

Scalable Explainability for Music Recommendation via Precomputed Reasoning

Aileen Mi^{1,*†}, Jianhong Chen^{1,†}, Zhonghao Luo¹, Tim Greer¹, Ajita Mishra¹ and Amina Shabbeer¹

¹Amazon Music, San Francisco, California, USA

Abstract

Personalized playlists are a core experience in music streaming, yet generating titles and descriptions that both represent playlist content and read as more than generic templates remains difficult at scale. We show that high-quality explanations can be generated largely offline, removing expensive online LLM inference without sacrificing quality. Our baseline generates each explanation online with Qwen3.5-4B. It is cheaper per request than a large proprietary model, but retains two limitations: the work is repeated for every playlist without amortizing over reusable content, and the model’s capacity caps output quality. We propose two methods that move reasoning off the generation path. Nearest Seed Lookup (NSL) removes online inference entirely, retrieving explanations from a precomputed per-seed index. Two-Stage Atom Composition (2SAC) has a large model (e.g. Qwen3.5-122B) precompute per-track descriptive “atoms” offline, and a tiny model (e.g. Qwen3.5-0.8B) assemble them into an explanation for each playlist. In a seven-criteria LLM-as-a-Judge evaluation over 1,000 production playlists, NSL removes online inference at a cost of 6.8pp in description accuracy, while 2SAC improves title accuracy by 5.5pp over the baseline at 5x throughput and 6x lower latency on identical hardware. Our results support a general principle: product-critical quality should be enforced by the structure of the pipeline rather than by prompt guidelines, which additionally removes the online model’s tendency to editorialize.

Keywords

music recommendation, explainability, playlist description generation, large language models, scalable inference

1. Introduction

Personalized music playlists are a staple of modern streaming services, yet the algorithmic logic behind track groupings remains opaque to listeners. A playlist titled “My Rock Mix”—naming the playlist’s primary genre—tells the user little about *why* these particular tracks were assembled or what distinguishes this mix from another. Natural-language explanations, i.e. a descriptive title and short summary, can bridge this gap, helping users understand the sonic character of a playlist and decide whether to engage. In this paper, we present two scalable methods for generating such explanations by moving expensive reasoning offline.

1.1. The Personalized Playlist Surface

We study a large-scale personalized playlist surface that presents themed mixes to individual listeners. Each playlist consists of up to 50 tracks. In the base experience, playlists carry a templated title derived from the playlist’s primary genre (e.g., “My Rock Mix”) and a generic description listing the first three artists. We address the task of replacing these templates with natural-language titles and descriptions that are creative, accurate, and informative.

MuRS 2026: 4th Music Recommender Systems Workshop, September 28th, 2026, Minneapolis, Minnesota, USA

*Corresponding author.

†These authors contributed equally.

✉ aileenmi@amazon.com (A. Mi); jhchenjh@amazon.com (J. Chen); zhongluo@amazon.com (Z. Luo); greertim@amazon.com (T. Greer); ajitamis@amazon.com (A. Mishra); ashabb@amazon.com (A. Shabbeer)



© 2026 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

1.2. The Explanation Task

We define an *explanation* as natural language that accurately captures the dominant semantic and sonic characteristics of a playlist, communicates them concisely, and helps listeners anticipate the listening experience before playback, all while remaining grounded in the playlist’s actual tracks. The system must produce a *title* (“My *<theme>* Mix”, 3–5 words) and a *description* (a single sentence, ≤ 15 words summarizing genre and mood). At generation time, the system has access to per-track catalog metadata (genre, mood, era, artist tags) and sonic descriptions from an upstream audio-understanding model.

Quality desiderata. We define four dimensions of output quality, each corresponding to a distinct failure mode of LLM-generated text:

- **Accuracy:** The title and description must faithfully represent the playlist’s dominant genre, era, and mood.
- **Tone:** Language must be natural and appropriate for a consumer product: concise, direct, and free of marketing prose, lyrical filler, or forced enthusiasm.
- **Safety:** Outputs must not contain offensive, divisive, emotionally manipulative, or suggestive language.
- **Creativity:** Titles must go beyond mechanical concatenation of input tags. A title that simply echoes the playlist facet provides no editorial value over the base experience.

