Bridging Deep Learning and Clinical Development

Transformers, Diffusion Models, and AI Agents

Kui Shen, PhD

Oct 2026

Disclaimer

The opinions expressed in this presentation are the authors' own and do not represent those of their employers.

Outline: prediction, generation, and AI automation

- Part I: Predict clinical outcomes
  - From Cox regression to neural networks and tabular transformers.
- Part II: Generate structured data
  - From Bayesian inference to diffusion, SurvDiff, and OrdinalDiff.
- Part III: Build verifiable AI-assisted analyses
  - From governed clinical data to AI-agent workflows and evaluation.

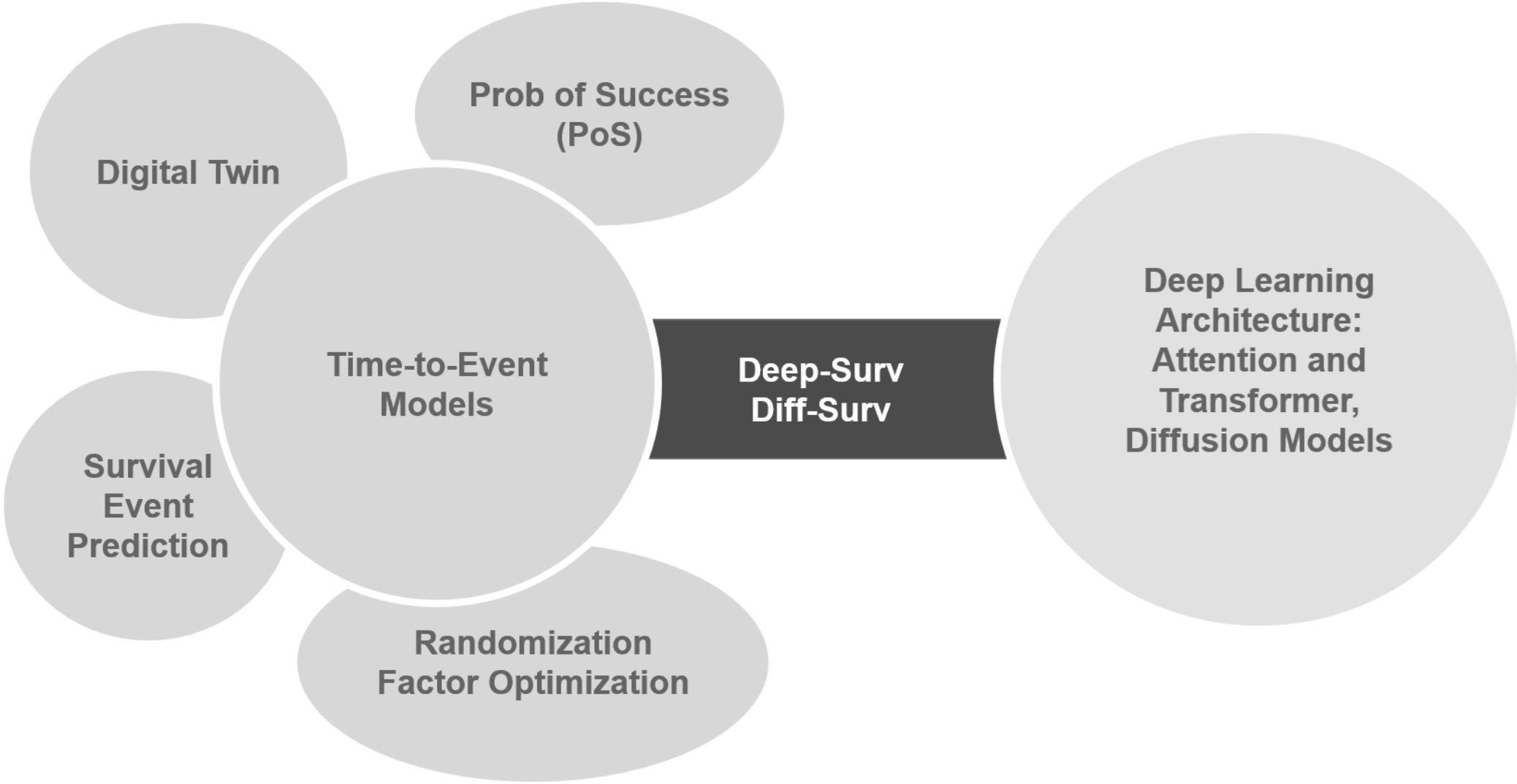
The evolving role of clinical statistics in the AI era: a paradigm shift

- Traditional responsibilities: study design, trial monitoring, health-authority submissions, and statistical deliverables, alongside strategic scientific input.
- AI-era vision: extend statisticians' strategic role across the clinical development lifecycle, from early-phase planning through regulatory approval and beyond.
- A barrier to this vision: systematically integrating internal trial data with external sources such as ClinicalTrials.gov, PubMed, and public or licensed databases requires better data access, harmonization, provenance and validated modeling.

The opportunity in the AI era

- The transformation: extend clinical statistics' capacity as a strategic partner.
- A proposed approach: integrate appropriate external evidence with clinical data and develop validated models for decision support across the development lifecycle.
    - Deep learning algorithms for survival modeling
    - Generative (diffusion and flow-matching) models for synthetic patient data
    - Clinical knowledge databases and relational databases, and AI agent for automation

Time-to-event models: a conceptual bridge



Deep learning approaches for survival modeling

Part I: Transformer-based models

Survival fundamentals: a one-slide recap

Let T be event time, C censoring time, $\tilde{T} = \min(T, C)$ and $\delta = \mathbf{1}(T \leq C)$. For continuous event times:

$$S(t) = \Pr(T > t), \qquad h(t) = \lim_{\Delta t \downarrow 0} \frac{\Pr(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t}.$$

Hazard and survival describe the same event-time distribution:

$$h(t) = \frac{f(t)}{S(t)} = -\frac{d}{dt} \log S(t), \qquad S(t) = \exp\left[-\int_0^t h(u)\, du\right].$$

The Cox proportional hazards model

Factor the hazard into a shared baseline and a covariate risk score:

$$h(t \mid x) = \underbrace{h_0(t)}_{\text{nonparametric baseline}} \exp\Big(\underbrace{\beta^\top x}_{\text{log-risk score}} \Big).$$

For distinct event times, estimate β without specifying $h_0(t)$:

$$L(\beta) = \prod_{i:\delta_i=1} \frac{\exp(\beta^\top x_i)}{\sum_{j \in R(\tilde{t}_i)} \exp(\beta^\top x_j)}, \qquad R(u) = \{j : \tilde{t}_j \geq u\}.$$

The one limitation we want to relax

$$\text{risk score} = \beta^\top x = \beta_1 x_1 + \cdots + \beta_p x_p$$

This plain predictor is linear and additive in the supplied features.

What it does not capture without suitable features:

- Nonlinear effects, such as U-shaped biomarker risk.
- Interactions, such as a treatment association that varies by genotype.
- Structure in high-dimensional raw inputs, such as images or mutations.

Classical Cox can include splines and interaction terms. Deep learning instead learns a flexible function from data; this does not itself relax proportional hazards or establish causal effects.

