# Bridge Deep Learning and Clinical Development

## From Transformers to Time-to-Event Models

Kui Shen, PhD · June 2026

# Disclaimer

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

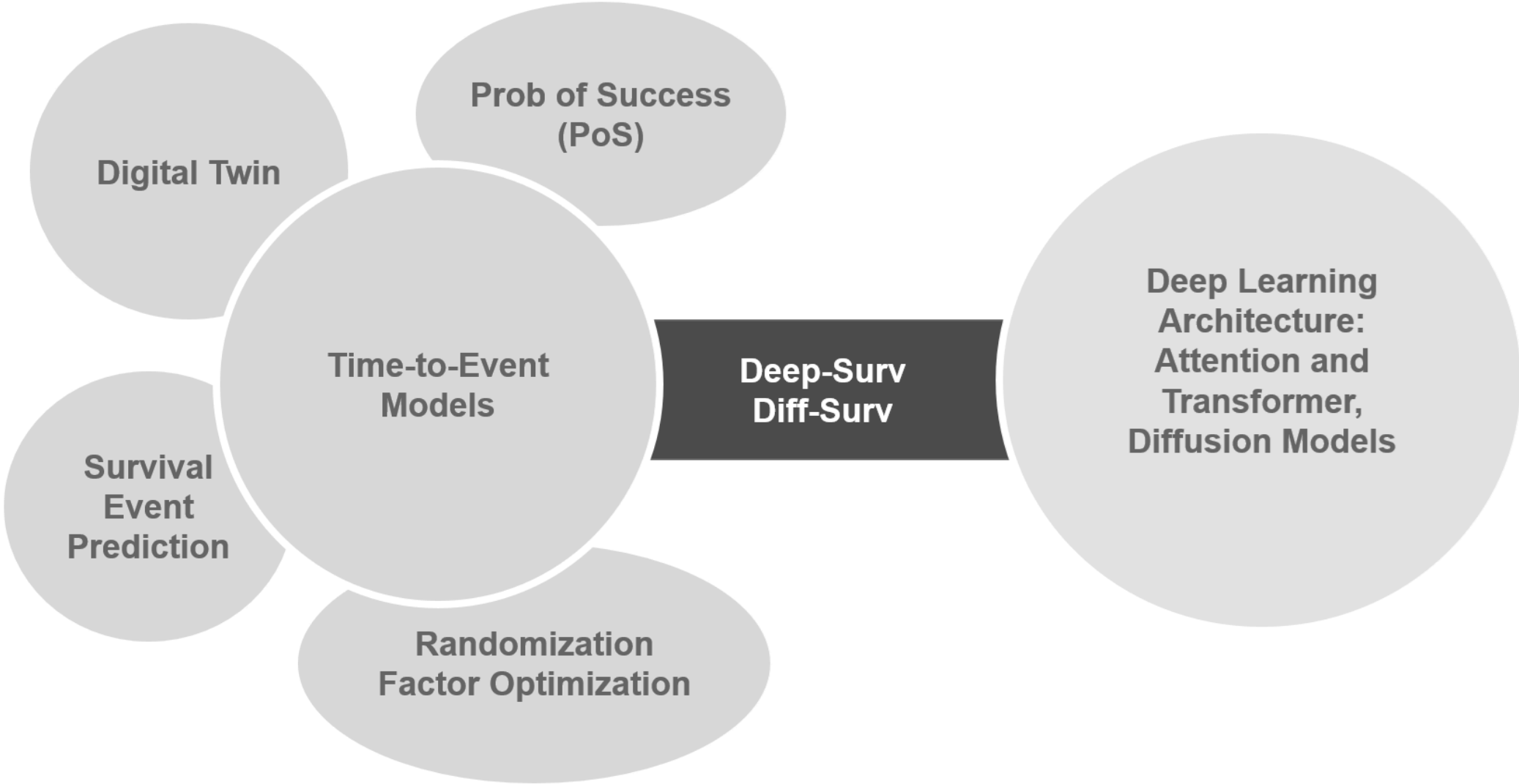
# The evolving role of clinical statistics in the AI era: a paradigm shift

- **Traditional role:** technical support centered on study design, trial monitoring, health-authority submissions, and statistical deliverables.
- **AI-era vision:** strategic decision-makers who shape the entire clinical development lifecycle: from early-phase planning through regulatory approval and beyond.
- **Key barrier to transformation:** Clinical development remains overly reliant on internal clinical trial data, with limited capability to systematically leverage external sources such as ClinicalTrials.gov, PubMed, and public or licensed databases, and insufficient advanced modeling capability for large-scale data integration and analysis..

