.2.6 Physics-Informed Neural Networks & Neural Operators

- **PINNs (Raissi et al., 2019):** Embed PDEs directly into the neural network loss:

$$\mathcal{L}_{\text{PINN}} = \underbrace{\mathcal{L}_{\text{data}}}_{\text{match observations}} + \lambda \underbrace{\mathcal{L}_{\text{PDE}}}_{\text{satisfy physics}}$$

  - For Navier-Stokes: $\mathcal{L}_{\text{PDE}} = \|\frac{\partial u}{\partial t} + (u \cdot \nabla)u + \nabla p - \nu\nabla^2 u\|^2 + \|\nabla \cdot u\|^2$.
  - Derivatives computed via autodiff of the network with respect to its inputs $(x, t)$.
  - **Pro:** Can solve inverse problems (infer viscosity $\nu$ from observations). **Con:** Training is notoriously difficult — spectral bias (low frequencies converge first), failure on multi-scale problems.
- **Fourier Neural Operators (FNO, Li et al., 2021):** Learn the solution *operator* $\mathcal{G} : f \mapsto u$ that maps PDE parameters/initial conditions to solutions.

$$u_{l+1}(x) = \sigma\left(W_l u_l(x) + \mathcal{F}^{-1}\left(R_l \cdot \mathcal{F}(u_l)\right)(x)\right)$$

  - $\mathcal{F}$ = FFT, $R_l$ = learnable weight in Fourier space (low-pass filter). Captures global interactions in one layer.
  - **Resolution-invariant:** Trained at 64x64, evaluated at 256x256. Discretization-independent.
  - **1000x faster** than classical PDE solvers at inference. Used for weather prediction (FourCastNet), fluid dynamics, material science.
- **DeepONet (Lu et al., 2021):** Two sub-networks: branch net (encodes input function) + trunk net (encodes query location). Output = dot product. Universal approximator for nonlinear operators.
- **Connection to World Models:** FNO/DeepONet can serve as the "physics backbone" in a world model. Instead of learning physics from pixels (Genie), inject known PDE structure. The trade-off: Genie-style models are more general (learn from video), PINNs/FNO are more accurate for *known* physics but require PDE formulation.

### 2.2.7 Differentiable Rendering, 3DGS & the Full Pipeline

- The full sim-to-real pipeline: **differentiable physics → differentiable rendering → image loss**.

  - Optimize physical parameters (shape, pose, material) to match real-world images by backpropagating through the entire rendering pipeline.
- **NeRF (Mildenhall et al., 2020):** Volume rendering is naturally differentiable: $C(\mathbf{r}) = \int_0^\infty T(t)\sigma(t)c(t)dt$, where $T(t) = \exp(-\int_0^t \sigma(s)ds)$. Gradients flow from pixel loss through volume rendering to the radiance field. But NeRF is slow (1-5 fps) — unsuitable for real-time or physics-in-the-loop training.
- **3D Gaussian Splatting (3DGS, Kerbl et al., 2023):** An explicit 3D representation using Gaussian primitives, each defined by position $\mu \in \mathbb{R}^3$, covariance $\Sigma \in \mathbb{R}^{3\times3}$, opacity $\alpha \in [0, 1]$, and Spherical Harmonics (SH) coefficients for view-dependent color.
  - **Differentiable Rasterization:** Project 3D Gaussians to 2D screen space. The projected covariance: $\Sigma' = JW\Sigma W^T J^T$ ($W$ = view transform, $J$ = Jacobian of projection). Pixel color via alpha-compositing:

$$C = \sum_{i=1}^{N} c_i \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j)$$

  - Fully differentiable — gradients flow from pixel loss to all Gaussian parameters.
  - **Covariance Parameterization:** $\Sigma$ must be PSD. Decompose as $\Sigma = RSS^T R^T$ where $R$ = rotation (quaternion $q$), $S$ = diagonal scaling. Optimize $(q, s)$ instead — PSD guaranteed by construction.
  - **Adaptive Density Control:** During training, Gaussians are split (too large), cloned (too small but high gradient), or pruned ($\alpha \approx 0$).
  - **Speed:** 100-200 fps rendering (vs NeRF 1-5 fps). Enables real-time interactive environments.
  - **Standardization:** Khronos adopted 3DGS as a glTF extension (Aug 2025) — becoming a standard interchange format.
  - **Limitations:** No inherent surface (blobs, not meshes). Solutions: SuGaR (surface extraction), 2DGS (flat Gaussians for surface alignment).
- **Physics + 3DGS — The Full Pipeline:** Treat Gaussians as material points with mass $m$, velocity $v$, elasticity. Simulate forward (differentiable physics via Brax/Warp/Genesis), render (differentiable rasterizer), compare to real video, backprop through everything. This is the frontier: jointly learning appearance and dynamics from video.
- **Think Deeper:** "Design a system that, given multi-view video of a ball bouncing, jointly infers the ball's restitution coefficient, gravity vector, and initial velocity using only pixel-level loss. What are the minimum components needed?" (Answer: (1) 3DGS for differentiable rendering, (2) differentiable rigid-body simulator with restitution/gravity as learnable parameters, (3) a trajectory optimizer that minimizes $\sum_t \sum_v \|I^{v,t}_{\text{rendered}} - I^{v,t}_{\text{real}}\|^2$. Gradients flow: pixel loss → rendering → Gaussian positions → physics state → physical parameters. Minimum 3 views for 3D, ~50 frames for temporal dynamics.)
- **Think Deeper:** "How do you learn a physics-aware 3DGS from multi-view video of a deforming object (cloth falling)? What prevents the Gaussians from cheating?" (Answer: Need (1) photometric loss, (2) physics loss penalizing non-physical accelerations, (3) As-Rigid-As-Possible (ARAP) regularization so neighboring Gaussians deform coherently. Without ARAP, each Gaussian teleports independently to minimize photometric loss.)

### 2.3 Physical Consistency Metrics (Evaluation)

- FID is useless for physics. We need **Physics-FID**.
- **Technique:** Using pre-trained **Video Action Recognizers** and **Depth Estimators** to check for:
  - **Temporal Monotonicity:** Does entropy increase or decrease unnaturally?
  - **Object Permanence:** Do embeddings of a "masked" object remain stable across occlusions?
- **NewtonGen (Yuan et al., 2025):** Physics-consistent video generation via **Neural Newtonian Dynamics (NND)** — trainable modules that model Newtonian motions and inject latent dynamical constraints into the generation process.
  - **Physical Invariance Score (PIS):** Measures whether a generated video preserves its expected physical invariant $C$ over time:

$$\text{PIS} = \left(1 + \frac{C_\sigma}{|C_\mu| + \epsilon}\right)^{-1}$$

    Score in $[0, 1]$; 1 = perfect invariance. Uses **SAM2** to segment objects per-frame, extract centroids, and estimate velocities from frame-to-frame differences.

  - **Evaluation across 12 Newtonian motion types:** uniform velocity, uniform acceleration, deceleration, parabolic, 3D motion, slope sliding, circular, rotation around axis, parabolic with rotation, damped oscillation, size change, deformation.
  - **Results:** NewtonGen achieves PIS 0.98 on uniform motion ($v_x$ invariance) vs. 0.53–0.97 for baselines (CogVideoX, Wan2.1). Consistent gains across all 12 motion categories.
- **Advanced Challenge:** Defining a **Lyapunov-based loss function** to penalize non-physical energy gains in a generated trajectory.

---

## Module 3: Engineering & Systems

**Core Focus:** Stability is the only feature that matters at scale.

### 3.1 Optimization Dynamics

- **Gradient Clipping:** The geometric interpretation (Trust Region).
- **AdamW:** Why Weight Decay $\neq$ L2 Regularization in adaptive settings.

### 3.2 Kernel Optimization (Triton / CUDA)

- **FlashAttention:** Tiling and Recomputation.
- **Task:** Write a Softmax kernel in Triton handling block boundaries.

### 3.3 Distributed Training: DDP vs. FSDP vs. DeepSpeed

- **The Three Paradigms & Their Trade-offs:**

| | DDP (DistributedDataParallel) | FSDP (Fully Sharded Data Parallel) | DeepSpeed ZeRO |
|---|---|---|---|
| **What's sharded** | Nothing (replicate full model) | Parameters + Gradients + Optimizer states | Configurable: ZeRO-1/2/3 |
| **Memory per GPU** | Full model + optimizer | $\sim\frac{1}{N}$ of total state | $\sim\frac{1}{N}$ of total state |
| **Communication** | AllReduce gradients only | AllGather params (fwd+bwd) + ReduceScatter grads | Same as FSDP (ZeRO-3) |
| **Comm volume** | $2\|M\|$ per step | $3\|M\|$ per step (ZeRO-3) | $3\|M\|$ per step (ZeRO-3) |
| **Best for** | Model fits in 1 GPU | Model doesn't fit; PyTorch-native | Model doesn't fit; maximum flexibility |
| **Ecosystem** | PyTorch native | PyTorch native (torch.distributed.fsdp) | Microsoft; HuggingFace integrated |
| **Mixed Precision** | Manual AMP | Built-in | Built-in with fp16/bf16/fp8 options |

- **DDP Deep Dive:**
  - Each GPU holds a full model replica. Forward pass is independent.
  - Backward: **AllReduce** gradients across all GPUs. Uses **bucket gradient AllReduce** — overlaps communication of early-layer gradients with computation of later-layer gradients.
  - **When to use:** Model fits in one GPU memory (including optimizer states). <~2B parameters on 80GB A100/H100.
- **FSDP Deep Dive:**
  - Shards model parameters across GPUs. Before each layer's forward/backward, **AllGather** to reconstruct full parameters. After backward, **ReduceScatter** gradients.
  - **Sharding strategies:** `FULL_SHARD` (maximum memory saving), `SHARD_GRAD_OP` (like ZeRO-2, keep params), `NO_SHARD` (= DDP).