Why this is hard. A playlist explanation must compress a *set* of tracks—potentially spanning multiple genres, eras, and moods—into a single coherent concept that captures what makes the collection distinctive. Ideally every track, not just those carrying the dominant genre, relates to the explanation. This must hold under production constraints: millions of playlists daily, at low latency and bounded compute cost. Invoking a large language model for every playlist is operationally infeasible, while a purely template-based approach sacrifices the expressiveness that makes explanations valuable.

1.3. Our Contribution

In this work, we study this problem on a large-scale personalized playlist surface and propose two methods that move expensive reasoning off the playlist generation path. Both replace per-playlist LLM inference with precomputed content, keeping only a lightweight assembly stage online. We make the following contributions:

1. **Nearest Seed Lookup (NSL) Method:** We precompute explanations for seed tracks offline and retrieve the nearest seed at generation time, removing LLM inference from the generation path and resolving the scalability constraint.
2. **Two-Stage Atom Composition (2SAC) Method:** Building on the insights from NSL, we further propose a two-stage approach that separates reasoning from serving. An offline stage precomputes per-track descriptors — an atom — with a large model, optimizing for output quality. An online stage assembles explanations with a tiny model, optimizing for serving efficiency. By separating these concerns, 2SAC jointly achieves high quality and low serving cost.
3. **Seven-criteria LLM-as-a-Judge evaluation framework:** We create an evaluation framework that assesses the explanation task for accuracy, tone, safety, and novelty, calibrated against author review of judge verdicts.
4. **Offline-Online Design Principle:** Through comprehensive evaluations of the proposed methodologies, we summarize the core principle for scalable explainability: keeping the online model constrained to selection and assembly, while pushing accuracy and creativity into the offline stage, improves title accuracy while substantially reducing generation cost.

2. Related Work

Prior work on explanation generation follows three designs. The first keeps a large model on the serving path [1, 2]. The second trains a small model to generate explanations directly [3, 4]. The third generates text once for a unit coarser than the playlist and amortizes that cost across the playlists the unit produces [5]. The first two designs trade quality against serving cost. The third avoids that trade, but its text is generated for the coarser unit rather than for the individual playlist. Below we review the three lines of work we build on: playlist captioning and music description, distillation, and LLM serving in recommender systems.

2.1. Playlist Captioning and Music Description

Describing a playlist in natural language is known as *playlist captioning*, introduced by Choi et al. [6]. Dedicated caption models address it directly: PlayNTell [7] is a multi-modal encoder–decoder that injects linguistic and musical knowledge to caption editorial playlists, while related work generates titles via machine translation [8] or from artist information [9]. These target editorial or user-created playlists, which are comparatively few and static. We target algorithmically assembled per-listener playlists, which are regenerated daily and must additionally satisfy a fixed product title template.

Closest to our setting is the captioning system deployed at Deezer [5]. It applies an LLM to aggregated catalog tags, which is the same recipe as our baseline. Unlike our baseline, it generates captions for artist clusters rather than for individual playlists, amortizing generation cost across the playlists each cluster produces. NSL adopts the same amortization at the granularity of seed tracks, whereas 2SAC precomputes per-track components and composes them for each playlist.

Separately, music captioning translates audio or catalog metadata into text for *single* tracks [10, 11]; we consume per-track description of this kind as an input signal rather than emit it as output. Work on playlist generation [12, 13] and user perception [14, 15, 16] concerns *which* tracks to select, which we take as given.

2.2. Distilling and Composing with Small Models

To deploy efficient models, standard distillation transfers a large model’s step-by-step reasoning into a smaller one [3, 4, 17, 18]. Because the student learns from a fixed set of teacher outputs, at inference it must condition on its own generations rather than the teacher’s; on-policy distillation addresses this mismatch by having the teacher correct the student’s own outputs during training [19]. Distillation is orthogonal to where reasoning is placed: it makes a small model better at a task, whereas our methods change what the online model is asked to do. The two approaches can therefore be combined. A distilled student is also a plausible alternative to both of our methods, which we do not evaluate in this work and revisit in §7.

2.3. LLMs in Recommender Systems and Scalable Serving

LLMs are increasingly applied across the recommendation stack [1, 2, 20], including large-scale industrial systems: PLUM adapts pretrained LLMs for generative retrieval at billion-user scale [21], and GR2 trains a generative *reasoning* re-ranker with reinforcement learning [22]. Both keep reasoning on the serving path, where even with continuous batching and model parallelism, real-time inference struggles to meet catalog-scale latency and cost budgets. This motivates two-stage designs that separate heavy reasoning from light serving [23, 24].