A neural network model for survival data

- Faraggi & Simon (1995): an influential early neural-network extension of Cox proportional hazards.
- This classical work combined a neural risk function with the Cox framework, an important conceptual precursor to DeepSurv. It relaxed predictor linearity, not proportional hazards.

> David Faraggi & Richard Simon. A neural network model for survival data. Statistics in Medicine 14(1):73–82 (1995). DOI: 10.1002/sim.4780140108.

The idea: relax Cox's linearity with a neural net

The limitation of a plain Cox predictor: log-risk is linear in the supplied features:

$$h(t \mid x) = h_0(t)\, \exp(\beta^\top x).$$

Replace the linear predictor $\beta^\top x$ with a feed-forward network $g(x; \theta)$:

$$h(t \mid x) = h_0(t)\, \exp\big(g(x; \theta)\big).$$

The network can learn nonlinearities and interactions, subject to data, regularization and validation. A time-invariant score still assumes PH. With a linear output and no hidden layers, $g(x) = \beta^\top x$ recovers linear Cox.

Estimation

How they fit it

- Estimate network weights θ by maximizing Cox partial likelihood, now over network parameters. Recover the baseline using standard Cox methods.

Why it matters today

- The blueprint: swap the linear risk score for a neural network, keep the partial likelihood.
- Revived and scaled by DeepSurv (Katzman et al., 2018), and used by attention/transformer models with Cox heads. Other survival heads use different objectives.

Katzman et al. (2018), DeepSurv, BMC Medical Research Methodology 18:24. DOI: 10.1186/s12874-018-0482-1.

Beyond the MLP: the tabular transformer

Beyond a multilayer perceptron (MLP): contextual category embeddings

> A plain category embedding is fixed. TabTransformer contextualizes it using the other categorical columns in the same record (Huang et al., 2020).

Architecture

- Each categorical token combines category information with column identity.
- Multi-head self-attention transforms tokens into contextual embeddings.
- Continuous features are normalized and concatenated with those embeddings.
- An MLP maps the combined representation to the prediction.

$$\text{categoricals} \xrightarrow{\text{embed}} E \xrightarrow{\text{Transformer}} \tilde{E} \parallel \text{norm}(\text{continuous}) \xrightarrow{\text{MLP}} \hat{y}$$

The attention mechanism: the heart of the transformer

The idea: let each feature update its representation using other features in the same record.

- Queries and keys produce data-dependent weights over tokens, usually including the feature itself. Before attention dropout, weights are nonnegative and sum to one.
- A head combines projected value vectors using these weights; the representation now depends on the patient's feature context.
- Weights are recomputed for each record rather than applying one fixed coefficient vector to every patient.
- Multiple heads learn different representation subspaces; a head is not guaranteed to represent a named clinical relationship.
- The complete block also has output projections, residual connections and feed-forward layers. It is more than a weighted average.

Attention for statisticians: a data-adaptive weighted average

Each row of E represents one categorical field. Query/key width is d_k:

$$Q = EW_Q, \quad K = EW_K, \quad V = EW_V, \qquad \alpha = \mathrm{softmax}_{\mathrm{row}}\left(\frac{QK^\top}{\sqrt{d_k}}\right).$$

Each head combines value vectors $m_j = V_j$ (before dropout):

$$\mathrm{output}_i = \sum_j \alpha_{ij} m_j, \qquad \alpha_{ij} \geq 0, \quad \sum_j \alpha_{ij} = 1.$$

- The weights are kernel-like, but generally asymmetric, not a covariance kernel.
- Unlike plain OLS/Cox coefficients, weights depend on the record. This can represent interactions; weights are not automatically feature importance or causal effects.

From attention to a transformer encoder

A transformer block combines attention with familiar components:

```
input  -->  Multi-Head Self-Attention  -->  Add & LayerNorm
       -->  Position-wise Feed-Forward  -->  Add & LayerNorm  --> output
```

- Residual connections help gradients flow through deep stacks.
- Layer normalization normalizes within a token's representation; it is not training-set standardization of each clinical variable.
- Tabular tokens need column identity, not an arbitrary visit order. Longitudinal tokens additionally need time/order information and masking to prevent future-data leakage.

Putting it together: transformer survival models

- Use a transformer as a feature extractor, then attach a survival objective.
- Option A: Cox/DeepSurv head, retaining proportional hazards:

$$\hat{g}(x) = w^\top z(x;\theta) \quad \Rightarrow \quad \text{negative log partial likelihood.}$$

- Option B: discrete-time head, not necessarily PH. Choose boundaries $0 = a_0 < a_1 < \cdots < a_K$ and predict, for $k = 1, \ldots, K$:

$$h_k(x) = \Pr(a_{k-1} \le T < a_k \mid T \ge a_{k-1}, x) = \text{sigmoid}\left(\eta_k(x;\theta)\right).$$

Train with a censoring-aware discrete-time likelihood and specify within-bin censoring treatment; a sigmoid alone does not handle censoring.

How do we evaluate these models?

Discrimination, prediction error and calibration:

- Harrell's C-index: risk ranking among comparable pairs; results depend on the censoring distribution.
- Uno's C: inverse-probability-of-censoring weighting (IPCW) for truncated concordance, not time-dependent AUC.
- Time-dependent AUC: horizon-specific discrimination with defined cases and controls.
- Brier score / integrated Brier score (IBS): squared prediction error, with censoring adjustment and supported time horizons.
- Calibration plots: predicted versus estimated observed survival, with uncertainty.

Tune on training/validation data; evaluate on held-out patients. Censoring weights must match the censoring assumptions.

Experiments: embedding categorical data

- The original experiments compare model structures on synthetic datasets; they have not been independently reproduced.
- Risk-function scenarios
  - Linear model
  - Nonlinear model
  - Nonlinear model with only interactions
- Structures compared
  - Attention (Transformer Cox) vs. feed-forward network (MLP Cox)
  - One-hot encoding vs. entity (column) embedding
  - Shared embedding vs. no shared embedding

Original experiments: evidence and comparison limits

- Generating code, complete simulation specifications and raw results were not supplied. The figures are historical source claims, not reproduced evidence.
- The bars have no uncertainty intervals; the MAE target and censoring treatment remain unresolved. They do not establish architectural superiority.
- Comparisons need matched tuning, independent-replicate uncertainty and a defined censored-data prediction target.
- In the interaction-only figure, the small displayed Transformer–MLP difference does not establish an attention benefit without uncertainty.

Synthetic data

Reported in the original source; not independently reproduced:

- Covariates: 7 categorical variables (Cat1–Cat7) and 3 continuous variables.
- Cont1, Cont2, Cont3 are normal with mean 0 and variances $1, 2, 3$.
- The source describes Weibull-generated times, but does not supply the event-time parameters or a separate censoring mechanism.
- Three risk scenarios: linear, nonlinear, and nonlinear with only interactions.

Experiment settings

Reported settings; original runs have not been reproduced:

- Same reported MLP architecture across models within each scenario:
  - Linear: 2 hidden layers, 16 neurons each. Nonlinear: 3 hidden layers.
- Transformer Cox: 1 encoder block, 2 heads, embedding width 8, shared width 2.
- Neural models: dropout 0.1, learning rate 0.01, weight decay 0.0001.
- 10-fold CV: 10% test, 18% validation and 72% training per iteration.
- Reported stopping rule: validation loss increases for 5 consecutive epochs.
- Metrics: C-index and mean absolute error (MAE).