  - **Wrapping policy:** Controls granularity — wrap per-TransformerBlock for best overlap. Too fine-grained = excessive communication; too coarse = memory spikes.
- **DeepSpeed ZeRO Stages:**
  - **ZeRO-1:** Shard optimizer states only. Memory: $\frac{1}{N}$ optimizer + full params + full grads.
  - **ZeRO-2:** + shard gradients. Memory: $\frac{1}{N}$ (optimizer + grads) + full params.
  - **ZeRO-3:** + shard parameters. Memory: $\frac{1}{N}$ everything. Equivalent to FSDP FULL_SHARD.
  - **ZeRO-Offload / ZeRO-Infinity:** Offload to CPU/NVMe. Allows training models larger than total GPU memory but 2-5x slower.
- **3D/5D Parallelism Stack (2025):** DDP/FSDP + Tensor Parallel (TP) + Pipeline Parallel (PP) + Expert Parallel (EP) + Context Parallel (CP).
- **Think Deeper:** "You have 64 H100 GPUs and need to train a 70B model. Design the parallelism strategy. What's the communication volume per step for DDP vs. FSDP, and at what model size does FSDP become necessary?" (Answer: 70B params = ~140GB in BF16 + ~560GB optimizer states in AdamW. DDP impossible (140GB > 80GB). Use FSDP with TP=8 within nodes, FSDP across 8 nodes. Comm: DDP AllReduce = $2|M|$; FSDP ZeRO-3 = $3|M|$. FSDP necessary when $2 \times |M|_{\text{bytes}} + 12 \times |M|_{\text{params}} >$ GPU_mem, roughly >13B for AdamW on 80GB.)

### 3.4 Softmax & Attention Complexity Optimization

- Self-attention is $O(N^2 d)$ FLOPs, $O(N^2)$ memory. For a 1-min video at 24fps, 512x512, patch-4: $N = 1440 \times 128 \times 128/16 \approx 1.5M$ tokens. Naive attention is impossible.
- **FlashAttention (Dao et al.):** The standard solution.
  - **Key Insight:** Attention is **memory-bound**, not compute-bound. The $N \times N$ attention matrix transfer between HBM and SRAM dominates wall-clock time.
  - **Tiling:** Compute attention in SRAM-sized blocks. Never write the full $N \times N$ matrix to HBM.
  - **Recomputation:** In backward pass, recompute attention weights instead of storing them. Trades $O(N^2)$ memory for $O(N^2)$ extra FLOPs (but FLOPs are cheap, memory bandwidth is expensive).
  - **FlashAttention-2:** 2x speedup via better work partitioning across warps.
  - **FlashAttention-3:** Exploits H100 features — async TMA, warp specialization, FP8 accumulation. Up to 740 TFLOPS on H100.
- **Alternatives for Extreme Sequence Lengths:**
  - **Ring Attention:** Shard sequence across GPUs. Each GPU computes attention for its local chunk, passing KV blocks in a ring. Communication overlapped with compute.
  - **Sliding Window Attention (Mistral):** Each token attends to only $w$ neighbors. $O(Nw)$ complexity. Stack multiple layers for effective global receptive field.
- **Think Deeper:** "FlashAttention doesn't change $O(N^2)$ FLOPs but achieves 2-4x wall-clock speedup. Explain the roofline model analysis — when is attention compute-bound vs. memory-bound, and how does tiling change this?" (Answer: Arithmetic intensity of standard attention = $O(1)$ (one multiply-add per element loaded). H100 has ~3TB/s bandwidth, ~1000 TFLOPS. Crossover at AI = 333. Standard attention AI « 333, so memory-bound. FlashAttention increases AI by keeping data in SRAM (20MB, ~30TB/s effective), making it compute-bound within tiles.)

### 3.5 Gradient Checkpointing & Training Memory Anatomy

- **What Consumes GPU Memory During Training:**

| Component | Memory (BF16, $P$ params) | Notes |
|---|---|---|
| Model Parameters | $2P$ bytes | BF16 = 2 bytes/param |
| Gradients | $2P$ bytes | Same dtype as params |
| Optimizer States (AdamW) | $8P$ bytes | 2 FP32 moments + FP32 param copy |
| Activations | $O(B \cdot N \cdot L \cdot d)$ | Batch x SeqLen x Layers x Hidden |
| **Total** | **12P + activations** | Activations often dominate for large B, N |

- Example: 7B model. Params: 14GB. Grads: 14GB. Optimizer: 56GB. **Total fixed: 84GB** (already exceeds one 80GB GPU without activations!).
- **Gradient Checkpointing (Activation Recomputation):**
  - **Idea:** Don't store all intermediate activations. Instead, store only at "checkpoint" boundaries (e.g., every $k$ layers). During backward pass, **recompute** activations from the nearest checkpoint.
  - **Trade-off:** Memory: $O(\sqrt{L})$ instead of $O(L)$ (for $L$ layers). Compute: ~33% extra FLOPs (one extra forward pass per segment).
  - **Selective Checkpointing (2025):** Only checkpoint the memory-hungry operations (attention, large FFN activations). Keep cheap activations (norms, residuals). Best cost-benefit ratio.
- **Other Memory Optimization:**
  - **Mixed Precision (BF16/FP16):** Halves activation memory. Master weights in FP32 only in optimizer.
  - **Activation Offloading:** Move activations to CPU between forward and backward. Slower but saves GPU memory.
  - **Micro-batching / Gradient Accumulation:** Reduce per-step activation memory by using smaller micro-batches, accumulating gradients over multiple steps.
- **Think Deeper:** "You're training a 13B model on 8xH100 (80GB each). The model + optimizer takes 156GB. Design the memory strategy." (Answer: FSDP across 8 GPUs -> 156GB/8 = ~20GB fixed per GPU. Remaining 60GB for activations. With gradient checkpointing every 2 layers + BF16 activations + batch size 4 + seq len 4096, activations ~30-40GB. Fits comfortably. Without FSDP, impossible on single GPU.)

### 3.6 Fault Tolerance & Bad Node Detection in Distributed Training

- **The Problem at Scale:** With 2048+ GPUs, hardware failures are **inevitable**. A single bad GPU can silently corrupt gradients, causing divergence without any error.
  - **Mean time between failures (MTBF)** for a 2048-GPU cluster: ~2-4 hours. Training runs last weeks.
- **Types of Failures:**
  - **Hard failures:** GPU crash, OOM, NaN in loss. Easy to detect — process dies or loss spikes.
  - **Soft failures (Silent Data Corruption):** GPU computes wrong results but doesn't crash. Bit flips in HBM, degraded memory cells, overheating throttling. **These are the dangerous ones.**
- **Detection Strategies:**
  - **Gradient Norm Monitoring:** Track $\|g_i\|$ per GPU per step. A bad GPU will show anomalous norms:
    - $\|g_i\| \gg \text{median}(\|g_j\|)$ -> likely corrupted activations.
    - $\|g_i\| = 0$ -> GPU stuck or dead.
  - **Heartbeat + Checksumming:** Periodic AllReduce of a known tensor. Compare checksums to detect bit errors.
  - **Redundant Computation:** Run critical operations on 2 GPUs, compare outputs. Expensive but used for high-stakes training.
  - **Loss Spike Analysis:** If loss spikes after a checkpoint but recovers after re-sharding to different GPUs -> suspect the removed GPU.
- **Recovery Strategies:**
  - **Elastic Training (torch.distributed.elastic):** Automatically restarts failed workers. Resumes from latest checkpoint.
  - **In-memory Checkpointing:** Save model state to CPU/NVMe every $K$ steps (e.g., every 5 min). Reduces recovery time from hours to minutes.
  - **Hot Spare GPUs:** Pre-allocated replacement GPUs. Automatic failover without stopping the job.
  - **DeepSeek-V3 Approach:** DualPipe + automated pipeline reconfiguration. When a node fails, redistribute pipeline stages across remaining nodes within seconds.
- **Think Deeper:** "During a 1000-GPU training run, you notice periodic loss spikes every ~200 steps but no GPU crashes. How do you diagnose and fix?" (Answer: Likely soft data corruption on one GPU. Steps: (1) Log per-GPU gradient norms — identify the outlier GPU. (2) Run memtest on suspected GPU. (3) If confirmed, exclude GPU and remap ranks. (4) For prevention: enable ECC memory checks, monitor GPU temperatures, use gradient clipping as a safety net.)

### 3.7 Communication-Computation Overlap & FP8 Training

***User-Defined Kernels** for FP8/Int8 training.* **Concept:** As models grow, we move from BF16 to **FP8 (E4M3/E5M2)**. * **Challenge:** Managing dynamic range (Scaling Factors) in a transformer to prevent "Activation Outliers" from crashing the loss.

### 3.8 Deployment: Distillation, Consistency Models & DMD

- Turn a 20-50 step diffusion model into a 1-4 step model for real-time inference.
- **Three Paradigms:**

**Consistency Models (Song et al., 2023)**

- **Consistency Property:** A function $f_\theta$ that maps any point on the ODE trajectory to the same output: $f_\theta(x_t, t) = f_\theta(x_{t'}, t')$ for all $t, t'$ on the same trajectory.
- **Consistency Distillation Loss:**

$$\mathcal{L}_{\text{CD}} = \|f_\theta(x_{t_{n+1}}, t_{n+1}) - f_{\theta^-}(x_{t_n}, t_n)\|^2$$

  - $f_{\theta^-}$: EMA target network (stabilizes training, like in DQN).
  - $x_{t_n}$: one ODE step from $x_{t_{n+1}}$ using the teacher model.
- **Consistency Training (CT):** No teacher needed — learns consistency from scratch using noise pairs. Simpler but lower quality.
- **Result:** 1-2 step generation. FID 3.55 on ImageNet 64x64 (1 step).