Taken together, these threads each address part of our problem but none the whole. Caption models describe single tracks or static editorial playlists; LLM-based recommenders keep reasoning on the serving path; and amortizing generation over a coarse unit gives up playlist-level specificity. Our contribution is to place reasoning offline at per-track granularity and leave only assembly online, which we develop in §4.

3. Baseline Approach

3.1. Whole-Playlist LLM Generation

Let a playlist $P = \{t_1, \dots, t_n\}$ be a set of n tracks (typically $n \approx 50$). Each track t_i carries a natural-language *sonic description* $s(t_i)$, produced offline by an upstream audio model, capturing tempo, instrumentation, production style, and mood (e.g., “mid-tempo reggae with a laid-back groove and clean offbeat guitar”). Reading all n descriptions $\{s(t_i)\}_{i=1}^n$ would be most faithful but is infeasible in production: the input grows with n , and the cost is paid for *every* playlist. The baseline therefore gives up this signal and conditions on a compact, playlist-level tag profile X_P instead.

Each track carries catalog tags grouped into four types: genre, era, mood, and theme. X_P counts how many playlist tracks carry each tag, retains only those appearing on at least 60% of tracks, and groups them by type. Tags below this threshold are dropped so the model does not fixate on minority signals. The baseline then generates the title and description in a single online call, $(\text{title}, \text{desc}) = M_{\text{gen}}(X_P; \pi)$, where M_{gen} is a compact whole-playlist generator, instantiated here as Qwen3.5-4B [25]. The elaborate system prompt π compensates for the thin input by encoding the output’s creativity, accuracy, and safety constraints: the title must follow “My $\langle \text{theme} \rangle$ Mix”, add a non-tag creative word, and preserve the dominant genre; the description must stay grounded in X_P and avoid templated phrasing. Playlists for the same listener are generated sequentially to reduce cross-playlist repetition.

This baseline is the reference point against which we measure all subsequent methods. However, it has two fundamental limitations: it cannot feasibly serve at production scale, and its output quality is capped. Like all methods in this work, it uses off-the-shelf models zero-shot with no task-specific training. *The core problem this work addresses is how to raise that quality ceiling without incurring a large model’s cost for every playlist.*

3.2. Limitations

Scalability (operational). Explanation generation sits on the playlist generation path, since every request triggers a forward pass of M_{gen} . Sustaining this for a high-volume production surface would require an always-on accelerator fleet dedicated purely to explanation — a cost that does not amortize, because the work is redone for every playlist even when the underlying tracks have not changed. Per-playlist LLM inference is therefore not a viable production design.

Quality ceiling (model capacity and input granularity). The second limitation is more consequential for *quality*, and stems from two compounding sources. *Model capacity* is the primary one. Lifting quality through training — for example, distilling a smaller model from a larger one — is a complementary direction we leave to future work (§7). Without it, quality is bounded by the model’s capacity: the limited vocabulary and reasoning budget of M_{gen} push even well-prompted titles and descriptions toward generic, repetitive phrasing that misses what makes a playlist distinctive. Prompt iteration improved these outputs during development, but with diminishing returns that left the gap unclosed. *Input granularity* compounds the problem: the coarse input X_P is itself a concession to feasibility, and it caps quality in two further ways regardless of model size:

- *Coarseness.* The per-track sonic character $s(t_i)$ that actually distinguishes playlists — tempo, instrumentation, production style, mood shading — is collapsed into a handful of tag counts. The model cannot see it, so its text defaults to generic phrasing (“upbeat pop with a modern feel”) regardless of a playlist’s real sonic identity.
- *Noise.* Catalog tags are unordered and imperfect, so the aggregate X_P can surface a misleading dominant label. For example, a playlist that is sonically reggae may contain tracks whose tag lists lead with “Alternative Rock”; X_P then over-weights a genre the playlist does not really represent. Having no sonic signal to fall back on, the model cannot correct it, yielding an inaccurate title.

