SkinAdvisor: An AI-Driven Decision Framework for Skin Production Planning via Multi-Modal Transformer Sales Forecasting

Ge Fan[1], **Kai Wang**[1], **Chaoyun Zhang**[1], **Yingjie Li**[2], **Yuze Liu**[3], **Junyang Chen**[4*]

[1]Independent Researcher
[2]Tencent Inc.
[3]Swinburne University of Technology
[4]CSSE, Shenzhen University
ge.fan@outlook.com, wangjinjie722@gmail.com, vyokky@163.com,
wallaceyjli@tencent.com, yuzeliu@swin.edu.au, junyangchen@szu.edu.cn

Abstract

Skins, cosmetic in-game items, are a primary revenue stream for modern online games, yet production decisions (which skins to make, when to release, how to price) remain driven by intuition and retrospective analytics. We present SkinAdvisor, an AI-driven decision framework for skin production planning, powered by SkinCast, a deployed multi-modal Transformer forecasting 180-day per-day sales trajectories for unreleased skins. SkinCast couples four encoders: a pretrained Informer for player-preference time series, a pretrained Player–Hero–Skin graph embedder for relational priors, and two lightweight Transformers for skin attributes and release context, fused through a shared backbone with a local-plus-global day-x positional encoding and decoded by an N-layer Informer in a single non-autoregressive pass. SkinAdvisor plugs these forecasts into a serving pipeline (Select→Fill→Infer→Sales→ROI→Ranking) that ranks candidate skins by forecast-based ROI and supports multi-candidate comparison. The system has been in production for a full release cycle. Offline experiments show SkinCast reduces RMSE by $\sim 20\%$ over the strongest baseline (Informer) and $\sim 60\%$ over classical methods; stress tests on a 16-core CPU server confirm p99 latency under 8ms and peak throughput above 1100 rps. We report two deployment lessons: order-of-magnitude accuracy suffices for production decisions, and pretrained embeddings can leak temporal information, requiring season-based splitting to prevent.

1 Introduction

Cosmetic skins are an important monetization mechanism in online games. Before production begins, producers must choose the target hero, visual tier, release channel, launch date, and production budget. These decisions are difficult because the candidate skin has no historical sales record and its demand depends on both player preferences and the release context. As illustrated in Figure 1, skins take different forms across genres, from lottery-style weapon finishes in the FPS *CS:GO* to directly purchased hero skins in the MOBA *League of Legends* (LoL), yet in both cases skin sales are a primary revenue stream.



Figure 1: Examples of in-game skins, cropped from the authors' gameplay screenshots. Left: a weapon skin from *CS:GO*. Right: a hero skin from *League of Legends*.

Producing a skin is resource-intensive: concept design, 3D modeling, animation, VFX, sound, and QA all require substantial investment, and sustained operations demand a steady cadence of new releases. In current practice, decisions about *which* skins to produce, *when* to release them, and *how* to price them are driven largely by producer intuition and retrospective analytics. This workflow fails to capture rapidly shifting player preferences, underestimates the interaction between release timing and preference trends, and often leads to missed opportunities or over-production of low-return items.

Predictive modeling offers a data-driven alternative. Concretely, given a candidate skin that has not yet been produced and all multi-source signals available at a prediction time t_p roughly one to two seasons before its planned launch date t_0, the task is to forecast its daily sales trajectory over the first 180 days after t_0. With such a forecast in hand, producers can rank candidates by forecast-based return on investment (ROI), allocate art and engineering capacity to the highest-return concepts, and stress-test alternative release dates against the seasonal demand curve. Deploying such a system in a live studio, however, raises challenges that generic time-series or tabular forecasters do not address:

- **Heterogeneous, multi-source inputs.** Game-level time series (daily active users, match counts, hero usage), the player–hero–skin interaction graph over millions of enti-

*Corresponding author.
Copyright © 2027, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

ties, and sparse skin-attribute and release-context fields (series, tier, price, hero win rate, event tags) arrive in different formats and must be ingested jointly rather than flattened into one tabular snapshot.

- **Seasonal and calendar effects.** Skin sales show strong periodicity from weekly play cycles, seasonal holidays, and summer / winter breaks; a model that only sees "days since launch" mis-estimates revenue at holiday peaks and off-season troughs, distorting the ROI ranking producers rely on.
- **Cold-start on unreleased skins.** The target item has no interaction history by construction (Volkovs, Yu, and Poutanen 2017), so the model must generalize from relational priors of its hero and attribute-level patterns of previously released skins.
- **Long-horizon forecasting at production scale.** Planning requires the full 180-day per-day trajectory, regenerated for thousands of candidate skins weekly under tight serving latency budgets.

We address these challenges with SkinCast, a multi-modal (Baltrušaitis, Ahuja, and Morency 2019) Transformer forecaster, together with SkinAdvisor, an ROI-driven decision framework built on top of it. SkinCast answers the input heterogeneity with four specialized encoders: a pretrained Informer for macro time series (Zhou et al. 2021), a pretrained Player–Hero–Skin graph embedder for relational priors (Fan et al. 2022a) that also anchors cold-start generalization to the target skin's hero, and two lightweight Transformer encoders for skin attributes and release context; a shared backbone fuses them, and a local-plus-global day-x positional encoding carries the calendar signal into a single-pass non-autoregressive decoder that emits the full 180-day trajectory in one forward call. SkinAdvisor plugs these forecasts into a Select→Fill→Infer→Sales→ROI→Ranking serving pipeline that ranks candidate skins by forecast-based ROI. Figure 2 gives a joint overview: the top panel shows SkinCast's encoder–decoder architecture, and the bottom panel shows how SkinAdvisor wraps it into a producer-facing serving pipeline.

The system has been deployed in production and is used by producers to prioritize the skin roadmap. In offline experiments, SkinCast reduces RMSE by $\sim 20\%$ over the strongest baseline (Informer) and $\sim 60\%$ over classical methods; stress tests on a 16-core CPU server show p99 latency under $8\mathrm{ms}$ at typical concurrency. We also report two deployment lessons: the sufficiency of order-of-magnitude accuracy for production decisions, and a temporal leakage via pretrained embeddings that we diagnosed and fixed with season-based splitting.