# The opportunity in the AI era

- **The transformation:** elevate clinical statistics from study design to a strategic partner.
- **The solution:** integrate external real-world sources with clinical data, and build AI-powered models on the unified dataset to enable clinical decision support across the entire development lifecycle.
    - Deep learning algorithms for survival modeling
    - Clinical knowledge database and relational database
    - AI agent operating model

# Time-to-event models serve as the bridge



# Deep learning approaches for survival modeling

## Part I: Transformer-based models

# Survival fundamentals: a one-slide recap

For event time $T$ (death, relapse, failure), we model:

$$S(t) = \Pr(T > t), \qquad h(t) = \lim_{\Delta t \to 0} \frac{\Pr(t \le T < t + \Delta t \mid T \ge t)}{\Delta t}$$

The hazard and survival are two views of the same object:

$$h(t) = \frac{f(t)}{S(t)} = -\frac{d}{dt} \log S(t), \qquad S(t) = \exp\Big( - \int_0^t h(u)\, du \Big)$$

# The Cox proportional hazards model

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

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

Estimate $\beta$ without specifying $h_0(t)$ via the partial likelihood:

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

# The one limitation we want to relax

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

This says the log-hazard ratio is linear and additive in the covariates.

What it cannot capture:

- Nonlinear effects (e.g., a U-shaped risk in a biomarker level).
- Interactions (drug effect that depends on a genotype).
- High-dimensional, raw inputs (images, mutations).

Classically we patch this with splines, manual interaction terms, and feature engineering. Deep learning learns that flexible function automatically.

# A neural network model for survival data

- Faraggi & Simon (1995): the origin of neural network survival analysis
- The classical work that first married the Cox model with a neural network, the conceptual ancestor of DeepSurv.

> David Faraggi & Richard Simon. *Statistics in Medicine*, 14: 73–82 (1995).

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

**The limitation:** the Cox model forces the log-risk to be linear in the covariates:

$$h(t \mid x) = h_0(t) \exp\left(\beta^\top x\right)$$

**The proposal**: replace the linear predictor $\beta^\top x$ with the output of a feed-forward network $g(x;\theta)$:

$$h(t \mid x) = h_0(t) \exp\left(g(x;\theta)\right)$$

The network output becomes the log-risk score, so nonlinearities and covariate interactions are learned automatically, yet the model still sits inside the proportional-hazards framework. With no hidden layer, it reduces to Cox.

# Estimation, findings & lasting legacy

## How they fit it

- Estimate the network weights $\theta$ by maximizing the Cox partial likelihood, the same objective as Cox, now over network parameters. The baseline hazard was handled as in standard Cox.

## Why it matters today

- The blueprint — *"swap the linear risk score for a neural network, keep the partial likelihood"* — is exactly what modern methods build on.
- Revived and scaled by *DeepSurv* (Katzman et al., 2018) and extended by attention/transformer survival models.

# Beyond the MLP: the tabular transformer

## The idea: contextual embeddings for categorical features

> A plain embedding gives each category one fixed vector, regardless of the other columns.
> TabTransformer makes those vectors **context-aware* (TabTransformer, Huang et al, 2020)

## Architecture

- Each **categorical** feature → a column embedding.
- A stack of Transformer (multi-head self-attention) layers transforms these into contextual embeddings that depend on the other categoricals.
- **Continuous** features are normalized and concatenated with the contextual embeddings.
- A plain MLP maps the combined vector to the prediction.

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

# The attention mechanism: the heart of the transformer

- Each input element is projected into a query, key, and value:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

- Output = a weighted average of valueswhere weights come from query–key similarity:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

- Multi-head attention runs several such maps in parallel to capture different relations.

# From attention to a transformer encoder

A transformer block stacks attention with simple, well-tested pieces:

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

- Residual connections: keep gradients flowing in deep stacks.
- Layer normalization: stabilizes training (per-sample standardization).
- Positional encoding: injects *time/order* so the model knows visit sequence.

# Putting it together: transformer survival models

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

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

- Option B: discrete-time head (relaxes proportional hazards): partition follow-up into intervals $t_1 < \cdots < t_K$ and predict an interval hazard

$$h_k(x) = \Pr(T \in [t_k, t_{k+1}) \mid T \geq t_k,\ x) = \text{sigmoid}\big(\eta_k(x;\theta)\big)$$

# How do we evaluate these models?

Discrimination and calibration:

- Harrell's C-index: probability that predicted risk orders a comparable pair correctly (concordance). Time-dependent variants: Uno's C.
- Time-dependent AUC: discrimination at a chosen horizon $t$.
- Brier score / Integrated Brier Score (IBS): squared error of $\hat{S}(t \mid x)$, with IPCW (inverse-probability-of-censoring weights) to handle censoring.
- Calibration plots: predicted vs observed survival in risk strata.