**DMD / DMD2 (Yin et al., CVPR 2024 / NeurIPS 2024 Oral)**

- **Distribution Matching Distillation:** Instead of matching individual trajectories (forward KL), match the **distribution** of outputs (reverse KL-style).
- **DMD Gradient:**

$$\nabla_\theta \approx \mathbb{E}_z \left[ (s_{\text{fake}}(G_\theta(z)) - s_{\text{real}}(G_\theta(z))) \cdot \nabla_\theta G_\theta(z) \right]$$

  - $s_{\text{real}}$: score of data distribution (pretrained teacher, frozen).
  - $s_{\text{fake}}$: score of student's generated distribution (trained online).
  - Pushes student's output distribution toward teacher's output distribution.
- **DMD2 Improvements:** Removes regression loss (no need for expensive noise-image pair dataset). Two-time-scale update rule (update fake critic faster than generator). **FID 1.28** on ImageNet, one step.
- **Adversarial Diffusion Distillation (ADD, Stability AI):** Combines diffusion loss + adversarial loss. Powers SDXL Turbo (1-4 steps).

**Comparison**

| Method | Steps | FID (ImageNet) | Needs Teacher Sampling? | Loss Type |
|---|---|---|---|---|
| Consistency Distillation | 1-2 | 3.55 | Yes (one ODE step) | Forward KL (trajectory matching) |
| DMD2 | 1 | 1.28 | No | Reverse KL (distribution matching) |
| ADD (SDXL Turbo) | 1-4 | ~competitive | Yes | Adversarial + Diffusion |
| Progressive Distillation | 1-4 | 2.40 | Yes (halving steps) | MSE |

- **Think Deeper:** "Why does DMD2 achieve lower FID than consistency models despite both being one-step? Relate to forward vs. reverse KL." (Answer: Consistency models use forward KL-style loss — they average over

trajectories, leading to mode-covering (slightly blurry). DMD2 uses reverse KL-style — it penalizes generating unrealistic samples, leading to sharper outputs. The two-score gradient estimation lets DMD2 achieve this without mode collapse.)

### 3.9 Continuous Batching & Inference Systems

- Static batching wastes GPU cycles. In a batch of 8 requests, the shortest finishes in 50 tokens, the longest in 500. With static batching, all 8 wait for the longest -> 90% of GPU cycles wasted for the short requests.
- **Continuous Batching (Iteration-Level Scheduling):**
  - **Idea:** After each decode step, evict finished sequences and insert new ones. The batch is always full.
  - **Result:** Up to 23x throughput improvement over static batching (Anyscale benchmark).
  - **Implementation:** vLLM, TensorRT-LLM, TGI all support this.
- **PagedAttention (Kwon et al., 2023 — vLLM):**
  - **Problem:** KV cache memory is allocated per-sequence and grows dynamically. Pre-allocating max-length wastes ~60-80% of memory.
  - **Solution:** Manage KV cache like **virtual memory pages**. Each sequence's KV cache is stored in non-contiguous physical blocks. A page table maps logical positions to physical blocks.
  - **Benefits:** Near-zero memory waste, enables memory sharing for beam search / parallel sampling, enables larger batch sizes.
- **Chunked Prefill:** Split long prompts (32K+) into chunks. Interleave prefill chunks with decode steps so prefill doesn't block latency-sensitive decode operations.
- **For Diffusion Models:**
  - Continuous batching is harder — each request takes a fixed number of denoising steps (not variable like LLM decoding).
  - **Batched denoising:** Group requests at the same noise level. Use different noise schedules for different quality tiers.
  - Real-time diffusion inference: 1-step distilled models + batched inference on H100 -> ~30-60fps for 512x512.
- **Think Deeper:** "Design an inference system that serves a diffusion model at 30fps for 100 concurrent users. What's the GPU requirement?" (Answer: 1-step distilled model (DMD2). Per-image: ~1 forward pass of UNet/DiT, ~10ms on H100. 100 users at 30fps = 3000 images/sec. Batch size 32 -> 94 batches/sec needed. At 10ms/batch: need ~1 H100. Add safety margin + networking: 2-4 H100s with continuous batching.)

### 3.10 LoRA & Parameter-Efficient Fine-Tuning (PEFT)

- **The Problem:** Full fine-tuning a 70B model requires storing a full copy of parameters + optimizer states = ~560GB. For $N$ downstream tasks, that's $N \times 560$GB.
- **LoRA (Hu et al., 2022):** Freeze pretrained weights $W_0 \in \mathbb{R}^{d\times d}$, add a low-rank update:

$$W = W_0 + \frac{\alpha}{r} BA, \quad B \in \mathbb{R}^{d\times r},\ A \in \mathbb{R}^{r\times d}$$

  - $r \ll d$ (typical: $r$ = 8-64 for $d = 4096$). Trainable params: $2dr$ vs $d^2$. For $r = 16$, $d = 4096$: **0.8%** of original parameters.
  - $\alpha/r$ scaling: $\alpha$ controls the magnitude of the update. Standard practice: set $\alpha = 2r$ so the effective learning rate is independent of rank choice.
  - **Initialization:** $A$ from Gaussian, $B = 0$. At initialization, $\Delta W = 0$ — fine-tuning starts from the pretrained model exactly.
  - **Why low-rank works:** Aghajanyan et al. (2021) showed that fine-tuning operates in a low intrinsic dimensionality — even random $d_{\text{intrinsic}} \approx 200$ subspaces capture 90% of fine-tuning performance on a 350M model.
  - **Applied to:** $W_Q, W_K, W_V, W_O$ in attention (most common), and $W_{\text{up}}, W_{\text{down}}$ in FFN (for larger adaptations).
- **QLoRA (Dettmers et al., 2023):** Quantize the base model to 4-bit (NF4 data type) + LoRA adapters in BF16. Enables fine-tuning a 65B model on a single 48GB GPU.
  - **NF4 (NormalFloat4):** Quantization levels placed at the quantiles of a normal distribution — optimal for normally-distributed weights.

  - **Double Quantization:** Quantize the quantization constants themselves. Saves additional ~0.37 bits per parameter.
  - Memory for QLoRA on 70B: $70B \times 0.5$ bytes (4-bit) + LoRA params $\approx$ 35GB + ~3GB LoRA = ~**38GB**. Fits on 1x A100.
- **DoRA (Liu et al., 2024):** Weight-Decomposed LoRA. Decomposes $W$ into magnitude $m$ and direction $V$:

$$W = m \cdot \frac{V}{\|V\|_c}, \quad V = V_0 + BA$$

  - LoRA applies to the direction component; magnitude is a separate learnable vector. Empirically matches full fine-tuning quality with LoRA-level parameter count.
- **Comparison:**

| Method | Trainable Params | Memory | Quality vs Full FT |
|---|---|---|---|
| Full Fine-Tuning | 100% | $16P$ (BF16 + Adam) | Baseline |
| LoRA ($r = 16$) | ~0.8% | $0.5P$ (frozen) + small | 95-99% |
| QLoRA ($r = 16$) | ~0.8% | $0.25P$ (4-bit) + small | 93-97% |
| DoRA ($r = 16$) | ~1% | $0.5P$ + small | 98-100% |
| Prefix Tuning | ~0.1% | $0.5P$ + small | 85-95% |
| Adapters | ~2-5% | $0.5P$ + adapters | 95-98% |

- **LoRA for Diffusion (DreamBooth-LoRA):** Apply LoRA to the cross-attention layers ($W_K, W_V$) of a DiT/UNet to learn new concepts (faces, styles, objects) from ~5 images. Training: ~5 minutes on 1 GPU. Resulting adapter: ~10MB vs ~6GB for full model.
- **Think Deeper:** "LoRA rank $r = 16$ works well for language tasks but fails for complex visual style transfer. Why, and what's your fix?" (Answer: Visual features require higher-rank updates because the style manifold has higher intrinsic dimensionality than task-specific language adaptations. Fix: (1) increase $r$ to 64-128 for visual tasks, (2) apply LoRA to FFN layers (not just attention), (3) use LoRA+ (different learning rates for $A$ and $B$), or (4) use DoRA for magnitude-direction decomposition which better captures stylistic changes.)

### 3.11 GQA/MQA & Speculative Decoding

- **The KV Cache Bottleneck:** During autoregressive decoding, the KV cache stores all past key-value pairs. For a model with $L$ layers, $n_h$ heads, head dim $d_h$, and sequence length $S$:

$$\text{KV cache size} = 2 \times L \times n_h \times d_h \times S \times \text{bytes}$$

  - Llama-2-70B ($L = 80$, $n_h = 64$, $d_h = 128$): KV cache for 4K context in BF16 $= 2 \times 80 \times 64 \times 128 \times 4096 \times 2$ bytes $\approx$ **10.7 GB per sequence**. For batch size 32: **342 GB** — more than the model weights.
- **Multi-Head Attention (MHA):** Standard. Each head has its own $W_K^i, W_V^i$. KV cache $= 2 \times L \times n_h \times d_h \times S$.
- **Multi-Query Attention (MQA, Shazeer 2019):** All heads share a *single* KV pair. KV cache $= 2 \times L \times 1 \times d_h \times S$. $n_h$**x reduction.** But quality degrades — one KV pair is a severe bottleneck for the information each head can attend to.
- **Grouped-Query Attention (GQA, Ainslie et al., 2023):** Interpolation between MHA and MQA. Group $n_h$ heads into $G$ groups, each group shares one KV pair:

$$\text{KV cache} = 2 \times L \times G \times d_h \times S$$

  - $G = n_h$: MHA. $G = 1$: MQA. Typical: $G = 8$ with $n_h = 32$ (Llama 3, Gemini).
  - **Quality:** GQA-8 matches MHA quality while using 4x less KV cache. The sweet spot for production LLMs.
- **Multi-Head Latent Attention (MLA, DeepSeek-V2/V3):** Compress KV into a low-rank latent:

$$c_t = W_{\text{DKV}} k_t^{(i)}, \quad k_t^{(i)} = W_{\text{UK}}^{(i)} c_t, \quad v_t^{(i)} = W_{\text{UV}}^{(i)} c_t$$

  - Store only $c_t \in \mathbb{R}^{d_c}$ (e.g., $d_c = 512$) instead of full KV ($n_h \times d_h = 4096$). **8x compression** beyond GQA.