Our contributions are:

- We formulate skin sales forecasting as a multi-modal, long-horizon, per-day regression problem, and identify four concrete production challenges (heterogeneous inputs, calendar effects, cold start, and serving scale) that shape the design space.
- We propose SkinCast, a deployed multi-modal Transformer forecaster that unifies a pretrained Informer time-series encoder, a pretrained Player–Hero–Skin graph encoder, and two lightweight Transformer encoders for static attributes and release context, with a local-plus-global day-x positional encoding and a direct (non-autoregressive) Informer decoder for 180-day trajectories. Ablations identify the main contributing modules, with the time-series encoder being the most critical ($+22\%$ RMSE degradation when replaced by an equal-parameter MLP).
- We design and operate SkinAdvisor, an end-to-end ROI-driven decision framework that couples SkinCast with a cost-allocation side path and a multi-candidate ranking service, turning it from a prediction tool into a production planning tool. The architecture could potentially be adapted to other content production scenarios (battle passes, seasonal events, fashion drops, collectible launches).
- We report offline accuracy, ablations, stress-test deployment metrics, and two concrete lessons learned from a full release cycle in a live game studio: (a) order-of-magnitude correctness suffices for early-stage production decisions, and (b) pretrained embeddings can leak temporal information indirectly, requiring season-based splitting to prevent.

2 Data and Feature System

SkinCast consumes five feature families harvested from production logs: four heterogeneous input families, one per encoder module of Section 3.1, plus a target-day-indexed calendar family that feeds the day-x positional encoding of Section 3.2. Below we state the two design constraints that shape the feature system; the inputs to each module are described in Section 3.1.

Prediction-time freezing. Producers make the production decision one to two seasons, roughly 90 to 180 days, before a skin's launch, so we define a *prediction time* t_p with $t_0 - t_p \approx$ 90–180 days (strictly $t_p < t_0$). Every input, the macro time-series window, the graph snapshot, and the release-context fields, is frozen as of t_p so that no post-launch information can leak into training or serving; the season-aligned cutoff used to enforce this in training is described in Section 6.

Family-to-module mapping. The macro time-series family feeds the pretrained Informer encoder (Module 1); the player–hero–skin interaction graph feeds the pretrained PPPNE encoder (Module 2) and is what resolves the cold-start challenge of Section 1; skin attributes feed the TF1 encoder (Module 3); release context feeds the TF2 encoder (Module 4); and the calendar family is *not* consumed by any encoder but added to the pooled encoder summary as $\mathbf{p}_x$ before the decoder stack.

3 Proposed Method

Given a candidate skin s with release date t_0, SkinCast predicts the per-day sales trajectory $\hat{\mathbf{y}}_{1:180} = (\hat{y}_1, \ldots, \hat{y}_{180})$ over the first 180 days after launch. The overall architecture (Figure 2) is a multi-modal encoder–decoder:

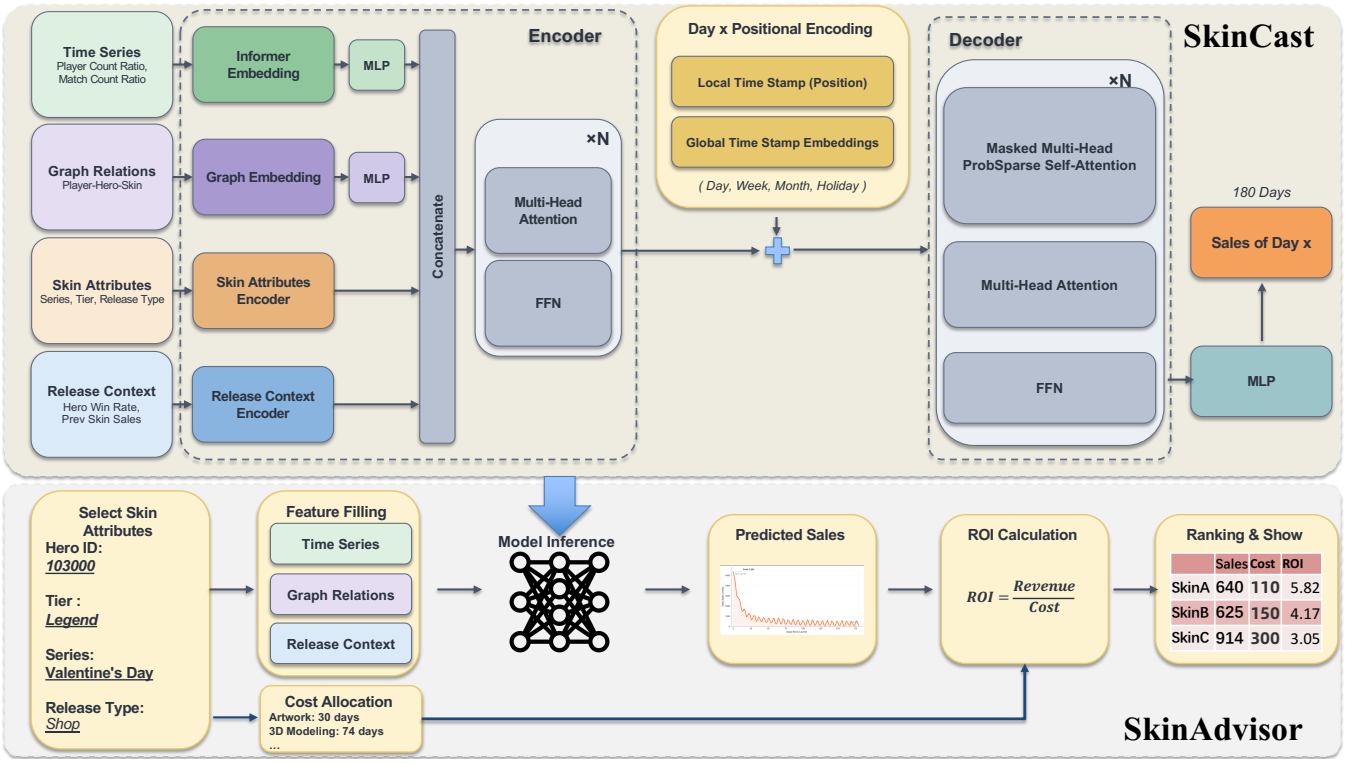


Figure 2: Overview of SkinCast (top: multi-modal encoder–decoder architecture) and SkinAdvisor (bottom: producer-facing serving pipeline from core attributes to an ROI-ranked shortlist).

$$\mathbf{H}_{\text{enc}} = \text{Backbone}^{N}\big([\mathbf{H}_{\text{ts}}; \mathbf{H}_{\text{gr}}; \mathbf{H}_{\text{sa}}; \mathbf{H}_{\text{rc}}]\big), \quad (1)$$