# Experiments: embedding categorical data

- Through experiments, we analyze the performance of various structures on different synthetic datasets.
- Risk-function scenarios
  - Linear model
  - Nonlinear model
  - Nonlinear model with only intersections
- Structures compared
  - Attention (Transformer Cox) vs. feed-forward network (MLP Cox)
  - One-hot encoding vs. entity (column) embedding
  - Shared embedding vs. no shared embedding

# Synthetic data

- Covariates: 7 categorical variables (Cat1–Cat7) and 3 continuous variables.
- Cont1, Cont2, Cont3 are normal random variables with mean $0$ and variance $1, 2, 3$.
- The censored/uncensored time is generated using a Weibull distribution.
- Three risk-function scenarios: linear, nonlinear, and nonlinear with only intersection.

# Experiment settings

- All neural-network models share the same MLP predictor structure.
  - Linear experiments: MLP predictor with 2 hidden layers, 16 neurons each.
  - Nonlinear experiments: MLP predictor with 3 hidden layers.
- Transformer Cox: 1 transformer unit with head $= 2$, embedding dim $= 8$, shared dim $= 2$.
- All models: dropout rate $= 0.1$, learning rate $= 0.01$, weight decay $= 0.0001$.
- 10-fold cross-validation: each iteration uses 10% as test, 18% of total as validation, and 72% of total as training.
- Early stopping when validation loss increases for 5 consecutive epochs.
- Evaluation metrics: C-index and mean absolute error (MAE).