Experiment results: linear scenario

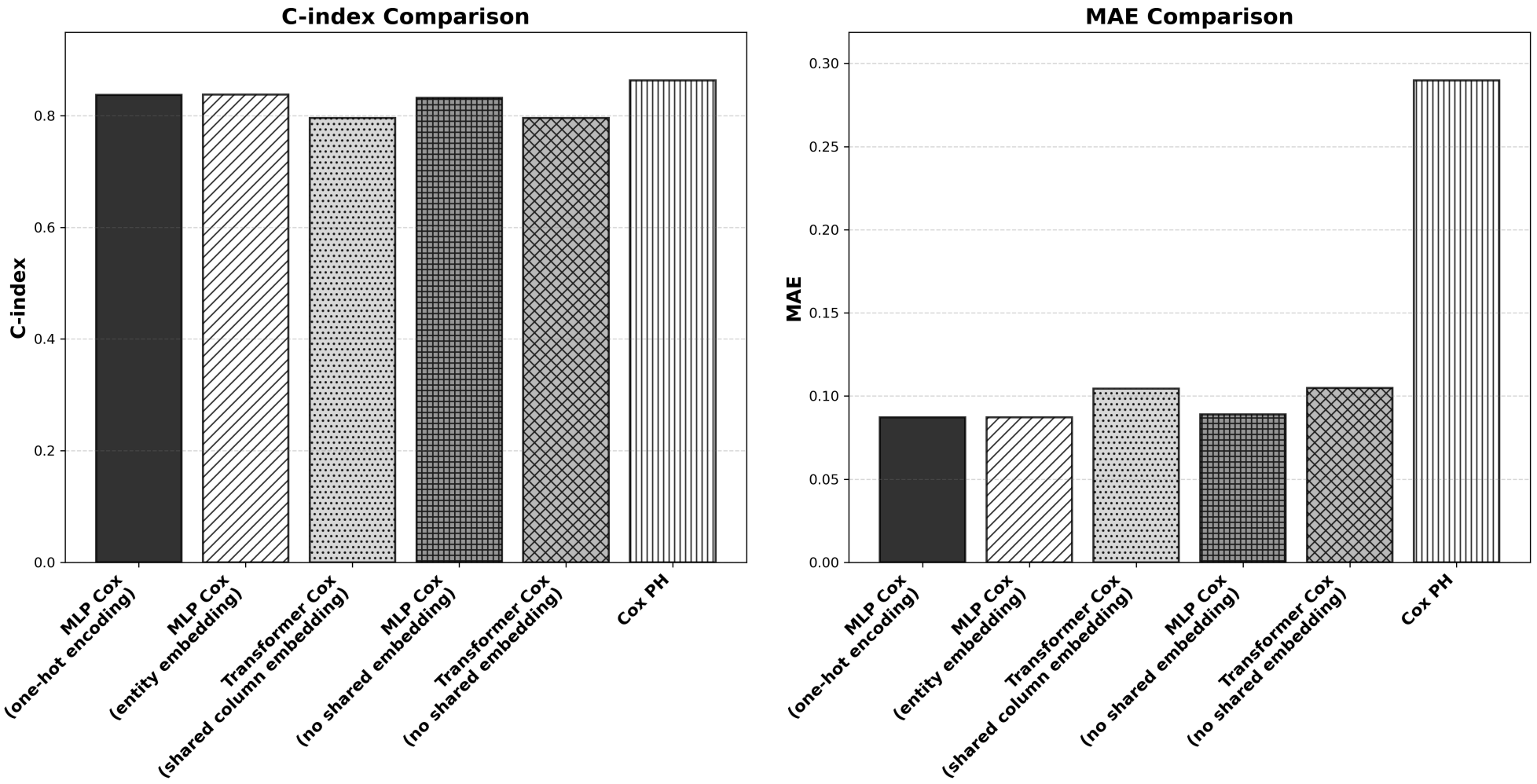


Figure 8.4: Performance comparison based on simulated linear models.

Experiment results: nonlinear scenario

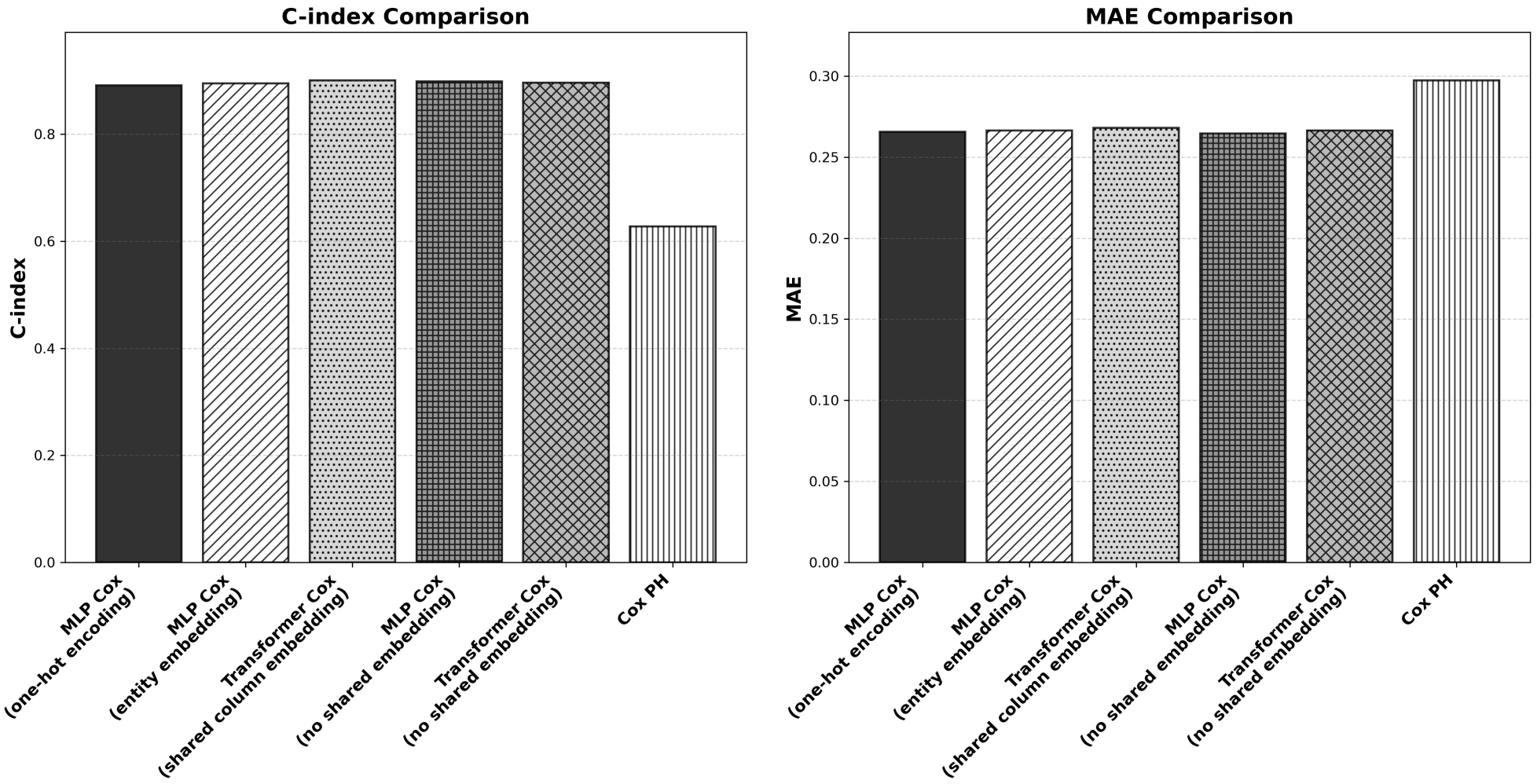


Figure 8.5: Performance comparison based on simulated nonlinear models.