Used in DeepSeek-V3 (Section 4.3).

- **Speculative Decoding (Leviathan et al., 2023; Chen et al., 2023):**
  - **Problem:** Autoregressive decoding is memory-bandwidth-bound. A 70B model generates ~30 tokens/sec on 1 GPU — most of the time is spent loading weights from HBM, not computing.
  - **Key Insight:** Verification is cheaper than generation. A large model can verify $K$ tokens in parallel (single forward pass) in approximately the same time as generating 1 token.
  - **Algorithm:**
    1. **Draft:** A small, fast model (e.g., 1B) generates $K$ candidate tokens autoregressively.
    2. **Verify:** The large target model scores all $K$ tokens in a single forward pass.
    3. **Accept/Reject:** Accept tokens from left to right while the target model agrees (within a probability threshold). Reject the first disagreement and resample from the target model.
  - **Acceptance probability:** For token $i$, accept with probability $\min(1, \frac{p_{\text{target}}(x_i)}{p_{\text{draft}}(x_i)})$. If rejected, sample from the "residual" distribution $\max(0, p_{\text{target}} - p_{\text{draft}})$ (normalized). This guarantees the output distribution is **exactly** $p_{\text{target}}$ — zero quality loss.
  - **Expected speedup:** If the draft model's acceptance rate is $\alpha$ and it generates $K$ drafts: expected tokens per step $= \frac{1-\alpha^{K+1}}{1-\alpha}$. With $\alpha = 0.8$, $K = 5$: ~3.4 tokens per step → **~3.4x speedup**.
- **Speculative Decoding Variants:**
  - **Medusa (Cai et al., 2024):** No separate draft model. Add $K$ parallel prediction heads to the target model itself. Each head predicts 1, 2, ..., $K$ tokens ahead. Verify via tree attention over candidate sequences. ~2-3x speedup with minimal additional parameters.
  - **EAGLE (Li et al., 2024):** Auto-regression in feature space. A lightweight head predicts the *hidden states* of future tokens (not logits), enabling higher acceptance rates. ~3-5x speedup.
  - **Lookahead Decoding:** Use Jacobi iteration — guess all future tokens simultaneously, iterate until convergence. No draft model needed.
- **Think Deeper:** "You deploy a 70B model with speculative decoding using a 1B draft. The acceptance rate is only 50% (expected: 80%). Diagnose the problem." (Answer: Low acceptance means the draft model's distribution diverges significantly from the target. Causes: (1) draft model was not distilled from the target — use a distilled draft model, (2) system prompt or temperature settings differ between draft and target, (3) the task requires specialized knowledge the 1B model lacks (e.g., code, math). Fix: distill a task-specific draft model, match sampling parameters exactly, or use Medusa/EAGLE which share the target model's representations.)

---

## Module 4: Sparse Architectures — Mixture of Experts (MoE)

**Core Focus:** Understanding why >60% of frontier models in 2025 are MoE.

### 4.1 Why MoE Dominates Frontier Models

- Dense scaling is dead. Every top-tier model (DeepSeek-V3, GPT-5, Gemini 3) uses Mixture of Experts.
  - **Core Insight:** Decouple total parameters (capacity) from active parameters (compute per token).
  - **DeepSeek-V3:** 671B total, only 37B active per token. 256 routed experts + 1 shared expert.
  - **GPT-5 (Aug 2025):** Hybrid architecture with a real-time router that dynamically selects between Standard (full reasoning), Mini (fast), and Nano (on-device) sub-models. While OpenAI has not confirmed MoE explicitly, the tiered routing design is functionally analogous — a meta-level MoE where each "expert" is an entire model optimized for a different cost/quality trade-off.
  - **Scaling Law:** MoE achieves equivalent loss to dense models at ~$\frac{1}{3}$ the training FLOPs.
- **Practical Application:** MoE naturally supports domain-specialized "physics experts" (fluids, rigid body, cloth) activated by content routing — ideal for simulation engines.

### 4.2 Expert Routing Mechanisms

- **Gating Function:**

$$G(x) = \text{TopK}(\text{softmax}(W_g \cdot x + \text{noise}), k)$$

  - Select top-$k$ experts; output: $y = \sum_{i \in \text{TopK}} g_i \cdot \text{Expert}_i(x)$.

- **Load Balancing Loss (Switch Transformer):**

$$\mathcal{L}_{\text{aux}} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i$$

  - $f_i$ = fraction of tokens routed to expert $i$ (non-differentiable, involves argmax).
  - $P_i$ = mean routing probability for expert $i$ (differentiable, gradients flow here).
  - Minimized when $f_i = P_i = \frac{1}{N}$ for all experts.
- **Expert Capacity Factor:**

$$C_{\text{expert}} = \left\lceil \frac{k \cdot T}{N} \cdot C_{\text{factor}} \right\rceil$$

  - Tokens exceeding capacity are **dropped** (skip expert, pass through residual only). Target: <1% drop rate.
- **Router Z-Loss (ST-MoE):**

$$\mathcal{L}_z = \frac{1}{B} \sum_{i=1}^{B} \left( \log \sum_{j=1}^{N} e^{l_j^{(i)}} \right)^2$$

  - Penalizes logit magnitude to prevent softmax overflow. Coefficient ~0.001.

### 4.3 DeepSeek-V3: The State-of-the-Art MoE

- **Multi-Head Latent Attention (MLA):** The real breakthrough behind DeepSeek's inference efficiency.
  - **Problem:** Standard MHA requires caching $2 \cdot n_h \cdot d_h$ per token. For $n_h = 128, d_h = 128$: 32,768 elements/token.
  - **Solution:** Low-rank KV compression:

$$c_t^{KV} = W_{DKV} \cdot h_t \quad (d \to d_c), \quad k_t = W_{UK} \cdot c_t^{KV}, \quad v_t = W_{UV} \cdot c_t^{KV}$$

  - Cache only $c_t^{KV}$ ($d_c = 512$) instead of full K,V. **57x KV cache reduction.**
  - **Decoupled RoPE:** Position-aware component computed separately ($d_h^R = 64$), not through the latent bottleneck. Total cache: $d_c + d_h^R = 576$ elements/token.
- **Auxiliary-Loss-Free Load Balancing:**
  - Route using bias: selected = TopK($s_i + b_i$). Gate using raw affinity: $g_i = \text{softmax}(s_i)$ for $i \in$ selected.
  - Dynamic bias update: $b_i \leftarrow b_i \pm \gamma$ based on actual vs. target load. No quality-degrading auxiliary loss needed.
- **Fine-Grained Expert Segmentation:**
  - 256 smaller experts (vs. Mixtral's 8 large ones). Each expert handles a narrower subdomain.
  - 1 shared expert always active (learns universal patterns: syntax, common knowledge).
  - Output: $y = \text{SharedExpert}(x) + \sum_{i \in \text{TopK}} g_i \cdot \text{Expert}_i(x)$.
- **FP8 Training at Scale:** E4M3 for all tensors (not E5M2 for backward). Fine-grained quantization: activations per $1 \times 128$ tile, weights per $128 \times 128$ block. Cost: $5.6M for 14.8T tokens on 2048 H800 GPUs.

### 4.4 MoE Training Challenges

- **Expert Collapse / Dead Experts:** Rich-get-richer dynamics. Popular experts get more gradient signal -> improve further -> attract more tokens. Solutions: noisy gating, expert-choice routing, DeepSeek's bias-term approach.
- **Communication Overhead:** Expert Parallelism (EP) requires two All-to-All per MoE layer (dispatch tokens -> return results). DeepSeek-V3 uses **DualPipe** to overlap forward/backward computation-communication phases.
- **Think Deeper:** "Why does the auxiliary load balancing loss $\mathcal{L}_{\text{aux}} = \alpha N \sum f_i P_i$ only send gradients through $P_i$ and not $f_i$? What does this actually optimize?" (Answer: $f_i$ involves argmax, which has zero gradient almost everywhere. Gradients flow through $P_i$ (softmax output), pushing routing probabilities toward uniformity, which indirectly balances the non-differentiable token counts $f_i$.)

### 4.5 Expert Parallelism (EP) & 5D Parallelism

- **The Stack (2025):** Data Parallel + Tensor Parallel + Pipeline Parallel + **Expert Parallel + Sequence/Context Parallel**.
- **EP Communication:** Each MoE layer requires two All-to-All: dispatch ($T \cdot d/P$ per device) + combine. For DeepSeek-V3: EP32 across 2048 GPUs.
- **DualPipe:** Novel pipeline schedule overlapping forward/backward across pipeline stages, reducing the ~1:1 computation-to-communication ratio.
- **Think Deeper:** "Given $E$ experts across $P$ devices, $T$ tokens per device, hidden dim $d$, expert FLOPs $F_e$, and interconnect bandwidth $B_w$: derive when communication becomes the bottleneck. At what $E/P$ ratio does EP stop scaling?"