4. Methodology

The limitations of the baseline approach motivate two solutions: **NSL** and **2SAC**. NSL addresses the scalability issue by moving expensive LLM reasoning off the generation path through offline pre-computation of explanations for a single representative track per playlist. While this significantly reduces latency, it introduces quality challenges because the playlist is represented by a single track’s explanation. To balance scalability and quality, we propose the 2SAC method that divides the generation task into two stages: a large model generates compact per-track descriptors once offline, and a lightweight model assembles them into titles and descriptions online.

4.1. NSL: Nearest Seed Lookup

Idea. Personalized playlists are constructed starting from a small set of *seed tracks*—tracks drawn from the listener’s history that anchor the playlist’s theme and guide the retrieval of similar tracks to fill the remaining slots. While the number of possible playlists is enormous, seeds come from a finite song catalog and recur across users (e.g., a popular track by a major artist may seed playlists for thousands of listeners). We can therefore precompute an explanation for each seed once, offline, and reuse it at runtime through a simple lookup.

Offline stage. Let $e(t) \in \mathbb{R}^d$ be a track’s embedding representation learned by moving sonically and behaviorally similar tracks together. For each seed t in the catalog seed pool $\mathcal{S}$, we retrieve its k -nearest neighbor tracks $N_k(t)$ using cosine similarity. We then aggregate the neighbors’ tags into a profile $X_{N_k(t)}$ using the same coverage-threshold construction as in §3.1, and pre-compute the explanation using the same generator and prompt as the baseline:

$$(\text{title}_t, \text{desc}_t) = M_{\text{gen}}(X_{N_k(t)}; \pi). \quad (1)$$

Results are stored in a lookup table mapping from a track to a pair of (title, description) $\mathcal{T} : t \mapsto (\text{title}_t, \text{desc}_t)$. Optionally, N inference variants per seed track are sampled and stored to add diversity.

Online stage. Given a playlist with the seed set $Q = \{q_1, \dots, q_m\}$, we compute the centroid in the embedding space $\bar{e} = \frac{1}{m} \sum_{j=1}^m e(q_j)$, and select the seed closest to the centroid as the representative track,

$$q^* = \arg \max_{q \in Q} \cos(e(q), \bar{e}), \quad (2)$$

We then return the corresponding title and description $\mathcal{T}[q^*]$.

Discussion. NSL eliminates online LLM inference entirely: since titles and descriptions are generated offline in batch, the online lookup scales easily to millions of playlists. However, because NSL shares the same generator and coarse tag-profile input as the baseline, it inherits the same quality ceiling—the precomputed text is only as faithful as $X_{N_k(t)}$ allows. Two further gaps arise from serving a single seed’s title and description for a whole playlist: 1) the nearest-seed text under-represents the playlist when seed tracks are diverse and span multiple facets, and 2) text diversity is limited because a small pool of precomputed strings is reused across playlists.

4.2. 2SAC: Two-Stage Atom Composition

Idea. Building upon the idea of offline generation, 2SAC evolves the offline and online stages into a composition framework: A heavy-weight *offline* stage reasons over each track’s full sonic metadata to produce a compact, composition-ready descriptor — an *atom* that serves as a fundamental building block. A light-weight *online* stage mechanically assembles a playlist’s atoms into a title and description. In the offline stage, we can afford to use a larger model and richer signals that previous methods had to avoid due to scaling and cost constraints. In the online stage, we can leverage an agile model to maintain diversity and quality.

Offline stage — atom generation (large model). For each track t in the catalog, we input a large model M_{atom} (e.g. Qwen3.5-122B) its nearest neighbor tracks $N_k(t)$, its sonic description $s(t)$, and its raw tags, and output a compact *atom* descriptor

$$a(t) = M_{\text{atom}}(N_k(t), s(t), \text{tags}(t)) = (w_t, g_t, \phi_t), \quad (3)$$

where w_t is a single evocative *title word* (a mood/image, not a genre), g_t is a *reconciled genre* — one specific genre label — and ϕ_t is a short, factual *mood phrase* describing the sound. Instead of trusting t ’s raw tags which are usually unordered and noisy, g_t is drawn from the neighborhood’s genre distribution and grounded in $s(t)$, allowing the model to resolve ambiguity when tags disagree. Returning to the §3.2 example, the reggae track’s neighborhood is predominantly reggae, so $g_t = \text{reggae}$ even though its raw tags led with “*Alternative Rock*”. Atoms are stored in a hash table $\mathcal{A} : t \mapsto a(t)$ to be used in the online stage.