Experiment results: nonlinear with only interactions

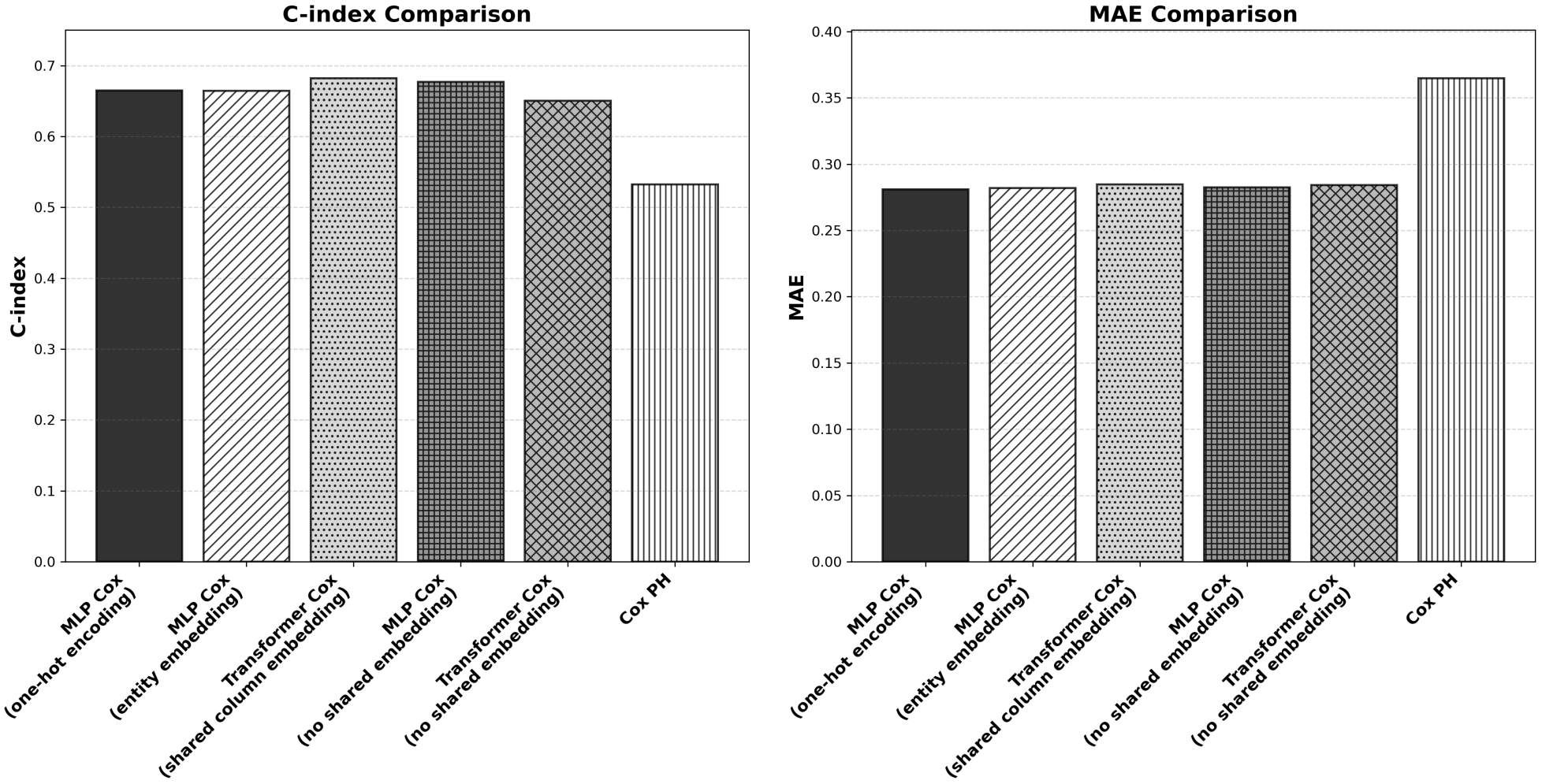


Figure 8.6: Performance comparison based on simulated nonlinear models with interaction only.

Implementation: building a Transformer–Cox model

Guided walkthrough: from equations to PyTorch

What we will build

A Transformer–Cox teaching pipeline: these excerpts illustrate the main components, not a complete training application.

The pipeline, in five moving parts:

1. Data prep: split patients; encode categorical and continuous covariates; sort training durations.
2. Model: shared-column/category embeddings → transformer → MLP → one log-risk $\hat{g}(x)$.
3. Loss: differentiable Cox partial likelihood with complete tied-time risk sets.
4. Baseline: training-cohort Breslow estimator → patient survival curves.
5. Evaluation: held-out Harrell's C-index, plus calibration and prediction error.

Step 1: data preparation

```
import numpy as np
import torch
from torch import nn
# 4 binary categorical columns + 5 training-standardized continuous columns.
def sort_by_descending_time(cat, cont, duration, event):
    order = np.argsort(duration, kind='stable')[::-1]  # descending
    return cat[order], cont[order], duration[order], event[order]
```

- Split patients first. Fit standardization on training data; integer category codes need a defined unknown-level policy.
- Inputs above are aligned NumPy arrays; durations are a one-dimensional finite numeric vector. At the end of an equal-time group, the prefix is the complete risk set, including censors at that time.
- Train with full training-cohort risk sets. Arbitrary independent-row minibatches change the objective; the loss below sorts internally.

Step 2: the model in code

```
class TransformerSurvival(nn.Module):
    def __init__(self, embeddings, predictor, num_continuous,
                 embedding_dim=8, n_heads=2, ff_dim=16, depth=1, dropout=0.1):
        super().__init__()
        self.embeddings, self.predictor = embeddings, predictor
        layer = nn.TransformerEncoderLayer(embedding_dim, n_heads,
                    ff_dim, dropout, batch_first=True)
        self.transformer = nn.TransformerEncoder(layer, num_layers=depth)
        self.norm_cont = nn.LayerNorm(num_continuous)

    def forward(self, x_categ, x_cont):
        z = self.transformer(self.embeddings(x_categ))
        z = z.flatten(start_dim=1)
        x = torch.cat([z, self.norm_cont(x_cont)], dim=1)
        return self.predictor(x).squeeze(-1)
```

Step 2: model excerpt requirements

- Supply embeddings and a multilayer perceptron (MLP). For this example: 4 categorical tokens of width 8, plus 5 continuous inputs.
- Embeddings must encode level and column identity. Category IDs are `torch.long`; continuous inputs and model weights need compatible floating dtypes and devices.
- Flatten 4 × 8 contextual features and append 5 continuous values. The MLP maps `[n,37]` to `[n,1]`; the class returns `[n]` log-risks.
- Embedding width must be divisible by the head count. No sigmoid is used for Cox log-risk.
- Continuous LayerNorm is optional, not feature-wise standardization. Use `model.eval()` before computing fitted scores.