---

## Module 5: Native Multimodal Intelligence

**Core Focus:** From "bolted-together models" to natively multimodal architectures.

### 5.1 Early Fusion vs. Late Fusion: The Fundamental Dichotomy

- **Early Fusion (Native):** All modalities tokenized into a shared space, processed jointly from the first layer.
  - Models: Gemini 3, Chameleon, GPT-5/5.2, Transfusion.
  - GPT-5 (Aug 2025) is natively multimodal: accepts text, image, audio, and video as input; generates text, image, and audio as output — all within a single end-to-end model. Reports ~45% fewer factual errors than GPT-4o on internal benchmarks.
  - Self-attention over **all** tokens: $\text{Attn}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, where $x = [x_{\text{text}}; x_{\text{image}}; x_{\text{audio}}]$.
  - Attention matrix dimensionality: $(N_{\text{text}} + N_{\text{image}} + N_{\text{audio}})^2$ – full cross-modal interaction at every layer.
- **Late Fusion (Adapter):** Frozen modality-specific encoders + lightweight adapter -> frozen LLM.
  - Models: LLaVA, BLIP-2, Flamingo.
  - Cross-attention: $\text{CrossAttn}(Q_A, K_B, V_B)$ – information flows asymmetrically through a bottleneck.
  - BLIP-2: 32 learnable queries attend to 257 ViT patches -> **10.7x information compression**.

| Criterion | Early Fusion | Late Fusion |
|---|---|---|
| Cross-modal depth | Every layer | Adapter bottleneck only |
| Training compute | $O(10^{24} - 10^{25})$ FLOPs | $O(10^{20} - 10^{22})$ FLOPs |
| Reuse existing LLMs | No (train from scratch) | Yes |
| Multi-modal generation | Natural | Requires separate decoders |
| Information flow | Unrestricted | Bounded by $K \cdot d$ (bottleneck) |

- **Think Deeper (Data Processing Inequality):** "Prove that there exists a function class that early fusion can represent but late fusion with $K$ queries cannot." (Answer: Late fusion with $K$ queries has mutual information $I(f; I) \leq K \cdot d \cdot \log(\text{precision})$. For BLIP-2: $K \cdot d = 32 \times 768 = 24{,}576$ vs. $M \cdot d_v = 257 \times 1024 = 263{,}168$ for full ViT features. ~10x representational capacity gap.)

### 5.2 Unified Discrete Token Space: Chameleon (Meta)

- **Architecture:** Decoder-only Transformer; ALL modalities mapped to a single vocabulary.
  - Text: 57,344 BPE tokens. Image: 8,192 codebook tokens (VQ-VAE, $512 \times 512 \to 1{,}024$ discrete tokens). Total: **65,536 unified vocabulary.**
  - Trained end-to-end on ~10 trillion tokens of interleaved mixed-modal data.
- **Critical Stability Techniques:**
  - **QK-Norm:** $q_{\text{norm}} = \text{LayerNorm}(q)$, $k_{\text{norm}} = \text{LayerNorm}(k)$. Prevents modality-specific norm explosion from dominating softmax.
  - **Z-Loss:** $\mathcal{L}_z = 10^{-5} \cdot \log^2(Z)$, where $Z = \sum_i e^{\text{logit}_i}$. Prevents logit drift across mixed vocabulary.
- **Think Deeper:** "Without QK-Norm, attention logits are $q^T k = \|q\|\|k\| \cos\theta$. If image tokens develop $\|q_{\text{img}}\| \gg \|q_{\text{text}}\|$, what happens to gradient flow and how does QK-Norm fix it?"

### 5.3 Hybrid Autoregressive + Diffusion: Transfusion (ICLR 2025)

- **Key Insight:** Instead of lossy VQ for images, keep images as continuous latent patches + diffusion loss, while using standard cross-entropy for text.
- **Combined Loss:**

$$\mathcal{L}_{\text{Transfusion}} = \mathcal{L}_{\text{LM}} + 5 \cdot \mathcal{L}_{\text{DDPM}}$$

  - $\mathcal{L}_{\text{LM}} = -\sum_{i \in \text{text}} \log P(y_i | y_{<i})$ (per-token cross-entropy).
  - $\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{x_0,t,\epsilon} \|\epsilon - \epsilon_\theta(x_t, t, c)\|^2$ (per-image MSE).
  - $\lambda = 5$: upweights diffusion because images are denser in information than individual text tokens.
- **Hybrid Attention Mask:** Text tokens -> **causal** (standard AR). Image patches within same image -> **bidirectional** (diffusion needs joint denoising). Image patches to other content -> **causal**.
- **Information Bottleneck Analysis:** Discrete codebook (8,192) = $\log_2(8192)$ = 13 bits/patch. Continuous float32 (dim-256) = 8,192 bits/patch. **630x gap** – why Transfusion outscales Chameleon for image generation.

### 5.4 Vision Encoders: CLIP vs. SigLIP

- **CLIP (InfoNCE):**

$$\mathcal{L}_{\text{CLIP}} = -\frac{1}{N} \sum_i \log \frac{e^{s_{ii}/\tau}}{\sum_j e^{s_{ij}/\tau}}, \quad \tau = e^s \text{ (learnable)}$$

  - Requires full $N \times N$ similarity matrix. Two all-gather ops across GPUs.
- **SigLIP (Sigmoid):**

$$\mathcal{L}_{\text{SigLIP}} = -\frac{1}{N^2} \sum_i \sum_j \log \sigma(z_{ij} \cdot (-t \cdot x_i \cdot y_j + b))$$

  - $z_{ij} = +1$ if positive pair, $-1$ otherwise. Independent binary classification per pair.
  - One all-gather op (no global normalization needed). Memory-efficient for large batches.
  - **Crossover:** SigLIP > CLIP for batch size < 32K; CLIP > SigLIP for batch size > 32K.
- **SigLIP 2 (Feb 2025):** Hybrid contrastive + generative training (captioning decoder during training, removed at inference).

### 5.5 Audio Tokenization & End-to-End Voice

- **GPT-5 Native Audio Architecture (Aug 2025):** Single end-to-end transformer for audio-in -> audio-out, processing all modalities within a unified model rather than the GPT-4o-era Whisper->LLM->TTS cascade. GPT-5 achieves ~**200ms** voice response latency (matching human conversational turn-taking at ~200-300ms). GPT-5.2 (late 2025) further improved long-context audio coherence and multi-turn voice dialogue.
- **Residual Vector Quantization (RVQ – SoundStream/EnCodec):**

$$q_1 = \text{Quantize}(z), \quad q_k = \text{Quantize}(z - \sum_{i<k} q_i), \quad z_q = \sum_{k=1}^{K} q_k$$

  - Each layer: codebook size 1,024 -> 10 bits. At 50Hz frame rate: $K \times 500$ bps.
  - Variable bitrate via structured dropout: randomly drop layers $k > K'$ during training. Single model: 3-18 kbps.
- **Practical Application:** Audio-visual sync uses cross-attention between audio RVQ tokens and video flow matching velocity field — exactly as described in Section 5.7.

### 5.6 The Any-to-Any Generation Frontier

- **Three Paradigms:**
  - **Fully Discrete** (Chameleon, Emu3): All modalities as discrete tokens, single cross-entropy loss. Simple but lossy.
  - **Hybrid AR+Diffusion** (Transfusion): Text discrete + images continuous. Best quality but complex inference.
  - **Pure Next-Token** (Emu3): With good enough visual tokenizer, pure AR matches SDXL. Published in Nature 2025.

- **Think Deeper:** "Derive the computational cost of training Transfusion vs. Chameleon on the same dataset. When does each dominate?" (Answer: Training cost nearly identical – diffusion uses single forward pass with random $t$, not $T$ denoising steps. Image generation inference: Transfusion is ~$T$x more expensive. Transfusion dominates when image quality > speed; Chameleon dominates for real-time applications.)

### 5.7 Audio-Visual Alignment & Temporal Synchronization

- Ensuring generated audio and video are temporally aligned (lip-sync, sound effects, musical rhythm) is critical for believable multimodal generation.
- **Contrastive Audio-Visual Pretraining (CAVP):** Learn a shared embedding space where temporally aligned audio-video pairs are close and misaligned pairs are far:

$$\mathcal{L}_{\text{CAVP}} = -\log \frac{e^{\text{sim}(a_t, v_t)/\tau}}{\sum_{t'} e^{\text{sim}(a_t, v_{t'})/\tau}}$$

  where $a_t$ = audio embedding at time $t$, $v_t$ = video frame embedding at time $t$. This is temporal InfoNCE — contrastive loss over time rather than batch.
- **Cross-Attention Control:** Audio spectrograms (or RVQ tokens from Section 5.5) serve as conditioning tokens for the video generation model. In a Flow Matching framework, the velocity field becomes audio-conditioned:

$$v_t(x_t, c_{\text{audio}}) = \text{DiT}(x_t, t, \text{CrossAttn}(Q_{\text{video}}, K_{\text{audio}}, V_{\text{audio}}))$$

- **Lip-Sync Metrics:** SyncNet confidence score (audio-visual offset detection), LSE-D (lip sync error - distance), LSE-C (lip sync error - confidence).
- **Think Deeper:** "Your text-to-video model generates a person speaking, but the lip movements are 200ms ahead of the audio. Where in the pipeline would you diagnose and fix this?" (Answer: (1) Check audio-video tokenization alignment — if audio RVQ operates at 50Hz and video at 24fps, there's a frame-rate mismatch that must be handled by the cross-attention positional encoding. (2) Check if training data had systematic A/V offset (common in web-scraped videos). (3) Add a SyncNet-based reward during post-training to penalize A/V desync.)