The concatenation above stacks four module vectors along the token axis, so $\mathbf{H}_{\text{enc}} \in \mathbb{R}^{4\times d}$. We define a global summary

$$\mathbf{g} = \text{Pool}_{\text{fusion}}(\mathbf{H}_{\text{enc}}) \in \mathbb{R}^{d}. \quad (2)$$

The full encoder output serves as cross-attention memory; its pooled summary initializes the daily decoder tokens. The head predicts $\hat{z}_x$ on the log-sales scale, with $z_x = \log(1 + y_x)$. Let $\mathbf{p}_x \in \mathbb{R}^d$ be the target-day encoding defined in Section 3.2. The decoder input is explicitly constructed as

$$\mathbf{Z}_0[x] = \mathbf{g} + \mathbf{p}_x, \quad x = 1, \ldots, 180, \quad (3)$$

so $\mathbf{Z}_0 \in \mathbb{R}^{180\times d}$. Throughout, $[x]$ denotes the vector at sequence position x. The overall log-sales prediction is

$$\hat{\mathbf{z}}_{1:180} = \text{MLP}_{\text{head}}\left(\text{Dec}^{N}(\mathbf{Z}_0; \mathbf{H}_{\text{enc}})\right). \quad (4)$$

The head is applied independently at each target-day position; the decoder's second argument supplies its cross-attention memory. Here Backbone^N and Dec^N are stacks of N Transformer encoder blocks (Vaswani et al. 2017) and N Informer decoder layers, and $\mathbf{p}_x$ is the day-x positional encoding defined below.

3.1 Multi-Modal Encoder

Four specialized encoders map heterogeneous production inputs to a shared d-dimensional space:

$$\mathbf{H}_{\text{ts}} = \text{MLP}_{\text{ts}}\big(\text{Pool}(\text{InformerEnc}_{\theta^*}(\mathbf{X}_{\text{ts}}))\big), \quad (5)$$

$$\mathbf{H}_{\text{gr}} = \text{MLP}_{\text{gr}}\big(\text{PPPNE}(\mathcal{G}_{\text{PHS}})[h(s)]\big), \quad (6)$$

$$\mathbf{H}_{\text{sa}} = \text{Pool}\big(\text{TF1}(\text{Emb}(\mathbf{f}^{\text{sa}}))\big), \quad (7)$$

$$\mathbf{H}_{\text{rc}} = \text{Pool}\big(\text{TF2}(\text{Emb}(\mathbf{f}^{\text{rc}}))\big), \quad (8)$$

where θ^* is a *frozen* Informer encoder pretrained on macro player-preference series, $\mathcal{G}_{\text{PHS}}$ is the Player–Hero–Skin interaction graph, $h(s)$ is the hero associated with skin s, and TF1 / TF2 are two parameter-separate instances of the same one-layer Transformer architecture (MHA+FFN with residual layer normalization (Ba, Kiros, and Hinton 2016)).

Time Series (Eq. 5). Encodes macro-temporal signals that capture how player preferences toward heroes and skins evolve, including DAU ratio, match-count ratio, hero-usage distribution, and skin-usage frequency, over a sliding window of the K days ending at the prediction time $t_p < t_0$. Freezing θ^* and training only MLP_{ts} reuses a player-preference embedder already maintained by the operations team, avoiding costly joint pretraining on the small sales dataset.

Graph Relations (Eq. 6). Injects relational priors mined from real player–hero–skin interactions via network embedding (Grover and Leskovec 2016; He et al. 2020; Zhang et al. 2021). Because the embedding is pretrained on the full graph,

a newly released skin inherits signal from its hero's neighborhood; this is the primary mechanism by which SkinCast addresses the cold-start challenge of Section 1, even with zero direct behavioral trace of the target skin, the hero embedding carries the size and purchasing propensity of the fan base it will be exposed to.

Skin Attributes (Eq. 7) and Release Context (Eq. 8). Both static-feature encoders share the same $\mathrm{Emb}{+}\mathrm{TF}{+}\mathrm{Pool}$ template (with TF1 and TF2 as separate parameter instances), treating each categorical / numerical field as a token and letting self-attention weigh cross-field interactions (Fan et al. 2022b) (e.g., tier $\times$ series, or price $\times$ hero popularity). One Transformer layer is sufficient because the field set is small ($F_{\mathrm{sa}}, F_{\mathrm{rc}} \sim 10$) and fully observed at release time.

Backbone. The four module tokens are concatenated and fed through N Transformer encoder blocks. Because they live in a shared d-dimensional space, the backbone performs cross-modal attention directly: the time-series token attends to the graph token, attribute tokens attend to release-context tokens, and so on. This is where higher-order interactions across heterogeneous sources are extracted, in contrast to late-fusion baselines that only combine module-level scalars.

3.2 Day-x Positional Encoding

$$\mathbf{p}_x \;=\; \underbrace{\mathbf{E}_{\mathrm{local}}(\mathrm{rel_day}(x))}_{\text{local}} \;\oplus\; \underbrace{\textstyle\sum_{c\in\mathcal{C}} \mathbf{E}_c(c(x))}_{\text{global}}, \qquad (9)$$

All embedding terms lie in $\mathbb{R}^d$, and $\oplus$ denotes element-wise addition. Set $\tau_x = t_0 + (x-1)$ days, with $x = 1$ denoting launch day. Here $\mathcal{C} = \{\text{day, wk, mo, hol}\}$ indexes day of month, day of week, month, and holiday status, respectively, evaluated at τ_x using calendar information known at prediction time. The local term orders the horizon since launch and captures the canonical revenue decay; the global term injects real-calendar periodicity (weekly play cycles, seasonal holidays) that a pure relative index cannot express. The encoding is added to the pooled summary $\mathbf{g}$ once to initialize the decoder tokens.

3.3 Informer Decoder and Training Objective

Starting from $\mathbf{Z}_0$ in Eq. 3, the decoder produces $\mathbf{Z}_N \in \mathbb{R}^{180\times d}$:

$$\mathbf{Z}_N = \mathrm{Dec}^N(\mathbf{Z}_0; \mathbf{H}_{\mathrm{enc}}), \qquad \hat{z}_x = \mathrm{MLP}_{\mathrm{head}}(\mathbf{Z}_N[x]), \qquad (10)$$

$$\mathcal{L} \;=\; \frac{1}{180}\sum_{x=1}^{180} \big(\hat{z}_x - \log(1+y_x)\big)^2. \qquad (11)$$

Here $y_x \geq 0$ is raw daily unit sales. Equation 11 is the loss for one skin; training averages it over skins in each batch. Equation (3) broadcasts the pooled encoder summary $\mathbf{g}$ to every target day and adds that day's positional embedding $\mathbf{p}_x$, yielding a 180-length decoder input sequence $\mathbf{Z}_0$ in which each position carries both the global multi-modal context and its own calendar / horizon position. Each decoder layer then applies, in order, a masked ProbSparse self-attention (causal over the 180 days), a cross-attention over $\mathbf{H}_{\mathrm{enc}}$, and an FFN; the ProbSparse operator reduces long-horizon cost from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log L)$.