Step 3: the heart: Cox partial likelihood as a loss

Minimize the event-mean negative Breslow log partial likelihood, with $D = \sum_i \delta_i > 0$ and $g_i = g(x_i; \theta)$:

$$\ell_{\mathrm{B}}(\theta) = \frac{1}{D} \sum_{i:\delta_i=1} \left[\log \sum_{j:\tilde{t}_j \geq \tilde{t}_i} e^{g_j} - g_i \right].$$

The repeated risk-set sums cost $O(n^2)$ naively. Sort by decreasing observed duration: the full risk set at u ends at the last row $b(u)$ of that equal-time group.

$$\log \sum_{j \in R(u)} e^{g_j} = \left[\mathrm{logcumsumexp}(g) \right]_{b(u)}.$$

Share this denominator across all events tied at u, including censors at u in the risk set. Sorting costs $O(n \log n)$; accumulation after sorting costs $O(n)$.

Step 3 (cont.): the loss, in PyTorch

```
class CoxPHLoss(nn.Module):
    def forward(self, log_risk, duration, event):
        g, t = log_risk.reshape(-1), duration.reshape(-1)
        e = event.reshape(-1).bool()
        order = torch.argsort(t, descending=True)
        g, t, e = g[order], t[order], e[order]
        if not torch.any(e):
            raise ValueError("At least one event is required")
        _, counts = torch.unique_consecutive(t, return_counts=True)
        ends = counts.cumsum(0) - 1
        den = torch.logcumsumexp(g, dim=0)[ends]
        den = torch.repeat_interleave(den, counts)
        return (den[e] - g[e]).mean()
```

Step 4: from log-risk to a survival curve (Breslow)

At distinct training event times u_m, let d_m count events:

$$\hat{H}_0(t) = \sum_{u_m \le t} \frac{d_m}{\sum_{j:\tilde{t}_j \ge u_m} e^{g_j}}, \qquad \hat{S}(t \mid x) = \exp[-\hat{H}_0(t) e^{g(x)}] = \hat{S}_0(t)^{\exp(g(x))}.$$

```
# Compute flat g and g_new from the same fitted model in model.eval().
# t, e, g: aligned training vectors; e: bool; require at least one event.
with torch.no_grad():
    u = torch.unique(t[e], sorted=True)
    log_dH = torch.stack([
        torch.log((t[e] == v).sum().to(g.dtype))
        - torch.logsumexp(g[t >= v], dim=0) for v in u
    ])
    log_H = torch.logcumsumexp(log_dH, dim=0)
    S = torch.exp(-torch.exp(log_H[:, None] + g_new[None, :]))
```

Step 5: evaluation & recap

Harrell's C uses comparable pairs, with half credit for tied risk scores; define time-tie handling. About 0.5 is chance ranking, not calibrated survival.

$$x \to \text{embed} + \text{transformer} \to \hat{g}(x) \xrightarrow{\text{CoxPHLoss}} \text{train} \xrightarrow{\text{Breslow}} \hat{S}(t \mid x)$$

- Breslow output: rows are training event times; columns are new patients. Use right-continuous lookup within supported follow-up.
- Check calibration and prediction error as well as ranking. C-index can be calculated directly from log-risk; PH still applies.
- Next: reduce the score to a linear layer and compare with a reference fit.

Validation: is the deep-learning Cox really the Cox model?

Building the bridge back to classical survival analysis

The test: shrink the network to one linear layer

A Transformer–Cox model puts a flexible score inside Cox's likelihood. Reduce it to one linear layer: no embeddings, transformer or hidden units.

$$\hat{g}(x) = \beta^\top x \quad \Longrightarrow \quad h(t \mid x) = h_0(t)\exp(\beta^\top x).$$

- Historical test specification: $\beta = (0.8, -0.6, 0.4, 0.2, -0.3)$ with ~30% independent censoring. Sample size, seeds and full generator are missing.
- Fit the same training data by Adam on `CoxPHLoss` and by `lifelines.CoxPHFitter` (Newton–Raphson).
- For a clean check: identical features, no bias or penalty, untied event times, and converged fits. Finite samples need not recover truth exactly.

Usual lifelines coefficient fitting uses Efron ties, despite a Breslow baseline estimator. Our loss uses Breslow: match conventions or test untied event times.

Historical comparison with the classical package

Historical source values; unverified, not reproduced:

coef	true	PyTorch	lifelines
x_0	0.800	0.812	0.812
x_1	-0.600	-0.594	-0.594
x_2	0.400	0.468	0.468
x_3	0.200	0.267	0.270
x_4	-0.300	-0.306	-0.304

- Maximum displayed PyTorch–lifelines difference: $\max |\Delta\hat{\beta}| = 0.003$; not evidence of equivalence.
- Reported held-out C-index: 0.7365 vs. 0.7366; reported oracle: 0.738.

The bridge, made concrete

- When $\hat{g}(x) = \beta^\top x$, the model family is classical Cox. Numerical agreement requires matched ties, penalties and convergence, not merely a linear layer.
- The transformer changes $\hat{g}(x)$ to learn nonlinearities and interactions; classical Cox can also use engineered nonlinear features. Predictive benefit requires independent evaluation.
- Cox loss and Breslow inference can be shared across encoders, with the same risk-set and PH assumptions.

$$\underbrace{\beta^\top x}_{\text{classical Cox}} \xrightarrow{\text{flexible predictor}} \underbrace{g(x;\theta)}_{\text{Transformer--Cox / DeepSurv}}$$

Recent advances I: from architecture to foundation models

The field includes both task-specific architectures and large pretrained models for structured EHR and claims data.

- SurvTRACE: Wang & Sun, ACM-BCB 2022. Transformer representations with time-discretized, piecewise-constant hazards for single or competing events, plus auxiliary tasks. Not Cox PH, but not assumption-free; attention plots are not causal explanations.
- TransformerJM: Lin & Luo, Statistics in Medicine 2022. Joint longitudinal/survival prediction using temporally masked attention, supporting dynamic prediction.
- MOTOR: Steinberg, Fries, Xu & Shah, 2023 preprint (arXiv:2301.03150). Time-to-event pretraining on up to 55 million patient records in EHR and claims data, then linear probing or fine-tuning.

Recent advances II: competing risks, scale, and open questions

- Cause-specific hazard transformer: Oliver et al., BMC Bioinformatics 2024. Discrete-time cause-specific hazards for competing events, without a PH restriction; published experiments used singleton-length inputs.
- SurvivEHR: Gadd, Gokhale et al., npj Digital Medicine 2026. Decoder-only EHR foundation model for next-event prediction and continuous-time competing risks. Downstream calibration and transportability still require evaluation.
- Open needs: Wiegrebe et al., Artificial Intelligence Review 57:65 (2024), reviewed literature through 2022. It found limited complex-event/censoring coverage and underreported calibration, motivating standardized benchmarking—not a census of 2026 methods.

From Bayesian Inference to Diffusion Models

Part II: Generative models for synthetic data

Roadmap: from the familiar to the frontier

The connecting question: how do we approximate distributions and draw useful samples?