# Experiment results: linear scenario

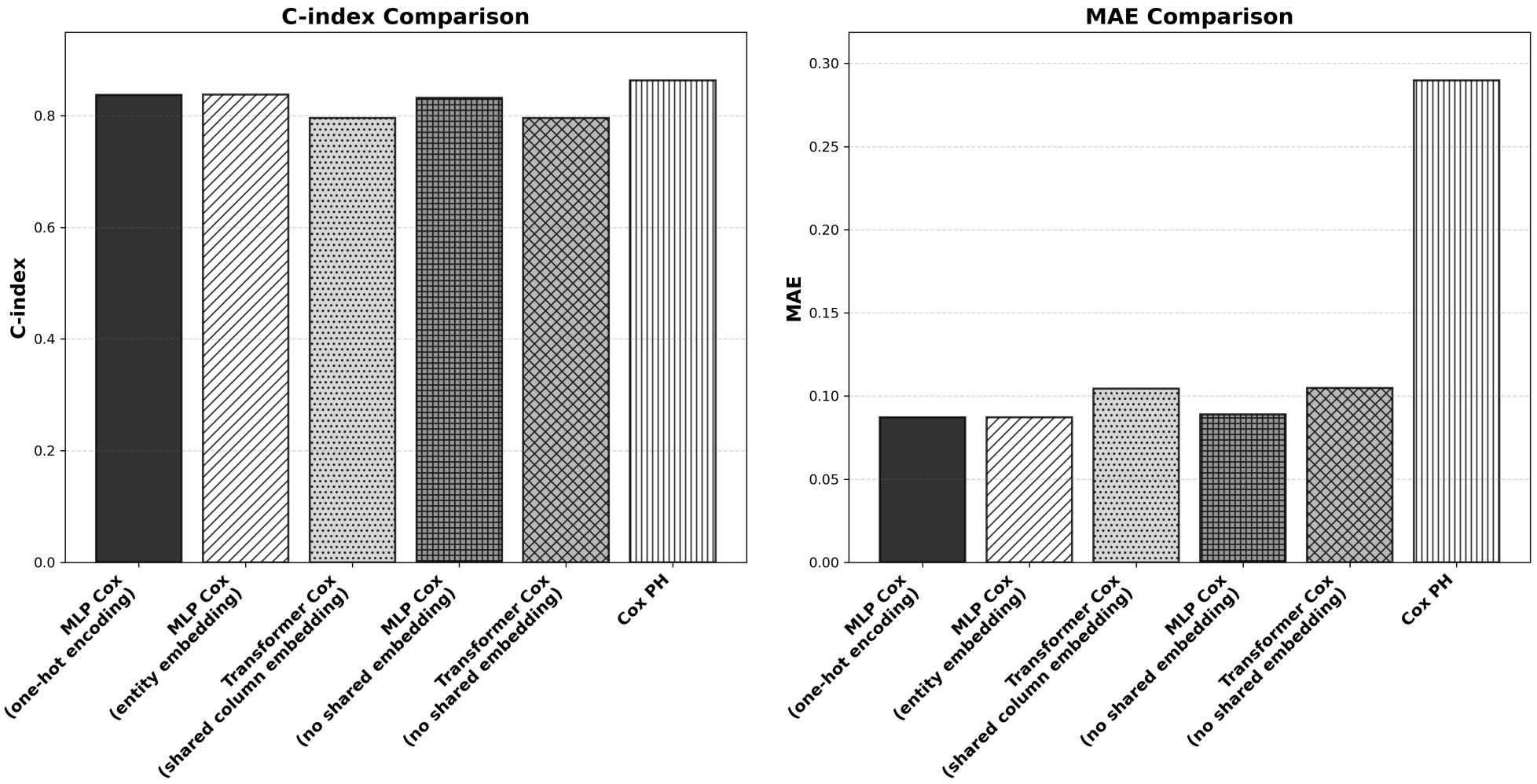



**Figure 8.4: Performance comparison based on simulated linear models.**

# Experiment results: nonlinear scenario

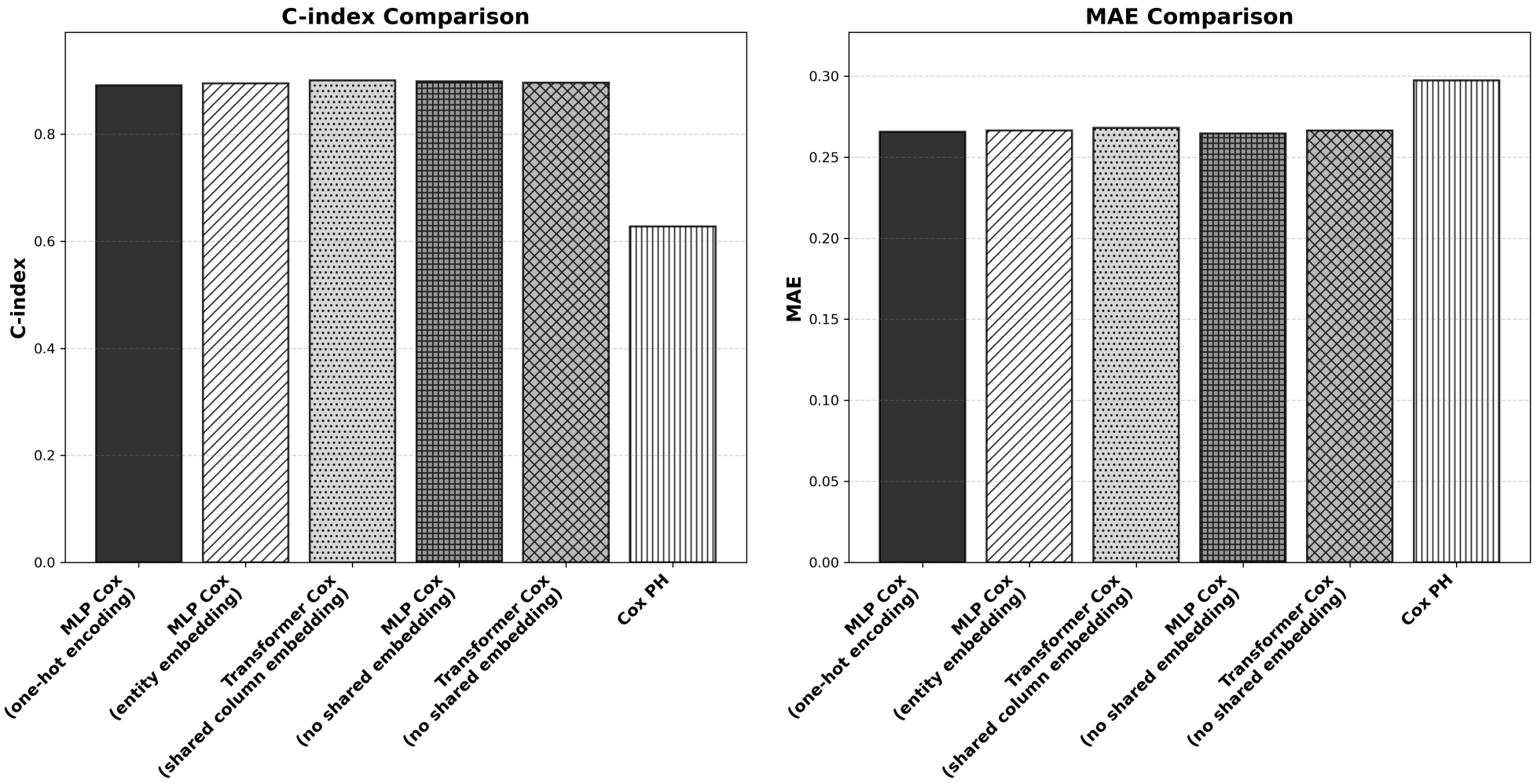