---

## Module 6: Reasoning, Post-Training & Test-Time Compute

**Core Focus:** The 2025 paradigm shift – from "scale training" to "scale inference."

### 6.1 Preference Optimization (RLHF / DPO)

- **DPO:** Implicit reward maximization via $\beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}$. Eliminates the need for an explicit reward model by reparameterizing the RLHF objective as a classification loss on preference pairs.
- **Think Deeper:** "Why does DPO sometimes lead to mode collapse faster than PPO?" (Answer: DPO pushes down probability of rejected responses without explicitly maintaining diversity. PPO's KL penalty against the reference policy provides a softer constraint. DPO's implicit reward can become unbounded for out-of-distribution responses, causing the policy to concentrate on a narrow set of high-preference outputs.)

### 6.2 The Test-Time Compute Paradigm

- Instead of making models bigger, let them "think longer." Performance scales with inference compute, not just parameters. This is the defining paradigm of 2025-2026.
  - **Key milestones:** OpenAI **o3** (Apr 2025) scored 88.9% on AIME 2025, 87.7% GPQA Diamond, and a landmark **25.2% on Frontier Math** (vs. <2% for all prior models including o1). **o4-mini** (Apr 2025) achieved 92.7% AIME 2025 at 10x lower cost than o3, with 2719 Codeforces ELO. **Gemini 3 Deep Think** won gold medals at Physics & Chemistry Olympiads. **DeepSeek-R1** matched o1-level reasoning with open weights.
  - **Compute-Optimal Inference (REBASE, ICLR 2025):**

$$(N_{\text{opt}}, T_{\text{opt}}; S) = \arg\min E(N, T; S) \quad \text{s.t. FLOPs}(N, T, S) = C$$

  - **Empirical scaling:** $\log_{10}(C) = 1.19 \cdot \log_{10}(N) + 2.03$.
  - Llemma-7B with tree search = Llemma-34B greedy, at **2x fewer FLOPs**.
- **When to Think Longer vs. Use a Bigger Model:**

| Problem Difficulty | Optimal Strategy |
|---|---|
| Easy | Test-time compute strongly preferred |
| Intermediate | Test-time compute preferred; iterative refinement effective |
| Hard | Bigger model preferred; test-time compute has diminishing returns |

### 6.3 Provable Scaling Laws (Chen & Pan, NeurIPS 2025)

- **Knockout Algorithm:** Generate $N$ candidates, aggregate via knockout tournament with $K$ pairwise comparisons per round.

$$P(\text{failure}) \leq (1 - p_{\text{gen}})^N + \lceil \log_2 N \rceil \cdot e^{-2K(p_{\text{comp}} - 0.5)^2}$$

  - $p_{\text{gen}}$: probability of generating a correct solution.
  - $p_{\text{comp}}$: probability of correctly comparing a correct/incorrect pair.
- **For $\delta$-failure:**

$$N \geq \frac{1}{p_{\text{gen}}} \log \frac{2}{\delta}, \quad K \geq \frac{1}{2(p_{\text{comp}} - 0.5)^2} \log \frac{2\lceil \log_2 N \rceil}{\delta}$$

- **Advanced Challenge:** Both algorithms require only a black-box LLM – no external verifier. The scaling is exponential in $K$, power-law in $N$. Derive the Pareto frontier.

### 6.4 DeepSeek-R1: Pure RL for Reasoning

- **R1-Zero (Minimal Approach):** Base model + GRPO with verifiable rewards only. **No SFT, no human-labeled reasoning traces.** Reasoning emerges purely from RL.
- **R1 (Full 4-Stage Pipeline):**

| Stage | Method | Data |
|---|---|---|
| 1. Cold Start | SFT | ~Thousands of curated reasoning examples |
| 2. Reasoning RL | GRPO + RLVR | Large-scale RL with accuracy + format rewards |
| 3. Rejection Sampling + SFT | Sampling | 800K samples (600K reasoning + 200K general) |
| 4. Alignment RL | RLHF | Helpfulness + harmlessness |

- **GRPO Objective:**

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q,\{o_i\}} \left[ \frac{1}{G} \sum_{i=1}^{G} \min \left( r_i(\theta) \hat{A}_i,\ \text{clip}(r_i(\theta), 1 \pm \epsilon) \hat{A}_i \right) - \beta \, D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right]$$

  - $r_i(\theta) = \frac{\pi_\theta(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)}$, $\hat{A}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$.
  - **No critic model** – advantage estimated from group statistics. ~50% memory savings vs PPO.

### 6.5 RLVR: Reinforcement Learning from Verifiable Rewards

- **Core Concept:** Replace learned reward models with deterministic verifiers.
  - Math: symbolic comparison against ground truth ($\boxed{\text{answer}}$ format).
  - Code: compilation + test case execution.

- **Reward Function:**

$$R(o_i, q) = R_{\text{accuracy}} + R_{\text{format}}$$

  - $R_{\text{accuracy}} = \mathbb{1}[y = y^*]$ (binary). $R_{\text{format}} = \mathbb{1}[\text{contains } \langle\text{think}\rangle ... \langle/\text{think}\rangle]$.
- **Why RLVR > Learned Reward Models:** DeepSeek explicitly abandoned neural reward models due to **reward hacking** at scale: models find spurious reward patterns. Verifiable rewards are immune (deterministic verification).

### 6.6 The "Aha Moment" & Emergent Behaviors

- **Self-Reflection Emergence:** During R1-Zero training, the word "wait" shows a **sudden frequency spike** at a specific training step. The model spontaneously learns to pause, re-evaluate, and correct – without any supervised signal.
- **Mechanistic Explanation (Entropy Dynamics):**
  - Token-level entropy **decreases** (execution becomes predictable).
  - Semantic entropy of planning tokens **increases** (model explores diverse strategies).
  - AIME 2024 accuracy trajectory: 15.6% -> 71.0% across RL training.
- **Emergent Behaviors:** Backtracking ("This leads to contradiction, let me try…"), case analysis, estimation-then-verification, dynamic strategy switching.

### 6.7 Search & Verification at Inference Time

- **Best-of-N:** Generate $N$ completions, select highest-scoring. $y^* = \arg\max_{y_i} R(y_i|x)$.
- **Majority Voting (Self-Consistency):** Sample $N$ chains, select by plurality: $y^* = \arg\max_a |\{i : \text{answer}(y_i) = a\}|$.
- **REBASE Tree Search:** Allocate expansion budget proportional to PRM score:

$$W_j = \text{Round}\left(B_i \cdot \frac{e^{R(n_j)/T_b}}{\sum_k e^{R(n_k)/T_b}}\right)$$

  - Pareto-optimal across all compute budgets. 7x compute reduction for equivalent performance.
- **MCTS for Reasoning:** DeepSeek tried MCTS and **failed** – search space too large, value model too expensive to train. Works only in structured domains (code with compilation feedback, formal proofs).

### 6.8 Reasoning Distillation

- **The Result That Changed Everything:**

| Model | Params | AIME 2024 | MATH-500 | GPQA Diamond | LiveCodeBench |
|---|---|---|---|---|---|
| OpenAI o1-mini | ~100B | 63.6% | 90.0% | 60.0% | 53.8% |
| R1-Distill-Qwen-32B | **32B** | **72.6%** | **94.3%** | **62.1%** | **57.2%** |

- A 32B distilled model beat the ~100B o1-mini across every benchmark. By Apr 2025, o3-mini closed this gap (reportedly ~80%+ AIME 2024), but the distillation insight stands: it is far more sample-efficient to distill from a strong teacher than to train small models from scratch with RL.
- **Why distillation > direct RL on small models:**
  1. Small models struggle to **discover** effective reasoning strategies through RL (sparse reward, large exploration space).
  2. The 671B teacher already found effective patterns; distillation transfers them directly.
  3. SFT on 800K curated traces » billions of RL exploration steps in sample efficiency.
  4. RL on small models produces degenerate outputs (repetition, language mixing – R1-Zero failure modes).
- **Latent variable interpretation:** $P(y|x) = \sum_{z \sim p(z|x)} P(y|x, z)$. The teacher provides high-quality samples from $p(z|x)$ (reasoning chains), which the student learns from directly.

### 6.9 Test-Time Training (TTT) Layers (Sun et al., 2024)

- **Beyond Attention:** TTT layers replace the attention mechanism with a **self-supervised learning step at inference time.**
  - **Key Insight:** The hidden state of a sequence model is itself a machine learning model (linear layer or MLP). The "update rule" is a gradient descent step on a self-supervised loss.
  - **TTT-Linear:** Hidden state is a linear model $W$. At each token, update: $W \leftarrow W - \eta \nabla_W \mathcal{L}_{\text{SSL}}(x_t; W)$.
  - **TTT-MLP:** Hidden state is a two-layer MLP. More expressive but more expensive per-token.
- **Why It Matters:**
  - Self-attention: $O(N^2)$ complexity, but the "hidden state" (KV cache) grows linearly and compresses nothing.
  - RNNs (Mamba, RWKV): $O(N)$ complexity, but fixed-size hidden state limits long-context performance.
  - TTT: $O(N)$ complexity with an **expressive hidden state that learns** from context. Perplexity keeps improving with context length (unlike Mamba which plateaus at ~16K).
- **TTT-E2E (End-to-End, 2025):** Matches full-attention accuracy at 128K context while being **2.7x faster** on H100.
- **Connection to Test-Time Compute (Section 6.2):** TTT is "scaling inference compute" at the *representation* level (better hidden states), while reasoning scaling (6.2) is at the *output* level (more thinking tokens). They are complementary.
- **Think Deeper:** "Compare the information-theoretic capacity of: (a) a KV cache of length $N$ with dim $d$, (b) a TTT-Linear hidden state of size $d \times d$, (c) a Mamba recurrent state of size $d \times r$. Which can represent the most information from $N$ past tokens, and why?" (Answer: KV cache stores $O(Nd)$ bits — lossless but expensive. TTT-Linear stores $O(d^2)$ bits but the model *learned* to compress. Mamba stores $O(dr)$ bits with fixed compression. For $N > d$, KV cache wins in capacity but TTT wins in efficiency. TTT's advantage: it optimizes *what* to remember via gradient descent, while Mamba uses a fixed gating mechanism.)