Online stage — atom composition (tiny model). Given a playlist $P = \{t_1, \dots, t_n\}$, we gather its atom for each track $a(t_i)$ and compose the explanation with a lightweight, tiny model M_{comp} (e.g. Qwen3.5-0.8B):

$$(\text{title}, \text{desc}) = M_{\text{comp}}(\{a(t_i)\}_{i=1}^n; \pi'). \quad (4)$$

The prompt π' restricts the tiny model to *selection and slot-filling*: pick a representative title word and the majority genre to form a title of the form “My {title} {genre} Mix,” and fold a few mood phrases into one grounded description sentence. Since all creative and accuracy-critical reasoning was already done in the offline stage, the tiny model only selects from and assembles precomputed components.

Discussion. All the creative, accuracy, and safety challenges are offloaded entirely to the offline atom stage, while the online model remains purely mechanical. Since atoms are per-track and reusable, the offline stage amounts to a one-time batch process. The resulting atoms become a catalog-level asset, shared across playlists and, in principle, across any surface that renders those tracks. Only the online composition, a single small-model call, remains on the generation path. 2SAC thus resolves both baseline limitations: scalability (cheap online assembly) and input granularity (the per-track sonic signal $s(t)$ enters the pipeline through the offline atoms). The labels *large* and *tiny* denote relative capacity rather than specific checkpoints. 2SAC requires only that M_{atom} be substantially larger than M_{comp} .

5. Evaluation and Results

Evaluating the quality of playlist titles and descriptions presents challenges common to open-ended text generation: outputs are subjective, high-dimensional, and resist automated scoring via traditional metrics. We adopt an LLM-as-a-Judge (LAJ) framework [26], using a strong model (Claude Opus 4.6 via Amazon Bedrock) to evaluate the outputs of each strategy.

5.1. Setup

Dataset. We evaluate on 1,000 playlists sampled from production traffic, each from a different customer. Playlists are drawn by stratified sampling over genre facets, covering more than 600 unique genres, so that popular genres are not over-represented relative to the long tail. Sampling one playlist per customer avoids correlated observations from listeners who receive several playlists. All three methods are evaluated on this same set of playlists.

Generation settings. All methods use off-the-shelf models zero-shot with no task-specific training. Generation runs at temperature 0.7, chosen after comparing lower values during development. Judges run at temperature 0 to make verdicts reproducible.

Judge Design. We translate the four quality desiderata of §1.2 into seven binary (pass/fail) judges powered by Claude Opus 4.6, splitting accuracy and tone by output field (title vs. description), adding output integrity as a guardrail, and scoring creativity as title novelty. Table 1 summarizes each judge’s scope. Each judge is implemented as an MLflow Guidelines scorer with explicit pass/fail criteria, an

Table 1

LLM-as-a-Judge criteria.

Judge	Scope
Safety	Title + description jointly
Output integrity	Generation artifacts
Title accuracy	Genre identifiable, no hallucinated genres
Description accuracy	Claims grounded in input signals
Title tone	Natural, not robotic or overly poetic
Description tone	Direct, no filler phrases
Title novelty	At least one non-tag word

Table 2

Seven-criteria LLM-as-judge pass rates (scale 0-1, higher is better).

Judge	Baseline (4B)	NSL	2SAC (0.8B)
Title accuracy	0.892	0.86	0.947
Description accuracy	0.918	0.85	0.902
Title tone	0.995	1.000	0.992
Description tone	0.926	0.91	0.980
Title novelty	0.943	0.97	0.924
Safety	0.998	1.000	0.998
Output integrity	0.994	1.000	0.993

ACCEPTABLE clause to prevent over-strict evaluation, and domain-specific calibration against manual review. Judge prompts were developed iteratively, with the authors reviewing judge verdicts on a 50-sample development set and revising the pass/fail criteria until disagreements became rare.

Limitations. Our evaluation relies on a single LLM judge, and LLM judges are themselves fallible: they are sensitive to prompt wording and exhibit systematic biases such as verbosity and self-preference [26, 27]. Calibration aligned each judge with our reading of the guidelines but did not measure its reliability on held-out data, so the pass rates in §5.2 carry an unmeasured judge error term. LAJ evaluation is also an offline proxy: it measures adherence to style and accuracy guidelines but not user engagement or listening behavior. Pass rates should therefore be read as relative comparisons under a fixed rubric rather than as absolute quality estimates or predictors of downstream metrics.