- Bayes' rule gives a posterior we often cannot compute in closed form.
- Classical sampling: importance sampling, MCMC, Gibbs: approximate expectations or draws from a specified target.
- Variational inference trades difficult posterior computation for optimization.
- VAE = learned latent-variable model with neural, amortized variational inference.
- Diffusion = a hierarchical latent-variable model with many learned reverse steps, related to the VAE construction.

Basic Bayesian concept

We treat unknown parameters θ as random and update our beliefs with data y:

$$p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}, \qquad p(y) = \int p(y \mid \theta)p(\theta)\, d\theta.$$

- Prior $p(\theta)$: beliefs before seeing data.
- Likelihood $p(y \mid \theta)$: the assumed data model.
- Posterior $p(\theta \mid y)$: updated beliefs, given that model.
- The evidence can be an intractable high-dimensional integral. If likelihood and prior are evaluable, we know the posterior up to a constant:

$$p(\theta \mid y) \propto \widetilde{p}(\theta) = p(y \mid \theta)p(\theta).$$

Sampling I: Importance sampling

Sample from an easy proposal $q(\theta)$ whose support covers the target p, and reweight:

$$\mathbb{E}_p[f] = \mathbb{E}_q\left[f(\theta)\frac{p(\theta)}{q(\theta)}\right] \approx \frac{\sum_s w_s f(\theta^{(s)})}{\sum_s w_s}, \qquad w_s = \frac{\tilde{p}(\theta^{(s)})}{q(\theta^{(s)})}.$$

- Draw $\theta^{(s)} \sim q$. Self-normalization cancels the unknown evidence; generally biased at finite sample size, consistent under suitable integrability conditions.
- No Markov chain: independent proposals are parallelizable. Weight-based effective sample size is a diagnostic, not a guarantee.
- Limitation: poor overlap or light proposal tails can concentrate weight on a few draws, especially in high dimensions.

Sampling II: Markov chain Monte Carlo (MCMC)

Build a Markov chain with target $p(\theta \mid y)$ as its stationary distribution. Metropolis–Hastings proposes $\theta' \sim q(\theta' \mid \theta)$ and accepts with

$$\alpha = \min\left(1, \frac{\widetilde{p}(\theta')q(\theta \mid \theta')}{\widetilde{p}(\theta)q(\theta' \mid \theta)}\right).$$

- Ratios of unnormalized densities cancel the evidence.
- Suitable irreducibility and ergodicity conditions support convergence; burn-in alone does not guarantee posterior draws.
- MCMC can explore targets that defeat importance sampling, but high-dimensional mixing may still be difficult.
- Cost: autocorrelation and slow mixing. Check multiple chains, convergence diagnostics and effective sample size for each analysis.

Sampling III: Gibbs sampling

Each exact full-conditional update is an MH step with acceptance probability one. A systematic sweep uses the latest available coordinates:

$$\theta_i^{(s+1)} \sim p\Big(\theta_i \mid \theta_{1:i-1}^{(s+1)}, \theta_{i+1:d}^{(s)}, y\Big), \qquad i = 1, \ldots, d.$$

- No proposal tuning or rejection for exact conditional draws; conditionals must be sampleable.
- Especially useful with tractable, often conjugate, full conditionals in hierarchical models; BUGS/JAGS combine Gibbs with other updates when needed.
- Strongly correlated parameters can mix slowly; blocking may help.

The turning point: stop sampling, start optimizing

- MCMC can be asymptotically exact under appropriate conditions, but correlated draws and convergence assessment can make repeated inference expensive.
- Alternative idea: choose a tractable family of distributions and optimize an approximation to the posterior. Neural networks can parameterize rich approximations.
- Sampling $\longrightarrow$ optimization is a useful bridge from Bayesian computation to modern generative models—not the elimination of sampling. Variational training often still uses Monte Carlo estimates.
- The tradeoff: computational convenience versus approximation, optimization and model error.

Variational inference: integration → optimization

For model $p_\theta(x, z)$, approximate its posterior with $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \mathcal{E}(\theta, \phi; x) + \mathrm{KL}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)),$$

$$\mathcal{E}(\theta, \phi; x) = \mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)}\right].$$

- Since KL is nonnegative, $\mathcal{E}$ is the evidence lower bound (ELBO).
- At fixed θ, maximizing the ELBO over ϕ minimizes this posterior KL.
- If θ is also learned, the model and its target posterior both change.
- Gradient optimization can scale well, but the family and optimization constrain approximation quality.

The VAE: variational inference with neural nets

Two neural networks, trained together by maximizing the ELBO across observations:

- Encoder $q_\phi(z \mid x)$: maps data to an approximate posterior distribution over latent z; inference is amortized across observations.
- Decoder $p_\theta(x \mid z)$: specifies a distribution over data given z. Choose the prior $p(z) = \mathcal{N}(0, I)$.

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]}_{\text{reconstruction term}} - \underbrace{\text{KL}(q_\phi(z \mid x) \,\|\, p(z))}_{\text{prior regularization}}.$$

The KL regularizes the approximate posterior; it does not make every latent code or generated observation exact.

The VAE as a sampler

- Classical samplers approximate expectations or draws for a specified target.
- A VAE learns a distribution from training data. To generate a new observation:

$$z \sim \mathcal{N}(0, I) \quad \longrightarrow \quad x \sim p_\theta(x \mid z).$$

- These are samples from the fitted generative model, not guaranteed draws from the true data distribution or a Bayesian posterior over parameters.
- Some VAE image models produce smooth or blurry outputs. This depends on the observation likelihood, representation and training—not simply one latent draw or a universally inadequate posterior.
- Diffusion offers another construction: a sequence of learned denoising transitions.

From one step to many: diffusion = hierarchical VAE

Replace one latent decoding stage with a chain of reverse transitions:

- Forward (fixed in standard DDPM): add noise, giving latent states $x_1, \ldots, x_T$.
- Reverse (learned): gradually denoise, usually with one time-conditioned network shared across steps.
- The hierarchical-VAE view concerns the joint latent-variable model; each step is not a separately trained VAE.

	latent(s)	**one leap?**
VAE	single z	one latent decoding stage
Diffusion	chain $x_1, \ldots, x_T$	many learned reverse steps

The core idea in one picture

- Forward (fixed): gradually add Gaussian noise until the terminal distribution is close to a simple Gaussian prior.
- Reverse (learned): remove noise through stochastic transitions, turning a prior draw into a data-like sample.
- Analogy: ink spreading in water is easy to describe forward. A diffusion model learns a probabilistic reversal, not a guaranteed recovery of the original ink configuration.
- The specified forward and reverse models are Markov chains: each transition uses the current state and time.
- Here $t = 1, \ldots, T$ indexes diffusion steps; clinical event time is a different quantity.