### 6.10 Mamba & State Space Models (SSMs)

- **Motivation:** Attention is $O(N^2)$ compute, $O(N)$ KV cache that grows without bound. Can we build sequence models with $O(N)$ compute and $O(1)$ state?
- **Continuous-Time State Space Model:**

$$x'(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t)$$

  where $A \in \mathbb{R}^{n \times n}$ is the state matrix, $B \in \mathbb{R}^{n \times 1}$ is the input matrix, $C \in \mathbb{R}^{1 \times n}$ is the output matrix. The state $x(t) \in \mathbb{R}^n$ acts as a compressed memory.
- **Discretization (Zero-Order Hold):** For discrete sequences with step size $\Delta$:

$$\bar{A} = e^{\Delta A}, \quad \bar{B} = (\Delta A)^{-1}(e^{\Delta A} - I) \cdot \Delta B$$

$$x_k = \bar{A}x_{k-1} + \bar{B}u_k, \quad y_k = Cx_k$$

  - This is a linear recurrence — $O(N)$ sequential compute, $O(n)$ constant state. But it can also be expressed as a **convolution** (unroll the recurrence): $y = \bar{K} * u$ where $\bar{K} = (C\bar{B}, C\bar{A}\bar{B}, C\bar{A}^2\bar{B}, ...)$. Convolution mode enables GPU-parallel training via FFT.
- **S4 (Gu et al., 2022 — Structured State Spaces):**
  - **Problem:** Naive SSMs forget long-range dependencies because eigenvalues of $\bar{A}$ decay exponentially.
  - **HiPPO Initialization:** Initialize $A$ as the HiPPO matrix (History Polynomial Projection Operator), which optimally compresses continuous signals into a polynomial basis. The resulting state tracks a running approximation of the entire input history.
  - **Diagonal + Low-Rank:** Decompose $A = \Lambda + PQ^T$ where $\Lambda$ is diagonal. Allows efficient computation while maintaining HiPPO structure.
  - S4 achieved state-of-the-art on Long Range Arena (LRA, sequences up to 16K) — the first sub-quadratic model to match Transformer performance on long-range tasks.
- **Mamba (Gu & Dao, 2024 — Selective State Spaces, S6):**
  - **Key Innovation:** Make SSM parameters **input-dependent** (selective). Instead of fixed $A, B, C, \Delta$:

$$B_t = \text{Linear}_B(x_t), \quad C_t = \text{Linear}_C(x_t), \quad \Delta_t = \text{softplus}(\text{Linear}_\Delta(x_t))$$

  - $\Delta_t$ controls the "gate" — large $\Delta$ = focus on current input (reset memory), small $\Delta$ = retain memory. This is analogous to the forget gate in LSTMs, but within the SSM framework.
  - **Why selectivity matters:** Fixed SSMs treat all tokens equally — they cannot ignore irrelevant tokens or focus on important ones. Selective SSMs learn *what to remember and what to forget*, content-dependently.
  - **Hardware-Aware Algorithm:** The selective (input-dependent) SSM breaks the convolution trick (can't precompute kernel). Gu & Dao designed a **parallel scan** algorithm that computes the recurrence in $O(N \log N)$ using GPU-friendly operations, with careful SRAM/HBM management (similar philosophy to FlashAttention).
  - **Mamba Block:** Input → two branches: (1) Linear → Conv1D → SSM → output, (2) Linear → SiLU gate. Multiply branches. No attention, no MLP block — the SSM replaces both.
  - **Results:** Mamba-3B matches Transformer-3B on language modeling. **5x higher throughput** at inference (linear scaling with sequence length). Excels on DNA/audio/signal sequences.
- **Mamba-2 (Dao & Gu, 2024 — Structured State Space Duality, SSD):**
  - **Theoretical unification:** Showed that selective SSMs are mathematically equivalent to a structured form of attention:

$$y = M \cdot (Lx), \quad M_{ij} = C_i^T (\prod_{k=j+1}^{i} \bar{A}_k) \bar{B}_j$$

  - The matrix $M$ is a **semiseparable matrix** — a masked version of the attention matrix where the "attention pattern" is constrained to be expressible through the SSM recurrence. This means SSMs are attention with a specific structural prior.
  - **Practical:** Mamba-2 is 2-8x faster than Mamba-1 (leverages matrix multiply hardware via the SSD framework). Matches Transformer++ quality at 2.7B parameters.
- **Hybrid Architectures (The 2025 Consensus):**
  - **Jamba (AI21, 2024):** Alternates Mamba layers and Attention layers (ratio 7:1). Uses MoE in some layers. 256K effective context. The attention layers handle "precise recall" (needle-in-haystack), while Mamba layers handle "smooth integration" (summarization, generation).
  - **Zamba (Zyphra, 2024):** Shared attention layer interleaved with Mamba blocks. Only 1 attention layer (shared across all positions), rest are Mamba.
  - **Why hybrids win:** Pure Mamba struggles with tasks requiring **exact copying/retrieval** over long contexts (the fixed-size state lossy-compresses). Attention excels at retrieval but is expensive. Hybrid = Mamba for most layers (cheap, good at generation) + sparse attention layers for retrieval (expensive but precise).
- **Comparison:**

| Architecture | Compute | State Size | Retrieval | Generation | Training |
|---|---|---|---|---|---|
| Transformer | $O(N^2 d)$ | $O(Nd)$ KV cache | Excellent | Good | Parallel |
| Mamba (SSM) | $O(Nd)$ | $O(d \cdot n)$ fixed | Weak | Excellent | Parallel (scan) |
| RWKV | $O(Nd)$ | $O(d^2)$ fixed | Moderate | Good | Parallel (WKV) |
| TTT | $O(Nd)$ | $O(d^2)$ learned | Good | Good | Parallel (?) |
| Hybrid (Jamba) | $O(Nd + N^2 d/k)$ | Mixed | Good | Excellent | Parallel |

- **Open Questions:** Can SSMs fully replace attention for reasoning tasks? Current evidence suggests no — reasoning requires precise in-context retrieval that fixed-size states cannot guarantee. The field is converging on **hybrids** as the pragmatic solution, with the attention-to-SSM ratio as the key hyperparameter.
- **Think Deeper:** "Mamba's state $x_k \in \mathbb{R}^n$ compresses the entire history into $n$ dimensions. For a needle-in-a-haystack retrieval task (find a specific token in a 100K context), derive why Mamba's recall probability degrades with sequence length while Transformer's does not." (Answer: Mamba's state update $x_k = \bar{A}_k x_{k-1} + \bar{B}_k u_k$ mixes all past information through matrix multiplication. After $N$ steps, the contribution of token $j$ to state $x_N$ is $\prod_{k=j+1}^{N} \bar{A}_k \cdot \bar{B}_j u_j$. For $N \gg n$, the state rank is at most $n$ — it cannot represent $N$ independent memories. Retrieval degrades as $O(n/N)$. Transformer's KV cache stores every token losslessly — retrieval is $O(1)$ regardless of $N$.)

### 6.11 Open Problems

- **Overthinking:** 82.3% of problems solvable with minimal reasoning. Up to 22.4% token waste. Adaptive compute allocation (route by difficulty) is unsolved.
- **CoT Faithfulness (Anthropic, 2025):** Claude 3.7 Sonnet mentions hints only **25%** of the time. DeepSeek R1: **39%**. RL improves faithfulness initially but **plateaus at 20-28%**. If CoT is unfaithful, monitoring it cannot detect misaligned behaviors.
- **Beyond Math/Code:** RLVR requires verifiable rewards. Open-ended domains (law, philosophy, creativity) lack natural verifiers. No successful extension demonstrated.
- **Process vs. Outcome Supervision:** PRMs are **8% more accurate** and **6x more sample-efficient** than ORMs. But DeepSeek abandoned PRMs due to harder-to-detect reward hacking hidden within reasoning steps.

---

## Advanced Deep Dive Questions

*Questions requiring strong mathematical foundations.*

**Modules 1-3 (Foundations & Systems):**

1. **Topology & Continuity:** "In latent video space, how do you mathematically guarantee temporal continuity (no flickering) without using an expensive 3D Conv-VAE?"
2. **Symmetry & Invariance:** "How can we bake **SE(3) Equivariance** into a DiT so that the model understands that a rotating object is the same object from different angles?"
3. **Information Theory:** "What is the lower bound of 'Test-time Compute' required to resolve a physical ambiguity in a video (e.g., which way a spinning top will fall)?"
4. **High-Dimensional Geometry:** "Explain the Curse of Dimensionality in Vector Databases. Why does Euclidean distance lose meaning, and how do HNSW graphs solve navigation?"

**Module 4 (MoE):**

5. **MoE Scaling Theory:** "DeepSeek-V3 has 671B total / 37B active with 256 experts. Derive the 'effective parameter multiplier' – under the Chinchilla scaling law framework, is MoE provably better than dense scaling? At what expert count does the routing overhead negate the capacity gain?"
6. **MLA Information Loss:** "Prove that MLA KV compression is equivalent to standard MHA when $d_c \geq n_h \cdot d_h$. Then characterize the information loss as a function of $d_c/(n_h \cdot d_h)$ in terms of the rank of the key/value matrices."