Direct (non-autoregressive) decoding. SkinCast emits all 180 days in a single decoder pass rather than feeding predictions back as inputs. This choice was driven by an early autoregressive prototype that accumulated error over the horizon and complicated serving.

Target transformation. Raw daily sales are heavy-tailed. We apply $y \mapsto \log(1+y)$ before Eq. (11). Section 5 reports all metrics on this transformed scale; serving converts predictions back to unit sales before aggregation:

$$\hat{y}_x = \max\{0, \exp(\hat{z}_x) - 1\}. \qquad (12)$$

These are point forecasts, not necessarily conditional mean sales. Monotonicity preserves individual scalar orderings, but does not guarantee cumulative-sales or ROI ranking quality.

4 The SkinAdvisor Decision Framework

SkinAdvisor turns producer-specified skin attributes into a ranked shortlist with 180-day sales curves and forecast-based ROI through the pipeline in Figure 2.

4.1 End-to-End Pipeline

SkinAdvisor executes six stages per producer request; the serving half of Figure 2 (bottom panel) illustrates the full flow.

1. Select. The producer opens the SkinAdvisor web frontend and specifies the *core* attributes of the candidate skin: the associated hero, the product tier (e.g., legendary / epic / rare), the release channel (in-game store, battle pass, seasonal event), and the planned launch date. All four fields support multi-select so that the producer can pose comparison queries such as "same skin concept across three candidate heroes" or "same hero across two release channels".

2. Fill. The backend feature-completion service expands the producer-specified core attributes into the full input bundle consumed by SkinCast. Skin series, theme, and effects flag are inferred from a rule table indexed by (tier, hero, channel) and maintained by the art team; price is looked up in a pricing-policy table indexed by (tier, channel); hero win rate is fetched from the live game-stats service; previous-skin sales of the same hero come from the sales warehouse; the IP-collaboration flag and event tag come from the marketing calendar; the hero's node embedding is looked up in the representation store that serves the pretrained PPPNE table; and the macro time-series window is pulled from the operations dashboard. This stage is the bridge between how a producer thinks ("a legendary summer-event skin for hero X") and what SkinCast consumes (a fully-specified four-module input).

3. Infer. The filled feature bundle is forwarded to the deployed SkinCast inference service, which emits the 180-day per-day sales trajectory $\hat{\mathbf{y}}_{1:180}$. When the producer selected N candidate combinations in stage 1, the backend expands the Cartesian product and batches all N candidates into a single forward pass (batch dimension $= N$) to amortize inference cost.

4. Sales. The trajectory is aggregated to $\text{total_sales} = \sum_{x=1}^{180} \hat{y}_x$. The system also computes 30-, 90-, and 180-day subtotals so that producers can compare short-term spikes against long-tail revenue.

5. CostAlloc → ROI. In parallel with the inference chain, a cost-allocation side path looks up the expected person-days for the given tier in a rule table maintained by the production-management team (e.g., legendary $= 120$ person-days, epic $= 60$, rare $= 20$) and converts person-days to person-hours before multiplying by the studio's blended hourly rate to obtain Cost. ROI is then computed as

$$\text{ROI} = \frac{\text{total_sales} \times \text{unit_price}}{\text{Cost}}. \tag{13}$$

Here Cost is strictly positive and unit_price denotes revenue per unit under a fixed-price assumption. Throughout this paper, ROI is defined as predicted revenue divided by production cost. The ROI formulation (Eq. 13) decouples cost estimation from inference: CostAlloc is deliberately a *side path* that does not gate inference and can be revised (e.g., when the studio updates its rate card) without retraining or redeploying SkinCast.

6. Ranking. The N candidate ROI values are sorted descending, and the frontend renders the top-K candidates as overlaid 180-day sales curves (color-coded by candidate) alongside a ranked table showing hero, tier, channel, launch date, total_sales, Cost, and ROI. Producers can apply filters (e.g., "only candidates with $\text{ROI} > 3$") and export the shortlist to the roadmap planning system.

4.2 Multi-Candidate Comparison

Multi-select produces $N = H \times T \times C \times L$ candidate combinations of heroes, tiers, channels, and launch dates, capped at $N_{\max}$ to bound serving cost. Each candidate is evaluated independently within one batched forward pass, and the top-K by ROI (default $K = 20$) are returned for comparison.

4.3 Deployment Topology

SkinCast is served as a stateless CPU inference service behind a load balancer; the feature-fill, cost-allocation, and ranking services are CPU-only microservices; the frontend is a single-page web app. The entire pipeline runs on CPU only, with no GPU required at either training or serving time. Measured latency and throughput under stress-test loads are reported in Section 5.

5 Experiments

Since the quality of SkinAdvisor's ROI rankings depends directly on forecast accuracy, we evaluate SkinCast offline against seven baselines, ablate its components, and stress-test the serving pipeline.

5.1 Experimental Setup

Dataset. Our skin sales data comes from a live online game with tens of millions of Daily Active Users (DAU), spanning over three years of continuous operation. To prevent temporal leakage, we use the three most recent skin *releases* (each release batches multiple skins launched on the same date; all have been live for more than 180 days) as the test set, together accounting for roughly 5% of the total skin corpus; all earlier releases form the training set. For each test release, all model inputs are frozen as of its prediction time t_p (Section 2): the macro time-series window ends at t_p, both pretrained components are loaded from historical versions available at t_p and trained only on seasons completed by t_p, and historical Release Context statistics come from the latest such season. Planned price and event fields reflect information known at t_p. This matches the online serving condition, where a producer queries the system one to two seasons before release. The prediction target is the $\log(1 + y)$ transformed daily sales over the first 180 days post-launch, consistent with the training objective in Eq. (11).