Forward process: data → noise (no learning)

Add Gaussian noise at each step $t = 1, \ldots, T$, with $0 < \beta_t < 1$:

$$q(x_t \mid x_{t-1}) = \mathcal{N}\Big(\sqrt{1-\beta_t}\, x_{t-1}, \beta_t I\Big).$$

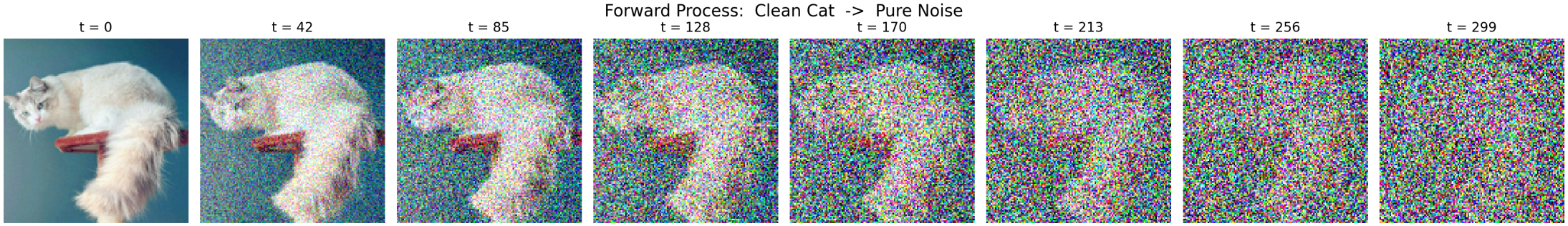
This linear Gaussian chain gives a closed-form jump from x_0 to any step:

$$q(x_t \mid x_0) = \mathcal{N}\Big(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t) I\Big), \qquad \alpha_t = 1 - \beta_t, \quad \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s.$$

Reparameterization:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

Forward process, visualized



A supplied illustration of increasing noise. For suitably small $\bar{\alpha}_T$, the terminal distribution is approximately $\mathcal{N}(0, I)$, not generally exactly Gaussian at finite T.

Reverse process: noise → data (learned)

DDPM approximates reverse transitions by Gaussians; the true reverse need not be Gaussian:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \sigma_t^2 I), \qquad \mu_\theta = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t \epsilon_\theta(x_t, t)}{\sqrt{1 - \bar{\alpha}_t}} \right).$$

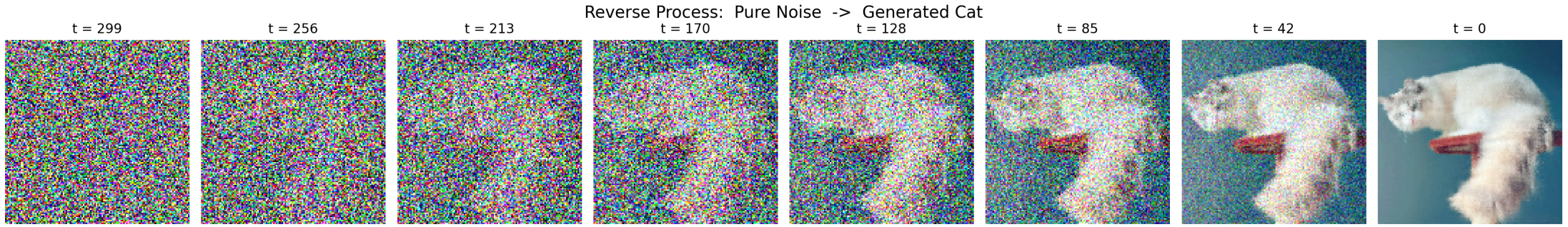
Train by predicting the noise added to training records x_0:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t, x_0, \epsilon} \|\epsilon - \epsilon_\theta(x_t, t)\|^2, \qquad t \sim \text{Unif}\{1, \ldots, T\}.$$

This simplified loss drops ELBO timestep weights. Sampling also needs a variance schedule and a specified data-endpoint likelihood.

Ho, Jain & Abbeel (2020), DDPM, arXiv:2006.11239.

Reverse process, visualized



Conceptual illustration: start from noise and denoise toward a data-like image. Generation does not guarantee recovery of the original cat.

Diffusion vs. other generative models

Approach	Analogy	Result
GAN	Builder and critic compete over proposed houses.	Can generate sharp samples; training instability and mode collapse are possible.
VAE	Draw a latent blueprint, then build from it.	A probabilistic generator; detail depends on likelihood and training.
Diffusion	Learn to reverse successive noisy demolition steps.	Iterative generation; no guaranteed perfect reconstruction.

These are teaching analogies, not a universal ranking. Compare fidelity, coverage, computation and downstream utility under a matched evaluation.

SurvDiff: a diffusion model for generating synthetic survival data

An end-to-end diffusion model that jointly generates mixed covariates, observed follow-up durations and event indicators: synthetic records for survival-model evaluation.

$$Y = \min(T_{\text{event}}, C), \qquad \delta = \mathbf{1}(T_{\text{event}} \leq C).$$

For a censored record, Y is the observed censoring duration—not the unobserved event time. Modeling (x, Y, δ) does not by itself identify latent uncensored survival times.

> Brockschmidt, Schröder & Feuerriegel · LMU Munich & Munich Center for Machine Learning · arXiv:2509.22352v3 (2026-07-24); first posted 2025.

The method: joint diffusion + a Cox-style loss

- Jointly generate $(x^{\text{cont}}, x^{\text{disc}}, Y, \delta)$.
- Continuous covariates and duration: variance-exploding Gaussian noise, different from the preceding DDPM schedule.
- Discrete covariates and event indicator: masked diffusion with learned reverse unmasking.
- An auxiliary MLP produces log-risk r_i from denoised covariates, including discrete probability vectors:

$$\mathcal{L}_{\text{surv}} = -\sum_{i:\delta_i=1} w_i \log \frac{e^{r_i}}{\sum_{j\in R(Y_i)} e^{r_j}}, \quad R(u) = \{j : Y_j \geq u\},$$
$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{diff}} + \lambda_{\text{surv}}\mathcal{L}_{\text{surv}}.$$

Weights w_i downweight late events; they are not censoring weights. The capped λ_{surv} is calibrated after warm-up, then fixed.

SurvDiff: reported results and limits

- Baselines: NFlow, TVAE, CTGAN, TabDiff, SurvivalGAN and Ashhad.
- Lowest reported mean marginal JS distance on AIDS, GBSG2 and METABRIC: not every fidelity metric.
- Lowest aggregate event-time-stratified covariate divergence (ETD); ETD tables exclude Ashhad.
- Train-on-synthetic/test-on-real results favor SurvDiff on AIDS/GBSG2; TabDiff has better downstream means on METABRIC.

Data considerations and AI agents for automation

Part III: Data foundation and AI agents

Data foundation for evidence-driven clinical development

Statistics already has a strategic role; AI may extend its execution capacity.

- Living evidence foundation: Continuously integrates clinical trials, publications, internal protocols/data, RWD, guidelines, and regulatory knowledge into a curated, versioned therapeutic-area knowledge base.
- Clinical trial intelligence: Supports trial design, feasibility, site/KOL selection, trial emulation, and external control arms; causal assumptions and fitness for use still require assessment.
- AI-driven documentation: Can assist drafting and maintaining protocols, SAPs, ICFs, CSRs, and dossiers; traceability, statistical rigor, and compliance require verification.

Data engineering for AI agents

CDISC standards vs. relational databases: Is CDISC built for AI agents?

- Clinical data manager: CDISC provides common clinical-research data standards supporting consistency, traceability, and regulatory review; conformance alone does not guarantee acceptance.
- Deep-learning engineer: AI agents need explicit data grain, semantics, and validated joins: not silent reshaping or guesses.
- What we will discuss:
  - What CDISC actually is (and was designed for).
  - CDISC against the relational normal forms (1NF / 2NF / 3NF).
  - What that structure means for AI agents (text-to-SQL, tool use, retrieval).
  - Whether a relational database is the better substrate, and the caveats.