**Module 5 (Multimodal):**

7. **Multimodal Stability:** "Chameleon's unified softmax over 65,536 tokens (57K text + 8K image) creates modality competition. Formalize this as a bi-modal gradient landscape problem. Why are standard techniques (warmup, gradient clipping) insufficient, and what does QK-Norm specifically fix?"
8. **Information Bottleneck:** "Derive the gradient of the CLIP contrastive loss with respect to the temperature parameter $\tau$. Why is temperature learnable, and what happens to training dynamics when $\tau \to 0$ vs. $\tau \to \infty$?"

**Module 6 (Reasoning):**

9. **Reasoning Emergence:** "The 'aha moment' in R1-Zero emerges without supervised signal. Model this as a multi-armed bandit: each reasoning strategy is an arm, reward is accuracy. Why does self-reflection (a strategy that costs tokens but improves accuracy) get discovered by GRPO but not by greedy search?"
10. **Test-Time Scaling Frontier:** "Given the knockout algorithm failure probability $P(\text{fail}) \leq (1 - p_{\text{gen}})^N + \lceil \log_2 N \rceil \cdot e^{-2K(p_{\text{comp}} - 0.5)^2}$, derive the Pareto frontier between $N$ (generation budget) and $K$ (comparison budget) for fixed total compute $C = 2N + K \log_2 N$."
11. **Distillation Theory:** "R1-Distill-Qwen-32B beats o1-mini on every benchmark, but 7B distilled models degrade sharply. Meanwhile, o3-mini and o4-mini recovered the gap through improved training. Is there a theoretical minimum model capacity below which distillation of reasoning fails? Relate this to information-theoretic compression bounds on reasoning traces of length $L$ with vocabulary $V$."
12. **SSM-Attention Duality:** "Mamba-2 shows that selective SSMs are equivalent to attention with a semiseparable mask matrix $M$. Derive the rank of $M$ as a function of state dimension $n$ and sequence length $N$. When $n \ll N$, what class of attention patterns can SSMs *not* express, and how does this explain hybrid architectures?"

---

## Suggested Study & Build Roadmap

**Phase 0: MoE & Multimodal Foundations**

- ☐ Build a simple MoE layer with 8 experts + top-2 routing in PyTorch. Implement load balancing loss and monitor expert utilization.
- ☐ Compare dense FFN vs. MoE-FFN at matched active FLOPs on multi-domain data.
- ☐ Build a minimal Chameleon-style unified tokenizer (text BPE + image VQ-VAE). Observe QK-Norm's effect.

**Phase 1: Data & Latent Pipeline**

- ☐ Build a VAE/MagViT encoder. Compare VQ vs FSQ codebook utilization. Analyze latent drift over long sequences.
- ☐ Set up a synthetic data pipeline (Blender/Unity/Genesis) for ground-truth physics training data.
- ☐ Use a VLM to caption synthetic data and verify label density.

**Phase 2: Scalable Training**

- ☐ Build a toy DiT with Flow Matching (AdaLN-Zero conditioning, RoPE 2D).
- ☐ Implement Ring Attention for long-context training.
- ☐ Train with FP8 precision and monitor loss spikes / gradient norms.
- ☐ Fine-tune a model with LoRA ($r = 16$) and QLoRA. Compare to full fine-tuning on a downstream task.
- ☐ Implement a GRPO training loop with verifiable rewards on GSM8K.
- ☐ Build a REBASE-style tree search using a PRM. Compare to Best-of-N and majority voting.
- ☐ Distill reasoning traces from a large model to a 7B student. Measure accuracy vs. direct RL.

**Phase 3: Sim-to-Real & Differentiable Physics**

- ☐ Train a simple policy in MuJoCo/Brax with domain randomization. Transfer to a modified environment and measure the sim-to-real gap.
- ☐ Implement a differentiable rigid-body simulator (e.g., using Brax or Warp). Optimize a trajectory via backpropagation through physics.
- ☐ Build a simple PINN for a 1D wave equation. Compare training difficulty to a standard MLP.

**Phase 4: Evaluation, Deployment & Architecture Experiments**

- ☐ Build an automated physics checker using VLM/depth estimation for video scoring.
- ☐ Implement speculative decoding with a draft model. Measure acceptance rate and wall-clock speedup.
- ☐ Build a minimal Mamba block (selective SSM + parallel scan). Compare to attention on a copy/retrieval task to observe the recall gap.
- ☐ Build a Transfusion-style dual-loss trainer (cross-entropy + diffusion) on a small shared transformer.
- ☐ Implement cross-modal attention masks (causal for text, bidirectional within images).

---

## Key References

**Module 1 (Generative Engine):**


- Flow Matching for Generative Modeling (Lipman et al.): arXiv:2210.02747
- Scalable Diffusion Models with Transformers (DiT): arXiv:2212.09748
- Stable Diffusion 3 / Rectified Flow: Stability AI
- MeanFlow – One-Step Generation (NeurIPS 2025 Oral): arXiv:2505.13447
- Pyramidal Flow Matching (ICLR 2025): arXiv:2410.05954
- Ring Attention (ICLR 2024): Paper


**Module 1 (Diffusion Additions):**


- Classifier-Free Guidance (Ho & Salimans, 2022): arXiv:2207.12598
- Min-SNR Weighting: arXiv:2303.09556
- Input Perturbation for Exposure Bias: arXiv:2301.11706
- DMD (CVPR 2024): arXiv:2311.18828
- DMD2 (NeurIPS 2024 Oral): arXiv:2405.14867

- Consistency Models (Song et al., 2023): arXiv:2303.01469

**Module 2 (Physical World & Sim-to-Real):**

- Genie: Generative Interactive Environments (ICML 2024): arXiv:2402.15391
- Genie 2 (DeepMind, Dec 2024): deepmind.google/blog/genie-2
- Genie 3 (DeepMind, 2025-2026): deepmind.google/blog/genie-3
- 3DGS in glTF (Khronos, Aug 2025): Volinga Blog
- V-JEPA 2 (Meta, Jun 2025): Meta AI
- NVIDIA Cosmos World Foundation Models: arXiv:2501.03575
- VBench-2.0: arXiv:2503.21755
- NVIDIA Newton Physics Engine: NVIDIA Blog
- Genesis Physics Engine: GitHub
- Brax (Google): GitHub
- DiffTaichi (Hu et al., ICLR 2020): arXiv:1910.00935
- NVIDIA Warp: GitHub
- Physics-Informed Neural Networks (Raissi et al., 2019): arXiv:1711.10561
- Fourier Neural Operator (Li et al., 2021): arXiv:2010.08895
- Domain Randomization for Sim-to-Real (Tobin et al., 2017): arXiv:1703.06907
- Solving Rubik's Cube with a Robot Hand (OpenAI, 2019): arXiv:1910.07113
- NewtonGen (Yuan et al., 2025): arXiv:2509.21309

**Module 3 (Engineering & Systems):**

- FlashAttention-3: arXiv:2407.08608
- PyTorch FlexAttention: PyTorch Blog
- Dynamic Context Parallelism (NVIDIA): NVIDIA Blog
- vLLM & PagedAttention: arXiv:2309.06180
- Continuous Batching (Anyscale): Anyscale Blog
- DeepSpeed ZeRO: arXiv:1910.02054
- PyTorch FSDP: PyTorch Blog
- Adversarial Diffusion Distillation (SDXL Turbo): arXiv:2311.17042
- LoRA (Hu et al., 2022): arXiv:2106.09685
- QLoRA (Dettmers et al., 2023): arXiv:2305.14314
- DoRA (Liu et al., 2024): arXiv:2402.09353
- GQA (Ainslie et al., 2023): arXiv:2305.13245
- Speculative Decoding (Leviathan et al., 2023): arXiv:2211.17192
- Medusa (Cai et al., 2024): arXiv:2401.10774
- EAGLE (Li et al., 2024): arXiv:2401.15077

**Module 4 (MoE):**

- DeepSeek-V3 Technical Report: arXiv:2412.19437
- Switch Transformer: arXiv:2101.03961
- ST-MoE: arXiv:2202.08906
- Auxiliary-Loss-Free Load Balancing: arXiv:2408.15664
- MoE LLMs Overview (Cameron Wolfe): Substack

**Module 5 (Multimodal):**

- Gemini 3: deepmind.google/models/gemini
- Gemini Technical Report (1.0): arXiv:2312.11805
- Chameleon: arXiv:2405.09818
- Transfusion (ICLR 2025): arXiv:2408.11039
- SigLIP: arXiv:2303.15343
- SigLIP 2: arXiv:2502.14786
- BLIP-2: arXiv:2301.12597
- Emu3 (Nature 2025): arXiv:2409.18869

**Module 6 (Reasoning & Post-Training):**

- DeepSeekMath / GRPO: arXiv:2402.03300

- "Let's Verify Step by Step" (OpenAI PRM): arXiv:2305.20050
- ThinkPRM: arXiv:2504.16828
- DeepSeek-R1: arXiv:2501.12948
- REBASE (ICLR 2025): OpenReview
- Provable Scaling Laws (NeurIPS 2025): arXiv:2411.19477
- "Reasoning Models Don't Always Say What They Think" (Anthropic, 2025): arXiv:2505.05410
- Self-Consistency (Wang et al., ICLR 2023): arXiv:2203.11171
- "Stop Overthinking" Survey (TMLR 2025): GitHub
- TTT Layers (Sun et al., 2024): arXiv:2407.04620
- TTT-E2E: test-time-training.github.io
- S4 (Gu et al., 2022): arXiv:2111.00396
- Mamba (Gu & Dao, 2024): arXiv:2312.00752
- Mamba-2 / SSD (Dao & Gu, 2024): arXiv:2405.21060
- Jamba (AI21, 2024): arXiv:2403.19887


---

*A comprehensive guide to the frontier AI research landscape, 2025-2026. Contributions and corrections welcome.*