Baselines. Since the target skin has no sales history of its own, all baselines operate in the same pre-release setting as SkinCast: they forecast from a *proxy* signal constructed from previously released skins, never from the target skin's own post-launch sales. We compare against seven baselines covering classical statistical methods, deep time-series models, and tree-based approaches. **ARIMA** (Box and Jenkins 1970) and **Prophet** (Taylor and Letham 2018) are each fitted on the aggregated historical sales sequence of same-hero / same-series skins (Prophet additionally with holiday and seasonality regressors) and extrapolated to the target's 180-day horizon. **MLP** (Rumelhart, Hinton, and Williams 1986) operates over concatenated flat features without sequence modeling but still emits the full 180-day trajectory via a multi-output regression head; **GBDT** (Friedman 2001) consumes the same flat features and fits one gradient-boosted regressor per target day to produce the same 180-day trajectory; a two-layer **LSTM** (Hochreiter and Schmidhuber 1997) encodes macro time-series concatenated with static features and feeds a dense head; **DeepAR** (Salinas et al. 2020) is an autoregressive RNN forecaster trained globally across all skins; and **Informer** (Zhou et al. 2021) is the ProbSparse Transformer forecaster trained end-to-end on the macro time-series features only (without graph, skin attributes, or release context). All seven baselines therefore output the same per-day 180-step sequence as SkinCast and are evaluated under identical metrics.

Metrics. We report Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) on the $\log(1 + y)$ scale, computed per day and averaged over the full 180-day horizon. Lower is better for both. We additionally report the relative improvement of SkinCast over each baseline: $\text{Impr.} = (\text{Baseline} - \text{SkinCast})/\text{Baseline} \times 100\%$.

5.2 Main Results

Table 1 reports the results. SkinCast attains the lowest RMSE and MAE among all compared models, with a relative improvement of $\sim 20\%$ over the strongest baseline (Informer) and $\sim 60\%$ over classical methods (ARIMA). Deep sequence models (LSTM, DeepAR, Informer) achieve lower error than flat models (MLP, GBDT) in this evaluation, indicating that temporal dynamics in macro player-preference signals carry predictive value. The gap between SkinCast and the plain

Table 1: Offline comparison on the $\log(1+y)$ scale.

Model	RMSE	MAE	RMSE Impr. (%)	MAE Impr. (%)
ARIMA	0.5185	0.4075	60.15	60.29
MLP	0.4594	0.3643	55.02	55.58
GBDT	0.4430	0.3472	53.37	53.39
Prophet	0.4315	0.3396	52.12	52.35
LSTM	0.3495	0.2774	40.88	41.66
DeepAR	0.2974	0.2331	30.53	30.59
Informer	0.2572	0.2018	19.67	19.79
SkinCast	**0.2066**	**0.1618**	—	—

Informer baseline isolates the benefit of multi-modal fusion: the Informer baseline sees the same macro time series but lacks graph relations, skin attributes, release context, and the local-plus-global day-x positional encoding. Classical statistical models (ARIMA, Prophet) perform worst because they compress the same-hero / same-series history into a single aggregated proxy series and extrapolate it, so they can model neither the multi-modal interactions nor the per-day calendar effects that SkinCast captures. Given the limited test set (three release batches), these results are best read as descriptive rather than statistically conclusive. Because SkinAdvisor ultimately consumes the forecast as a *ranking* signal, we additionally evaluate per-day ranking quality: on each target day the test SKUs are ranked by predicted same-day sales and scored by NDCG@k, averaged over the 180 days; SkinCast attains the highest NDCG at every cutoff ($k = 1, 3, 5, 13$), confirming that its point-accuracy advantage carries over to per-day sales ranking; cumulative revenue and ROI ranking require separate evaluation.

Horizon-wise error behavior. Table 1 aggregates error over the full 180-day horizon; Figure 3 breaks it down by target day for SkinCast and the strongest autoregressive baseline (DeepAR). Two patterns stand out. First, both models show lower RMSE in the first weeks after launch and rising error toward the tail of the horizon. We attribute this to the shape of skin demand: sales peak sharply in the first days post-launch and decay into a long tail, so the regression loss is dominated by the peak regime and the model fits the peak scale more faithfully than the low-volume tail. Second, both curves exhibit local oscillations that line up across the two models, which we attribute to skin-specific operational events and calendar holidays that perturb individual release trajectories. SkinCast tracks these perturbations with visibly smaller amplitude and a narrower spread across test skins than DeepAR, consistent with its explicit calendar encoding: the global (day-of-week / month / holiday) component of the day-x positional encoding lets SkinCast absorb holiday- and event-driven demand shifts rather than mistaking them for noise.

5.3 Ablation Study

We ablate one module at a time from the full SkinCast and re-evaluate on the same test set (Table 2). In every "w/o X" variant, the module X is replaced by a trainable MLP of comparable parameter count while retaining its input signal.

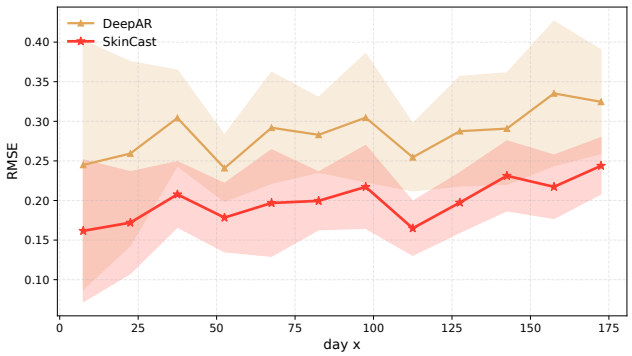


Figure 3: Per-day RMSE over the 180-day horizon (15-day bins) for SkinCast and DeepAR; the shaded band shows the spread across test skins.

Table 2: Ablation. "w/o X" replaces module X with a trainable MLP of comparable parameter count.

Variant	RMSE	MAE	ΔRMSE (%)
SkinCast (full)	**0.2066**	**0.1618**	—
w/o Time Series	0.2520	0.1987	+21.98
w/o Transformer Decoder	0.2350	0.1867	+13.75
w/o Graph	0.2230	0.1727	+7.94
w/o Transformer Encoder	0.2212	0.1749	+7.07
w/o TF2 (Rel. Context)	0.2108	0.1691	+2.03
w/o TF1 (Skin Attr.)	0.2096	0.1685	+1.45

This isolates the contribution of each module's *architectural choice* (pretrained Informer / pretrained PPPNE / Transformer encoder / Transformer decoder / one-layer Transformer for static fields) rather than the contribution of the raw input signal itself.