5.2. Quality: Three-Way Comparison

Table 2 presents per-judge pass rates across the three methods. Title accuracy is the most discriminating criterion, spanning 0.86 to 0.947, while the remaining criteria sit closer to ceiling. All three achieve comparable safety ($\geq 99.8\%$) and output integrity ($\geq 99.3\%$), confirming that changes in serving architecture do not introduce content-safety regressions.

2SAC improves title accuracy and description tone. The gains are +5.5pp on title accuracy and +5.4pp on description tone, reflecting 2SAC’s richer input signals and constrained online composition. The remaining five criteria differ from the baseline by at most 1.9pp in either direction, and we do not draw conclusions from differences of this size. Because 2SAC raises offline model capacity and enriches the input signal at the same time, this comparison does not isolate their individual contributions.

NSL trades description accuracy for serving cost. NSL eliminates online inference but shows a −6.8pp drop in description accuracy. This is expected: a single precomputed seed description cannot fully represent a diverse playlist.

Novelty-metric caveat. The novelty judge flags titles whose every word copies an input tag. We observe that Baseline and NSL score higher partly because their prompts mandate a non-tag word, and NSL’s score is further inflated by an evaluation mismatch: titles are precomputed against seed-

Table 3

Generation cost comparison. NSL requires no GPU; relative costs normalized to baseline.

	Baseline (4B)	NSL	2SAC (0.8B)
Online model size	4B	—	0.8B
Throughput/instance	12 playlists/s	—	58 playlists/s
p50 latency	1,290ms	<10ms	211ms
p99 latency	1,720ms	<10ms	368ms
Cost/M playlists	1.0×	<0.15×	0.2×

neighborhood tags but judged against playlist tags. Therefore we caution that small differences on this metric should not be interpreted as meaningful quality gaps.

5.3. Scalability Comparison

Table 3 compares generation cost on the same gpu type (L40S GPU, FP8 quantization). On identical hardware, 2SAC’s composer M_{comp} achieves approximately $5\times$ the throughput of the baseline’s M_{gen} per instance, with a $6\times$ reduction in p50 latency. At equivalent production volume, this translates to a $5\times$ reduction in fleet size and serving cost. The throughput gain reflects not only the smaller model but also a dramatically shorter context: by moving reasoning offline, 2SAC reduces online input from ~ 1300 tokens (system prompt + tag profile + CoT) to ~ 120 tokens (composition prompt + atom palette), and output from ~ 130 tokens (reasoning + title + description) to ~ 30 tokens (title and description only). NSL eliminates GPU serving entirely, requiring only storage for the precomputed lookup table.

6. Discussion

Our comparison reveals two transferable lessons about where to place reasoning compute:

Decouple quality dimensions from the runtime model. The baseline shows that a single prompt cannot reliably carry creativity, accuracy, and safety at once: despite a prompt π encoding all three concerns, M_{gen} ’s title accuracy falls short of what factoring the problem achieves (Table 2). 2SAC factors the problem: creativity and accuracy live in the offline atom stage (where M_{atom} selects evocative title words against robust catalog data), safety in a dedicated moderation layer. The online M_{comp} only assembles precomputed components, so each dimension is addressed independently rather than balanced within a single prompt.

Guarantee quality structurally, not by prompt. Soft guidelines are unreliable for small models: during baseline development M_{gen} produced flowery prose despite instructions to be concise. 2SAC narrows what the online model can get wrong. Creativity is precomputed as a per-track title word w_t by the large offline model, so M_{comp} receives a non-tag word rather than having to invent one, and the template “My $w^* g^*$ Mix” places it in the output by construction. Tone is controlled by the same restriction to *selection* and *slot-filling* over a fixed palette. Because enforcement resides in the prompt π' rather than in the pipeline, the property holds for most outputs but is not guaranteed: 92.4% of 2SAC titles satisfy the judge’s non-tag criterion (Table 2, with the caveats noted in §5.2). We discuss structural enforcement of this property in §7. Product-critical quality dimensions are best enforced by pipeline structure rather than prompt guidelines.