**Figure 8.5: Performance comparison based on simulated nonlinear models.**

# Experiment results: nonlinear with only intersection

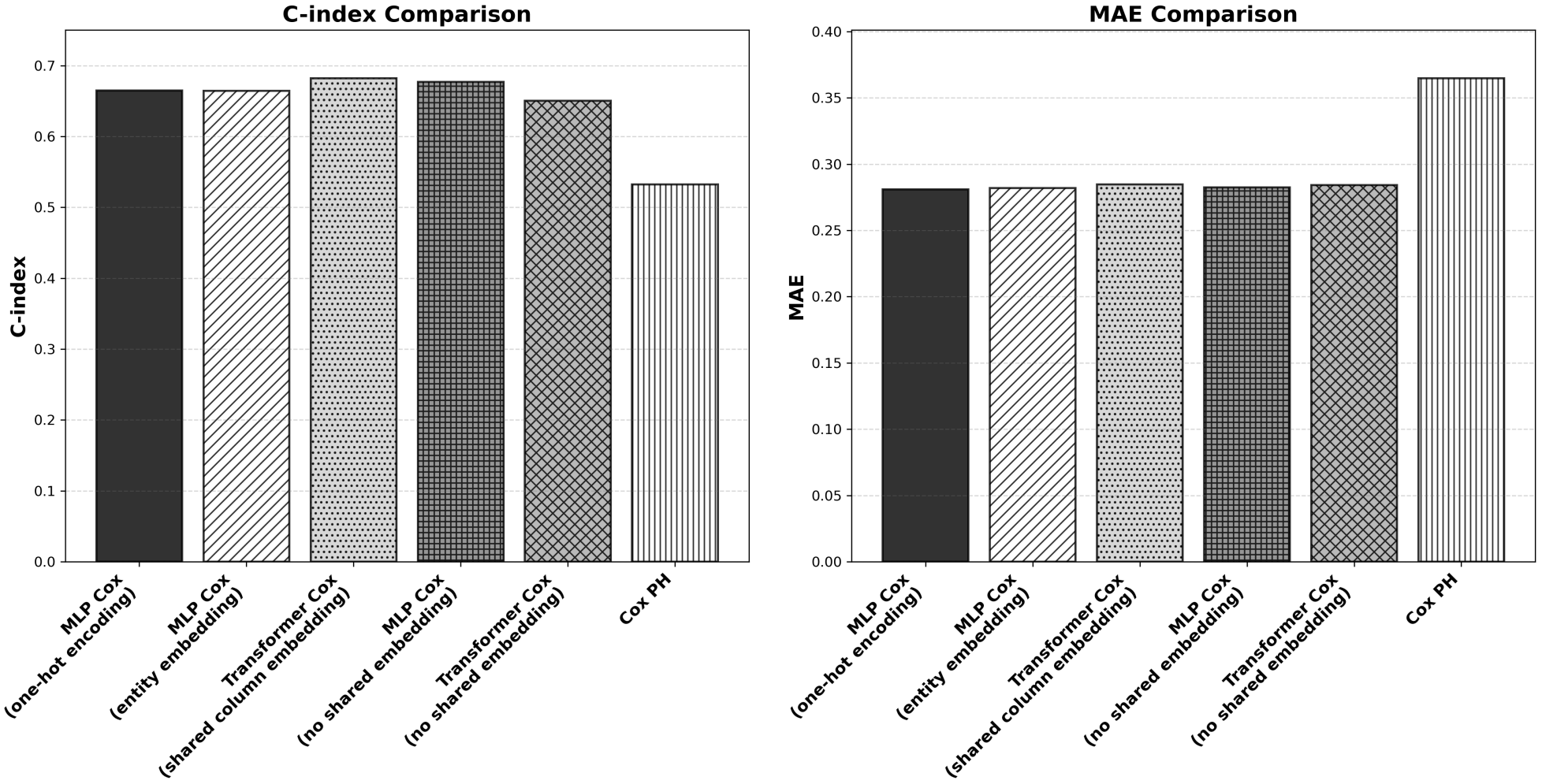
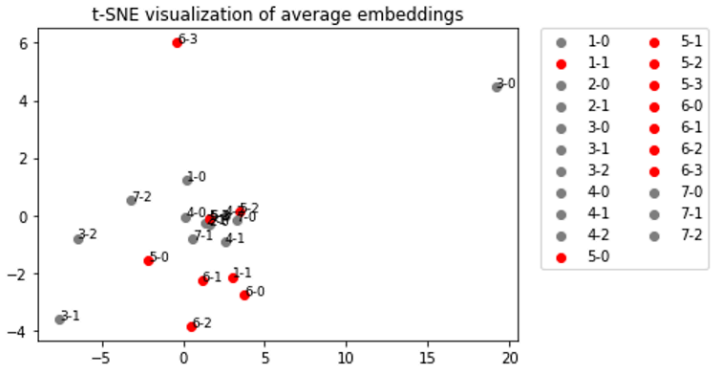