Replacing the pretrained Informer time-series encoder with an MLP produced the largest RMSE increase (+21.98%), followed by the Transformer decoder (+13.75%). Replacing the graph and backbone encoder modules increased RMSE by 7.94% and 7.07%, respectively. The changes associated with the two static-feature encoders were much smaller: replacing TF1 (Skin Attributes) and TF2 (Release Context) with MLPs increased RMSE by only 1.45% and 2.03%. This suggests that for small, fully observed static field sets ($F_{\text{sa}}, F_{\text{rc}} \sim 10$), cross-field self-attention adds little over an MLP, which is consistent with our choice of a single-layer Transformer for these two modules; deeper variants were not evaluated. The sequential, relational, and long-horizon-decoding pathways, by contrast, benefit clearly from Transformer-based components. Given the small test set, these results should be interpreted as descriptive rather than statistically conclusive.

5.4 Deployment Metrics

SkinAdvisor serves an internal team of product planners; peak concurrency is therefore low relative to consumer-facing services. We stress-test the SkinCast inference service to demonstrate that latency remains well within the producer-facing UI budget (p95 $<$ 3s) even under artificially high load.

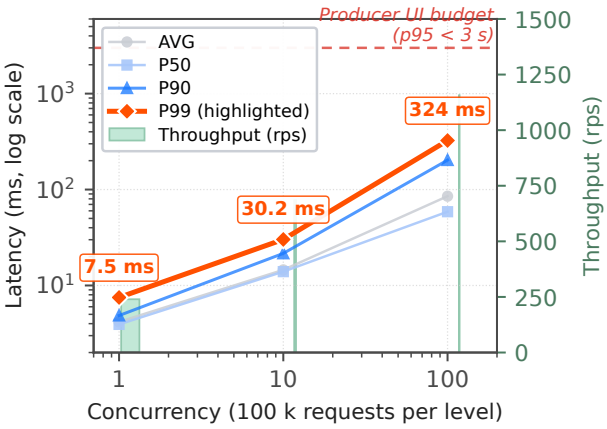


Figure 4: Inference latency (left axis) and throughput (right axis) versus concurrency on a single 16-core CPU server, 100k requests per level.

Figure 4 shows that at concurrency 1 (the typical producer workload), model inference completes in $\sim 4\text{ms}$ per request with p99 $< 8\text{ms}$. Even at concurrency 100, far beyond realistic internal usage, p99 remains under 325ms, leaving ample headroom for the feature-fill, cost-allocation, and ranking stages in the full pipeline. Peak throughput exceeds 1100 requests/second at concurrency 100, confirming that a single 16-core CPU server comfortably supports the entire producer team without queuing. The entire inference pipeline runs on CPU only, with no GPU required at serving time, because the model is compact (the Informer encoder operates over a short K-day window, and the decoder generates 180 tokens in a single pass) and the PPPNE embeddings are precomputed and cached. This eliminates GPU provisioning costs and simplifies operations, making the system practical for studios without dedicated ML infrastructure.

6 Lessons Learned

Two lessons from the development and deployment of SkinAdvisor shaped the final system and are likely applicable to other forecasting-driven decision tools.

6.1 "Good Enough" Beats "Precise"

Through iterative discussions with producers and product planners, we learned that *order-of-magnitude correctness* matters far more than point-estimate precision in the early stages of skin production planning. Producers do not ask "exactly how many units will this skin sell on day 47?"; they ask qualitative questions such as "if we sell the same tier of skin for the same hero via a lottery system instead of direct store purchase, how much does the expected revenue increase, one order of magnitude or two?"

This insight reshaped our evaluation criteria and UI design. Rather than optimizing solely for per-day MAE, we prioritize *rank-order consistency* between candidate skins (i.e., does the system correctly identify that option A outsells option B by a factor of ~ 10?). The multi-candidate comparison mode of SkinAdvisor (Section 4) was a direct consequence: producers want to see *relative* curves side by side and make a directional call, not to read a single number with a confidence interval. Formal ranking metrics (Spearman correlation, pairwise accuracy) are planned for future evaluation as more releases accumulate.

This experience highlights the importance of matching forecast precision to the granularity of the production decision.

6.2 Temporal Leakage via Pretrained Embeddings

In early versions of SkinCast, we observed suspiciously low training loss and strong in-sample accuracy, yet poor generalization to the held-out test skins. Root-cause analysis revealed a subtle form of **temporal leakage through the pretrained Informer encoder**.

The problem. The Informer time-series encoder was pretrained on macro player-preference data that extended *beyond* each skin's launch date. From a data-construction perspective there was no direct leakage: the pretraining target was next-step player-preference metrics, not skin sales. However, the pretrained encoder had implicitly absorbed information about post-launch demand surges (because a popular skin launch causes DAU spikes and match-count changes that the encoder saw during pretraining). When this encoder was frozen and used as a feature extractor inside SkinCast, the downstream model could exploit these leaked signals to "predict" sales of skins that the Informer had already implicitly observed, producing overfitting without any explicit label contamination.

The fix: season-based temporal splitting. We leveraged domain knowledge of the game's *season* structure (each season lasts $\sim$2–3 months) to impose strict temporal boundaries on *both* pretrained components. For each skin, let s_p denote the latest season completed by its prediction time t_p, rather than the season immediately preceding launch:

- The Informer encoder is pretrained on player-preference sequences from seasons $\leq s_p - 1$, with the pretraining target (label) drawn from season s_p; both inputs and labels are available by t_p.
- The PPPNE graph embedding is likewise trained only on player–hero–skin interaction data from seasons $\leq s_p$.
- For each test release, evaluation loads the historical pretrained model and embedding versions available at t_p, with the above season cutoff; no later-trained versions are used, and no input or supervision target from after t_p enters either pretraining pipeline.

Because the number of seasons is finite and well-defined, this scheme is both easy to audit and efficient: each season provides a substantial volume of time-series and graph data, so both pretrained components still receive sufficient signal without ever "seeing the future."

Impact. The season-based split removes the leakage path: no pretrained component can observe interactions or metrics

from after the prediction time t_p, so the reported test-set numbers reflect a genuine pre-release forecasting setting rather than an inflated in-sample fit. We note this as a methodological correction rather than reporting a before/after accuracy delta, since we did not retain a cleanly matched leaky-vs-fixed comparison under identical settings; the value of the fix is that it makes the evaluation protocol sound, not that it delivers a specific quantified gain.

The broader lesson: when using pretrained models as frozen feature extractors, one must audit not only the downstream train/test split but also the pretraining corpus boundary. Indirect leakage through learned representations is invisible to standard data-pipeline checks and requires domain-specific temporal reasoning to detect.