7. Future Work

Atoms as a surface-agnostic building block. Per-track atoms are keyed on catalog tracks, not by the surface that renders them. This makes them reusable across any recommendation context that presents tracks to users. Because the online composition step meets strict latency requirements ($\sim 200\text{ms}$), it can

be inserted into diverse serving paths without architectural changes. We plan to extend atom-based generation to additional surfaces beyond the personalized playlist context studied here.

Structural enforcement of the output template. The title template is instructed in the prompt π' and is therefore not guaranteed (§6). A natural next step is to move enforcement out of the model, assembling the title from selected atom fields in code and constraining decoding for the description, so that the format and the presence of a non-tag word become invariants rather than instructions.

Training-based composition. The online composer is used zero-shot and does not always follow the template (§6). Supervised fine-tuning or on-policy distillation [19] from a larger teacher on composition traces could improve its adherence, complementing the move of enforcement into code. A distilled whole-playlist generator is also a natural comparison point for the methods presented here, since it would lower per-playlist cost without moving reasoning offline.

Serving via CPU. The online composer (M_{comp}) is small enough that quantizing it for CPU inference would significantly expand scaling headroom, since CPU instances are more readily available than GPU at our scale. The tradeoff is increased per-playlist latency, which we plan to characterize.

Online engagement validation. The evaluation presented in this work measures text quality via offline LLM-as-a-Judge proxies. The natural next step is online experimentation to measure whether quality improvements in accuracy, creativity, and tone translate to measurable gains in user engagement. Deployment of playlist captioning at Deezer reports double-digit engagement gains from adding captions to an otherwise unchanged recommender [5], which indicates such gains are attainable but leaves open whether they scale with caption quality.

8. Conclusion

We presented two approaches to generating natural-language explanations for personalized playlists at scale. NSL solves the feasibility problem: eliminating online inference entirely via precomputed lookup, but inheriting the baseline’s coarse input and its associated quality ceiling. 2SAC addresses both feasibility and quality: by pre-computing per-track atoms with a large model that has access to rich sonic signals, it raises title accuracy over the baseline while keeping the online step to a single sub-billion-parameter call. The central lesson is that compute placement, i.e. separating expensive reasoning from playlist generation, enables scalable explanation generation without sacrificing quality. In future work, we plan to post-train the online composer using supervised fine-tuning and on-policy distillation [19], improving its adherence to the output template without returning reasoning to the generation path.

Declaration on Generative AI

During the preparation of this work, the authors used Claude for grammar and spelling checks. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

- [1] W. Hua, L. Li, S. Xu, L. Chen, Y. Zhang, Tutorial on large language models for recommendation, in: Proceedings of the 17th ACM Conference on Recommender Systems, 2023, pp. 1281–1283.
- [2] K. Bao, J. Zhang, Y. Zhang, W. Wang, F. Feng, X. He, Tallrec: An effective and efficient tuning framework to align large language model with recommendation, in: Proceedings of the 17th ACM conference on recommender systems, 2023, pp. 1007–1014.
- [3] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhosht, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, T. Pfister, Distilling step-by-step! outperforming larger language models with less training data and smaller