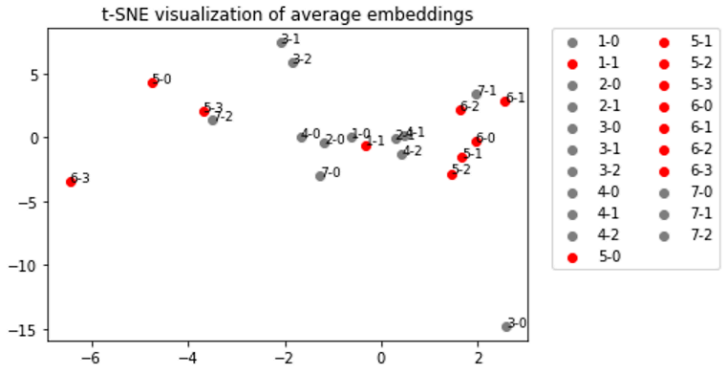

**Figure 8.6: Performance comparison based on simulated nonlinear models with interaction only.**

# Visualization of embedding outputs

We employ t-SNE to visualize the embedding outputs from both Transformer Cox and MLP Cox with entity embedding, focusing on the categories relevant to the risk function (red points).




Transformer Cox embedding: before training




Transformer Cox embedding: after training

> After training, the red points move to the corners of the plot, except Cat1=1 (influenced only by a weak intersection with Cat6=3). This shows Transformer Cox efficiently learns the information from the data; deviations from other points mean distinct information in the model.

# Deep learning approaches for survival modeling

## Part II: Diffusion-based models

# The goal: sample from an unknown distribution

- Idea: we have data $x_0 \sim p_{\text{data}}(x)$ (images, signals, ...). We want a machine that produces new samples from that same distribution.
- Statistician's view: modeling $p_{\text{data}}(x)$ directly in high dimensions is difficult because the density is complex, multimodal, and often intractable. Diffusion strategy: instead of learning the full data distribution directly, we gradually corrupt data with noise, a simple, known process, and train a model to reverse that process step by step.
- Key trick: destroying data with noise is trivial; we instead learn to reverse it.

# The core idea in one picture

- Forward (fixed): gradually add Gaussian noise until data becomes pure static.
- Reverse (learned): train a network to remove a little noise at a time, turning static back into data.
- Analogy. A drop of ink diffusing in water is easy to describe going forward. Diffusion models learn the un-mixing: one tiny, learnable step at a time.
- Both are Markov chains: each step depends only on the previous one.

# Forward process: data → noise *(no learning)*

Add a sliver of Gaussian noise at every step $t = 1, \ldots, T$:

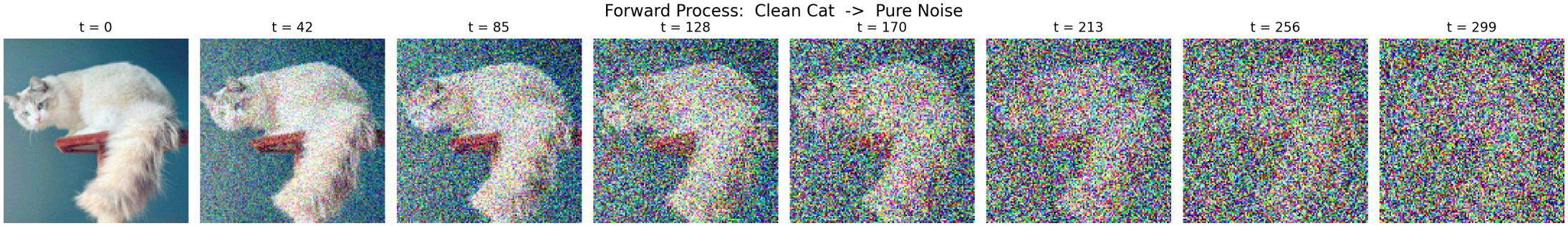
$$q(x_t \mid x_{t-1}) = \mathcal{N}\big(x_t;\ \sqrt{1-\beta_t}\, x_{t-1},\ \beta_t \mathbf{I}\big)$$

A composition of Gaussians is Gaussian, so we can jump to *any* step in closed form (no loop):

$$q(x_t \mid x_0) = \mathcal{N}\big(x_t;\ \sqrt{\bar{\alpha}_t}\, x_0,\ (1-\bar{\alpha}_t)\mathbf{I}\big), \qquad \alpha_t = 1-\beta_t,\ \ \bar{\alpha}_t = \prod_{s \le t} \alpha_s$$

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

# Forward process, visualized




A real cat at increasing noise levels $t$, pure math (the closed form above).