7 Related Work

7.1 Long-Horizon Forecasting with Transformers

Transformer-based forecasters have become the dominant paradigm for long-horizon time-series prediction, a task that also arises in per-entity outcome modeling such as clinical survival estimation (Zhu et al. 2026). Informer (Zhou et al. 2021) introduced ProbSparse attention to reduce complexity from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log L)$; Autoformer (Wu et al. 2021) added decomposition blocks; FEDformer (Zhou et al. 2022) moved to frequency-domain attention; PatchTST (Nie et al. 2023) showed that patching input sequences into subseries tokens further improves long-range forecasting; and TFT (Lim et al. 2021) introduced static-plus-dynamic feature gating for multi-horizon forecasting. On the non-Transformer side, DeepAR (Salinas et al. 2020) trains a global autoregressive RNN across many related series, N-BEATS (Oreshkin et al. 2020) uses pure MLP stacks with doubly residual connections, and Prophet (Taylor and Letham 2018) fits additive trend-plus-seasonality models with holiday regressors.

With the exception of TFT, which gates static and dynamic features separately, the above operate primarily on univariate or multivariate time-series inputs without relational graph structure or sparse categorical item attributes; recent work has even questioned whether Transformers are necessary at all for long-horizon forecasting, showing that a plain linear model can match or beat them on several standard benchmarks (Zeng et al. 2023; Qi et al. 2025). None addresses the multi-modal fusion problem that SkinCast targets: jointly encoding macro player-preference series, a heterogeneous interaction graph, sparse item attributes, and release-context fields into a unified representation for cold-start forecasting of items that do not yet exist. Our work extends the Informer lineage by wrapping its encoder and decoder inside a four-module multi-modal architecture with a local-plus-global positional encoding, and by demonstrating that the fusion yields $\sim 20\%$ RMSE reduction over a plain Informer baseline on the same data.

7.2 AI-Driven Production Decision Support

AI in games has been widely studied from the *player* perspective: skill assessment (Zhang et al. 2022), matchmaking and position-aware team formation (Fan et al. 2024; Wang et al. 2026), player preference prediction (Fan et al. 2021, 2023; Chen et al. 2023a,b), match outcome forecasting, and dynamic difficulty adjustment. On the *production* side (deciding which content to create, when to release it, and how to price it), published systems are scarce, though large-scale deployed recommendation and ranking systems at YouTube, Google Play, Alibaba, and Tencent (Covington, Adams, and Sargin 2016; Cheng et al. 2016; Pi et al. 2020; Choi et al. 2026; Zhang et al. 2026; Shao et al. 2026; Fan et al. 2022c, 2026b,a) demonstrate how ML models are operationalized at production scale in industry. E-commerce demand forecasting for new products shares the cold-start challenge, typically addressed via meta-learned user preference estimators (Lee et al. 2019) or item-attribute similarity, but without the relational graph structure (player–hero–skin) that is unique to games. Heterogeneous graph Transformers (Hu et al. 2020) offer a general architecture for such multi-type relational data, though they are not tailored to the cold-start forecasting problem. Retail assortment optimization (Ferreira, Lee, and Simchi-Levi 2015; Basavaraju and Valilai 2025) combines forecasting with inventory decisions but does not address the 180-day per-day trajectory granularity that game producers require for ROI planning.

To our knowledge, SkinAdvisor is the first deployed system that combines multi-modal sales forecasting with an end-to-end ROI ranking pipeline for game content production planning. It differs from player-facing game AI in that its end-users are internal producers rather than players, and from e-commerce demand forecasting in that it must handle a heterogeneous graph with cold-start items, a fixed 180-day horizon, and a multi-candidate comparison UX that turns predictions into actionable production shortlists.

8 Conclusion

We have presented SkinAdvisor, an AI-driven decision framework for skin production planning in online games, powered by SkinCast, a multi-modal Transformer that forecasts 180-day per-day sales trajectories for unreleased skins by jointly modeling temporal, relational, attribute, and contextual signals. Offline experiments demonstrate that SkinCast reduces RMSE by $\sim 20\%$ over the strongest baseline (Informer) and $\sim 60\%$ over classical methods, with ablations identifying the main contributing modules. The system has been deployed in production with stress-tested p99 latency under 8ms at typical concurrency and over 1100 requests/second at peak load.

The framework could extend to other content-production settings. Future work will incorporate concept art, community sentiment, and patch notes through vision-language encoders (Banerjee, Srivathsan, and Gupta 2026; Paul, Jiang, and Chen 2026) and retrieval-augmented knowledge to complement the current structured inputs.

Acknowledgements

This work was supported by the Shenzhen Science and Technology Program (JCYJ20240813141417023) and the Natural Science Foundation of Guangdong Province of China (2025A1515010233).

References

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450*.

Baltrušaitis, T.; Ahuja, C.; and Morency, L.-P. 2019. Multimodal Machine Learning: A Survey and Taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2): 423–443.

Banerjee, A.; Srivathsan, K.; and Gupta, S. K. 2026. Experience with single domain generalization in real world medical imaging deployments. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press. ISBN 978-1-57735-906-7.

Basavaraju, K.; and Valilai, O. F. 2025. Developing a demand planning strategy for joint forecasting and employing analytical tool in an empirical case study. *Discover Applied Sciences*, 7.

Box, G. E. P.; and Jenkins, G. M. 1970. *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day.

Chen, J.; Chen, Z.; Wang, M.; Fan, G.; Zhong, G.; Liu, O.; Du, W.; Xu, Z.; and Gong, Z. 2023a. A Neural Inference of User Social Interest for Item Recommendation. *Data Science and Engineering*, 8(3): 223–233.

Chen, J.; Fan, G.; Gong, Z.; Li, X.; Leung, V. C.; Wang, M.; and Yang, M. 2023b. A topic-aware graph-based neural network for user interest summarization and item recommendation in social media. In *International Conference on Database Systems for Advanced Applications*, 537–546. Springer.

Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M.; Anil, R.; Haque, Z.; Hong, L.; Jain, V.; Liu, X.; and Shah, H. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS)*, 7–10.

Choi, J.; Lee, Y.; Oh, H.; Shin, S.; Kim, J.; Kim, W.; and Kim, K.-E. 2026. Optimizing preferential rate in retail lending with causal inference and domain adaptation. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press. ISBN 978-1-57735-906-7.

Covington, P.; Adams, J.; and Sargin, E. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys)*, 191–198.

Fan, G.; Geng, B.; Tao, J.; Wang, K.; Fan, C.; and Zeng, W. 2022a. PPPNE: Personalized Proximity Preserved Network Embedding. *Neurocomputing*, 472: 103–112.

Fan, G.; Meng, K.; Chu, H.; Jiang, Y.; Zhu, H.; and Zheng, B. 2026a. From Waste to Discovery: Recycling High-Scored Unexposed Candidates in Taobao Feed Recommendation. arXiv:2611.05997.