What CDISC is, and what it is *not*

The CDISC family

- CDASH: data *collection* standards.
- SDTM: *tabulation* model; domains such as DM, AE, LB, VS, EX, CM have domain-specific row grain.
- ADaM: *analysis* datasets and metadata, including ADSL, BDS, and OCCDS, with traceability to source data.
- Define-XML + Controlled Terminology: machine-readable metadata and vocabularies.
- Key point. CDISC spans collection, representation, exchange, analysis, and review—not a prescribed normalized database schema. XPT is a common submission format; it does not dictate operational storage.

A 30-second normalization refresher

A candidate key is a minimal superkey; a prime attribute belongs to at least one candidate key.

- First Normal Form (1NF): One value from the declared domain per attribute; no repeating groups.
- Second Normal Form (2NF): In 1NF, every non-prime attribute depends on each whole candidate key, not on a proper subset.
- Third Normal Form (3NF): In 1NF, every nontrivial dependency $X \to A$ has X a superkey or A prime.

$$\mathcal{R}_{3NF} \subseteq \mathcal{R}_{2NF} \subseteq \mathcal{R}_{1NF}$$

CDISC under the normalization lens

- SDTM/ADaM commonly use scalar-valued tabular records, but their standards do not require strict relational 2NF or 3NF. Assess each relation using its grain, candidate keys, and functional dependencies.
- They support regulatory exchange, traceability, reviewability, controlled terminology, and analysis readiness, not fully normalized operational database design. Repeated subject attributes can be intentional.

What this means for an AI agent

- Where CDISC genuinely helps agents
  - Semantic standardization
  - Machine-readable metadata
  - Traceability and reviewability
- Where CDISC can challenge agents
  - File exchange alone does not enforce relational constraints
  - Parameter/value and supplemental-qualifier structures can require reshaping
  - Intentional redundancy needs reconciliation
  - Source-versus-derived distinctions need explicit metadata

Will a relational database work better?

- A governed relational layer can add what exchange files do not enforce:
  - Primary and foreign keys: explicit identities and references
  - Constraints and referential integrity: reject specified invalid states
  - A versioned source of truth: reduce conflicting duplicate facts
  - A documented schema: support text-to-SQL and auditable tool use

The AI Agent Operating Model

- Humans lead: set objectives, make decisions, review outputs, define priorities, and establish guardrails
- Agents execute within approved scope: synthesize evidence, extract data, run models, draft documents, and perform QC in parallel
- The loop can improve: reviewed outputs inform subsequent tasks, with controlled feedback and versioning
- Intended impact: faster cycle times while retaining human judgment, statistical accountability, and regulatory oversight; productivity gains require comparative evaluation.

Existing methods for automation

Method	Contribution	Important limit
ClinAgent	Clinical programming tools	Code that runs can still answer the wrong question
PROVE (preprint)	Interpret reports and run QC	Checks depend on extraction, mapping, and rule coverage
CAVE-Onc	Check cross-domain oncology rules	Consistent datasets may share the same scientific error

Author-reported methods; their experiments were not independently reproduced here.

Link scientific intent to executable evidence

Target architecture; the evaluated prototype combines everything in one source-linked contract.

```
Protocol and SAP in Markdown
    -> TLF specification (JSON): WHAT to report
    -> Programming specification (JSON): HOW to calculate
    -> R methods/packages (OOP programming): results and displays
    -> Standalone R + CSV/TXT/PDF
    -> Independent statistical and display review
    -> Versioned delivery and evidence
```

Explicit references connect scientific requirements to each result.

SAP Markdown preserves the scientific reason

Illustrative synthetic excerpt

```
# SAP-01
## endpoint-orr
Population: all randomized subjects (ITT).
Numerator: confirmed complete or partial response (CR/PR).
No qualifying response: retain as a nonresponder.
Report n/N, percent, two-sided 95% exact interval by arm.
## confirmed-response-rule
Two eligible CR/PR assessments >=28 days apart, before
progression; only CR/PR at intervening eligible assessments.
```

TLF specification in JSON: what to report

Illustrative proposed schema

```
{
  "output_id": "T06", "sap_ref": "SAP-01/endpoint-orr",
  "population_ref": "ITT", "method_ref": "METHOD-ORR-01",
  "programming_spec_ref": "PS-T06-v1",
  "columns": ["arm", "n", "N", "percent", "lower", "upper"],
  "footnotes": ["All randomized subjects are in the denominator."]
}
```

Programming specification JSON: how to calculate

Illustrative proposed schema

```
{
  "programming_spec_id": "PS-T06-v1", "output_ref": "T06",
  "sap_ref": "SAP-01/endpoint-orr", "method_id": "METHOD-ORR-01",
  "inputs": ["subject", "response"], "population_ref": "ITT",
  "analysis_unit": "subject", "derivation_ref": "confirmed-response-rule",
  "analysis": {"function": "stats::binom.test", "conf.level": 0.95}
}
```

Follow one result through the chain

Stage	Linked requirement or action
SAP: endpoint ORR	Confirmed response among all randomized subjects (ITT)
TLF	Arm-specific n/N, percentage, exact interval, footnote
Program	One subject record; confirmation; retain nonresponders
R calculation	Count responders and ITT subjects; calculate interval
Result/display	Preserve full precision; apply display rounding
Independent checks	Verify subjects, denominator, interval, and text

AI-generated work and independent checks

- Read sources → propose choices → validate and compile → independently review results.
- If scientific requirements are unclear, seek clarification; if unresolved, pause and document why.
- Keep reference answers and acceptance rules separate from the authoring AI.

Evaluation design

One synthetic oncology example: engineering tests, not a comparison of live AI models.

- Test set: 12 tables, 2 figures, and 2 listings.
- Methods: descriptives, Clopper–Pearson, Fisher exact, Kaplan–Meier, and pointwise log-log Greenwood intervals.
- Compare R with separately written Python/SciPy calculations.
- Test source changes, data perturbations, injected defects, and regeneration.
- Retain results, logs, source versions, and review status.

Test for invariance and sensitivity

Source-reported test family	Recorded scope; all passed
Source counterfactuals	80 output comparisons; 33 expected effects
Targeted data variants	11 datasets; 56 effects; 18 hand assertions
Sample-size boundaries	N = 0, 1, 47, 96; 64 output comparisons
Trusted synthetic-PDF control	2 synthetic PDFs; 16 output comparisons

Evaluation results

Challenge category	Full pass	Blocked	Failed
Boundary data	6	0	0
QC injections / clean controls	4	8	0
Authoring / source blocking	2	4	0
Security / evidence controls	3	5	0
Total	15	17	0

Future collaboration: an open-source AI benchmark

Compare AI models and agent workflows on simulated clinical data.

- Shared data: realistic trial structures, missingness, censoring, and injected errors.
- Common tasks: derive analysis datasets, endpoints, tables, listings, and figures.
- Independent checks: verified reference results and a hidden test set.
- Fair comparisons: common data, tools, and budgets; record all attempts and human help.
- Measures: accepted outputs / all assignments; correctness, time, and cost.
- Collaboration: co-design a pilot; release generators, tasks, and scoring code.

Thank you!

Questions or interested in collaborating?

Please feel free to reach out to kui.shen@bayer.com.