At $t = T$, $x_T \approx \mathcal{N}(0, \mathbf{I})$.

# Reverse process: noise → data *(learned)*

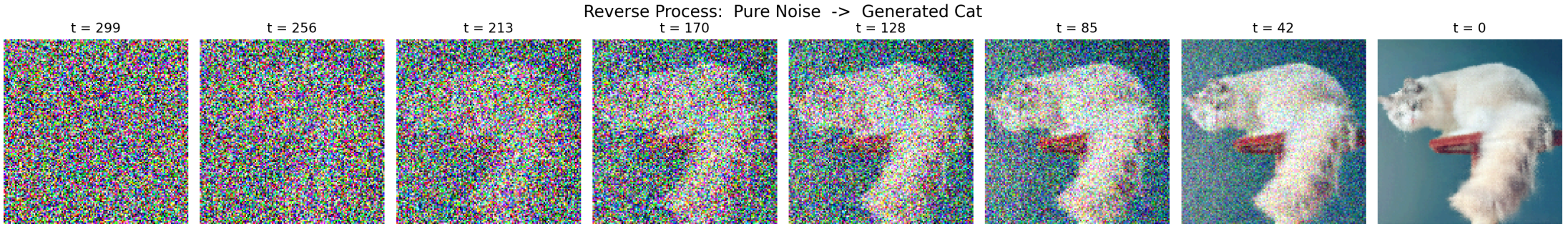
- The true reverse $q(x_{t-1} \mid x_t)$ is unknown, so we approximate it with a Gaussian:

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

- Instead of predicting the mean directly, the network predicts the noise $\epsilon_\theta(x_t, t)$ that was added. The mean then follows in closed form:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\, \epsilon_\theta(x_t, t) \right)$$

# Reverse process, visualized




Start from noise (left), denoise step by step, and the cat re-emerges (right).

# Why this is just statistics

- It is a **latent-variable model** with latents $x_1, \ldots, x_T$. We maximize a **variational lower bound** on the log-likelihood (same ELBO idea as a VAE):

$$\log p_\theta(x_0) \;\geq\; \mathbb{E}_q\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right]$$

- Each term reduces to a **KL divergence** between Gaussians (plus a Gaussian reconstruction term), so the bound has closed forms. After simplification (Ho et al., 2020), training collapses to a plain regression / MSE objective:

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

- No adversarial training, no normalizing flows — just predict noise by least squares.

# Key insight: a chain of small VAEs

- In diffusion, instead of one latent variable, we have many:
  the full latent set is $x_1, x_2, \ldots, x_T$.
- This makes diffusion a **deep latent-variable model**, i.e. a **hierarchical VAE**. Each step behaves like a tiny local VAE.

# Diffusion vs. other generative models

| Approach | Analogy | Result |
|---|---|---|
| **GAN** | Here's a pile of bricks — compete with another architect to build a house. | Sometimes brilliant, often collapses |
| **VAE** | Describe the house with ONE blueprint, then build from that. | House exists, but details are fuzzy |
| **Diffusion** | Start with a finished house, slowly demolish it (forward); then learn to reverse EACH tiny demolition step. | Learn each repair individually → perfect reconstruction |

# SurvDiff: a diffusion model for generating synthetic survival data

The first end-to-end diffusion model that jointly generates covariates, event times, and right-censoring: synthetic patients you can train survival models on.

> Brockschmidt, Schröder & Feuerriegel · LMU Munich & Munich Center for Machine Learning · arXiv:2509.22352 (2025)

# The method: joint diffusion + a Cox-style loss

- Mixed-type, joint generation of $(x^{\text{cont}}, x^{\text{disc}}, e, t)$:
- Continuous (covariates + event time $t$): variance-exploding Gaussian diffusion, network predicts the noise (score matching).
- Discrete (covariates + event indicator $e$): masked diffusion that unmasks categories on the reverse pass.
- Survival-tailored loss: a sparsity-weighted Cox partial likelihood on a risk score $r_i = f_\theta(\mathbf{x})$ from the denoised covariates:

$$\mathcal{L}_{\text{surv}} = -\sum_{i:\, e_i=1} w_i \, \log \frac{\exp(r_i)}{\sum_{j \in \mathcal{R}(t_i)} \exp(r_j)}, \qquad \mathcal{L}_{\text{total}} = \mathcal{L}_{\text{diff}} + \lambda_{\text{surv}} \, \mathcal{L}_{\text{surv}}$$

- Weights $w_i$ downweight rare late events (tiny risk sets) for stable training. With $w_i = 1$ this is exactly the DeepSurv/Cox loss. $\lambda_{\text{surv}}$ is auto-calibrated.