Fan, G.; Zhang, C.; Chen, J.; Li, B.; Xu, Z.; Li, Y.; Peng, L.; and Gong, Z. 2022b. Field-aware Variational Autoencoders for Billion-scale User Representation Learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 3413–3425. IEEE.

Fan, G.; Zhang, C.; Chen, J.; Li, P.; Li, Y.; and Leung, V. C. M. 2023. Improving Rating Prediction in Multi-Criteria Recommender Systems via a Collective Factor Model. *IEEE Transactions on Network Science and Engineering*, 10(6): 3633–3643.

Fan, G.; Zhang, C.; Chen, J.; and Wu, K. 2021. Predicting Ratings in Multi-Criteria Recommender Systems via a Collective Factor Model. In *DeMaL Workshop @ The Web Conference 2021 (WWW Companion)*, 1–6. Ljubljana, Slovenia: ACM.

Fan, G.; Zhang, C.; Wang, K.; and Chen, J. 2022c. MV-HAN: A Hybrid Attentive Networks based Multi-View Learning Model for Large-scale Contents Recommendation. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 1–5. ACM.

Fan, G.; Zhang, C.; Wang, K.; Li, Y.; Chen, J.; and Xu, Z. 2024. CUPID: Improving Battle Fairness and Position Satisfaction in Online MOBA Games with a Re-matchmaking System. *Proceedings of the ACM on Human-Computer Interaction*, 8(CSCW2): 1–39.

Fan, G.; Zhao, N.; Meng, K.; Luo, C.; Fu, Y.; Chu, H.; Liu, J.; Jiang, Y.; and Zheng, B. 2026b. Uniboost: Global Coordination with Value Alignment for Fair and Efficient Traffic Allocation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, 4572–4577. New York, NY, USA: Association for Computing Machinery.

Ferreira, K. J.; Lee, B. H. A.; and Simchi-Levi, D. 2015. Analytics for an Online Retailer: Demand Forecasting and Price Optimization. *Manufacturing & Service Operations Management*, 18(1): 69–68.

Friedman, J. H. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5): 1189–1232.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 855–864.

He, X.; Deng, K.; Wang, X.; Li, Y.; Zhang, Y.; and Wang, M. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 639–648.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020. Heterogeneous Graph Transformer. In *Proceedings of The Web Conference (WWW)*, 2704–2710.

Lee, H.; Im, J.; Jang, S.; Cho, H.; and Chung, S. 2019. MeLU: Meta-Learned User Preference Estimator for Cold-Start Recommendation. In *Proceedings of the 25th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, 1073–1082.

Lim, B.; Arik, S. O.; Loeff, N.; and Pfister, T. 2021. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *International Journal of Forecasting*, 37(4): 1748–1764.

Nie, Y.; Nguyen, N. H.; Sinthong, P.; and Kalagnanam, J. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations (ICLR)*.

Oreshkin, B. N.; Carpov, D.; Chapados, N.; and Bengio, Y. 2020. N-BEATS: Neural Basis Expansion Analysis for Interpretable Time Series Forecasting. In *International Conference on Learning Representations (ICLR)*.

Paul, S.; Jiang, H.; and Chen, C. 2026. Automatic funny scene extraction from long-form cinematic videos. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI Press. ISBN 978-1-57735-906-7.

Pi, Q.; Zhou, W.; Zhang, Y.; Wang, Z.; Cao, L.; Wang, F.; Kuang, K.; Xiao, Z.; Gu, J.; Zhu, Z.; Zhu, X.; Li, S.; Zhang, Y.; and Gai, K. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*, 2685–2692.

Qi, Y.; Hu, H.; Lei, D.; Zhang, J.; Shi, Z.; Huang, Y.; Chen, Z.; Lin, X.; and Shen, Z.-J. M. 2025. TimeHF: Billion-Scale Time Series Models Guided by Human Feedback. arXiv:2501.15942.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088): 533–536.

Salinas, D.; Flunkert, V.; Gasthaus, J.; and Januschowski, T. 2020. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *International Journal of Forecasting*, 36(3): 1181–1191.

Shao, Z.; Chen, H.; Lu, S.; Zhu, F.; Ma, Y.; Fan, G.; Gao, J.; Shi, W.; Liu, Y.; and Chu, H. 2026. UGR: Unified Generative Ranking for Heterogeneous E-commerce Recommendation. *arXiv preprint arXiv:2610.39426*.

Taylor, S. J.; and Letham, B. 2018. Forecasting at Scale. *The American Statistician*, 72(1): 37–45.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 5998–6008.

Volkovs, M.; Yu, G. W.; and Poutanen, T. 2017. DropoutNet: Addressing Cold Start in Recommender Systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 4937–4946.

Wang, K.; Fan, G.; Zhang, C.; Jiang, Y.; and Liu, Y. 2026. CHAMP: Cross-domain Hybrid Architecture for Matchmaking and Prediction in Online Multi-Player Games. *arXiv preprint arXiv:2609.04870*.

Wu, H.; Xu, J.; Wang, J.; and Long, M. 2021. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 22419–22430.

Zeng, A.; Chen, M.; Zhang, L.; and Xu, Q. 2023. Are Transformers Effective for Time Series Forecasting? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 11121–11128.

Zhang, B.; Zheng, B.; Yi, C.; Lai, C.; Chen, D.; Wang, D.; Guo, G.; Zhu, J.; Wu, J.; Yu, J.; et al. 2026. DREAM Technical Report. *arXiv preprint arXiv:2608.09408*.

Zhang, C.; Wang, K.; Chen, H.; Fan, G.; Li, Y.; Wu, L.; and Zheng, B. 2022. QuickSkill: Novice Skill Estimation in Online Multiplayer Games. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM)*, 3644–3653. ACM.

Zhang, T.; Liu, Y.; Chen, X.; Huang, X.; Zhu, F.; and Zheng, X. 2021. GPS: A policy-driven sampling approach for graph representation learning. *arXiv preprint arXiv:2112.14482*.

Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; and Zhang, W. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 11106–11115.

Zhou, T.; Ma, Z.; Wen, Q.; Wang, X.; Sun, L.; and Jin, R. 2022. FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting. In *International Conference on Machine Learning (ICML)*, 27268–27286.

Zhu, Z.; Sha, X.; Yin, C.; Fan, G.; and Lu, W. 2026. DeepOSSA: a deep learning framework for personalized survival prediction in osteosarcoma patients. *Frontiers in Cell and Developmental Biology*, 14: 1868333.