# Why it matters

- Best covariate fidelity and the lowest event-time divergence among baselines (NFlow, TVAE, CTGAN, TabDiff, SurvivalGAN, Ashhad).
- Strongest downstream survival performance (train-on-synthetic, test-on-real), especially under heavy censoring.
- Extends cleanly to differential privacy.
- Takeaway for clinical development: single, stable model that yields shareable synthetic cohorts preserving survival structure: useful for externalcontrol arms, digital twins, and privacy-preserving data sharing.

> Reference: Brockschmidt, Schröder & Feuerriegel (2025), *SurvDiff*, arXiv:2509.22352. Code: github.com/mariebrockschmidt/SurvDiff

# Data considerations for AI-powered clinical development

# Data foundation for evidence-driven clinical development

- **Living evidence foundation:** Continuously integrates clinical trials, publications, internal protocols/data, RWD, guidelines, and regulatory knowledge into an AI-ready therapeutic-area knowledge base.
- **Clinical trial intelligence:** Translates curated evidence into decision support for trial design, feasibility, site/KOL selection, trial emulation, and external control arms.
- **AI-driven documentation:** Uses agentic AI to draft and maintain protocols, SAPs, ICFs, CSRs, and dossiers with traceability, statistical rigor, and compliance readiness.

# Data engineering for AI agents

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

- Clinical data manager: CDISC is the common data language for regulated clinical trials: it makes data standardized, traceable, and acceptable to health authorities..
- Deep-learning engineer: AI agents need data they can query deterministically and join reliably, not reshape and guess.
- 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 for submission; one domain per topic (DM, AE, LB, VS, EX, CM, ...), observation-level rows.
- ADaM: *analysis-ready* datasets (ADSL + BDS), traceable to SDTM.
- Define-XML + Controlled Terminology: machine-readable metadata & vocabularies.
- Key point. CDISC is an interchange / submission standard optimized for human review, traceability, and regulatory trust; *not* a normalized database schema. It is shipped as flat datasets (XPT), not a live relational database.

# A 30-second normalization refresher

- First Normal Form (1NF): Each attribute contains atomic values; no repeating groups, arrays, or multi-valued cells.
- Second Normal Form (2NF): The table is in 1NF, and every non-key attribute depends on the entire candidate key, not just part of a composite key; that is, there are no partial dependencies.
- Third Normal Form (3NF): The table is in 2NF, and non-key attributes depend only on candidate keys, not on other non-key attributes; that is, there are no transitive dependencies.

$$\mathbf{1NF} \supset \mathbf{2NF} \supset \mathbf{3NF} \quad \Rightarrow \quad \textbf{“one fact, in one place.”}$$

> Normalization helps AI agents reason reliably by reducing duplicate facts and exposing clear, joinable relationships.

# CDISC under the normalization lens

- CDISC SDTM/ADaM are generally 1NF-like, but they are not designed to satisfy strict 2NF or 3NF.
- They are regulatory exchange and analysis datasets, optimized for traceability, reviewability, controlled terminology, and analysis readiness; not for fully normalized relational database design.

# 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
  - Limited relational enforcement
  - EAV (Entity–Attribute–Value) structures and reshaping complexity
  - Intentional redundancy
  - Source-vs-derived ambiguity

# Will a relational database work better?

- A normalized relational layer gives AI agents what CDISC does not enforce natively:
  - Primary and foreign keys: deterministic, composable SQL joins
  - Constraints and referential integrity: fewer invalid joins, orphan records, and impossible states
  - One governed source of truth: less redundancy and fewer contradictory facts
  - Clean, documented schema: stronger text-to-SQL reliability and agent reasoning
- Recommended architecture
  - Don't choose *CDISC* or *relational*: layer them.
  - Use each where it is strongest.

# The AI Agent Operating Model

- Humans lead: set objectives, make decisions, review outputs, define priorities, and establish guardrails
- Agents execute: synthesize evidence, extract data, run models, draft documents, and perform QC in parallel
- The loop compounds: reviewed outputs become the next set of tasks, enabling continuous project acceleration
- Impact: faster cycle times without removing human judgment, statistical accountability, or regulatory oversight.

# Operating Loop

- Review: humans inspect drafts, analyses, evidence tables, and QC outputs
- Scope: tasks are queued with clear acceptance criteria and approved data access
- Execute: agents run parallel workstreams across evidence, data, analytics, and documents
- Audit: all actions, inputs, prompts, and outputs are logged for reproducibility
- Bottom line: AI agents extend the clinical development team's execution capacity as digital co-workers, while humans retain scientific judgment, regulatory accountability, and final decision control.

# Thank you!

## Questions?