- model sizes, in: Findings of the Association for Computational Linguistics: ACL 2023, 2023, pp. 8003–8017.
- [4] Y. Fu, H. Peng, L. Ou, A. Sabharwal, T. Khot, Specializing smaller language models towards multi-step reasoning, in: International Conference on Machine Learning, PMLR, 2023, pp. 10421–10430.
 - [5] M. Delcluze, L. Briand, B. Chapus, D. Mekik, G. Salha-Galvan, Music playlist captioning at scale with large language models, 2026. URL: <https://arxiv.org/abs/2606.22460>. arXiv:2606.22460.
 - [6] K. Choi, G. Fazekas, B. McFee, K. Cho, M. Sandler, Towards music captioning: Generating music playlist descriptions, 2017. URL: <https://arxiv.org/abs/1608.04868>. arXiv:1608.04868.
 - [7] G. Gabbolini, R. Hennequin, E. Epure, Data-efficient playlist captioning with musical and linguistic knowledge, in: Y. Goldberg, Z. Kozareva, Y. Zhang (Eds.), Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2022, pp. 11401–11415. URL: <https://aclanthology.org/2022.emnlp-main.784/>. doi:10.18653/v1/2022.emnlp-main.784.
 - [8] S. Doh, J. Lee, J. Nam, Music playlist title generation: A machine-translation approach, 2021. URL: <https://arxiv.org/abs/2110.07354>. arXiv:2110.07354.
 - [9] H. Kim, S. Doh, J. Lee, J. Nam, Music playlist title generation using artist information, 2023. URL: <https://arxiv.org/abs/2301.08145>. arXiv:2301.08145.
 - [10] I. Manco, E. Benetos, E. Quinton, G. Fazekas, Muscaps: Generating captions for music audio, in: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–8.
 - [11] S. Doh, K. Choi, J. Lee, J. Nam, Lp-musiccaps: Llm-based pseudo music captioning, in: Proceedings of the 24th International Society for Music Information Retrieval Conference (ISMIR), 2023.
 - [12] G. Bonnin, D. Jannach, Automated generation of music playlists: Survey and experiments, ACM Computing Surveys (CSUR) 47 (2014) 1–35.
 - [13] R. Dias, D. Gonçalves, M. J. Fonseca, From manual to assisted playlist creation: a survey, Multimedia Tools and Applications 76 (2017) 14375–14403.
 - [14] S. J. Cunningham, D. Bainbridge, A. Falconer, “more of an art than a science”: Supporting the creation of playlists and mixes, in: Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR), 2006, pp. 240–245.
 - [15] A. N. Hagen, The playlist experience: Personal playlists in music streaming services, Popular Music and Society 38 (2015) 625–645.
 - [16] H. Schweiger, E. Parada-Cabaleiro, M. Schedl, The impact of playlist characteristics on coherence in user-curated music playlists, EPJ Data Science 14 (2025) 24.
 - [17] S. Li, J. Chen, Y. Shen, Z. Chen, X. Zhang, Z. Li, H. Wang, J. Qian, B. Peng, Y. Mao, et al., Explanations from large language models make small reasoners better, arXiv preprint arXiv:2210.06726 (2022).
 - [18] J. Huang, S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, J. Han, Large language models can self-improve, in: Proceedings of the 2023 conference on empirical methods in natural language processing, 2023, pp. 1051–1068.
 - [19] M. Song, M. Zheng, A survey of on-policy distillation for large language models, arXiv preprint arXiv:2604.00626 (2026).
 - [20] J. Zhang, R. Xie, Y. Hou, W. X. Zhao, L. Lin, J.-R. Wen, Recommendation as instruction following: A large language model empowered recommendation approach, ACM Transactions on Information Systems 43 (2025) 1–37.
 - [21] R. He, L. Heldt, L. Hong, R. Keshavan, S. Mao, N. Mehta, Z. Su, A. Tsai, Y. Wang, S.-C. Wang, X. Yi, L. Baugher, B. Cakici, E. Chi, C. Goodrow, N. Han, H. Ma, R. Rosales, A. Van Soest, D. Tandon, S.-L. Wu, W. Yang, Y. Zheng, PLUM: Adapting pre-trained language models for industrial-scale generative recommendations, arXiv preprint arXiv:2510.07784 (2025).
 - [22] M. Liang, Y. Li, J. Xu, K. Asadi, X. Liu, S. Gu, K. Rangadurai, F. Shyu, S. Wang, S. Yang, Z. Li, J. Liu, M. Sun, F. Tian, X. Wei, C. Sun, J. Tao, S. Mei, W. Chen, S. Kolay, S. Pandey, H. Firooz, L. Simon, GR2: Generative reasoning re-ranker, arXiv preprint arXiv:2602.07774 (2026).
 - [23] Z. Yue, S. Rabhi, G. d. S. P. Moreira, D. Wang, E. Oldridge, Llamarec: Two-stage recommendation using large language models for ranking, arXiv preprint arXiv:2311.02089 (2023).
 - [24] S. Doh, K. Choi, J. Nam, Talkplay: Multimodal music recommendation with large language models,

arXiv preprint arXiv:2502.13713 (2025).

- [25] Qwen Team, Qwen3.5: Towards native multimodal agents, 2026. URL: <https://qwen.ai/blog?id=qwen3.5>.
- [26] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, I. Stoica, Judging LLM-as-a-judge with MT-Bench and chatbot arena, in: Advances in Neural Information Processing Systems, volume 36, 2023.
- [27] D. Li, B. Jiang, L. Huang, A. Beigi, C. Zhao, Z. Tan, A. Bhattacharjee, Y. Jiang, C. Chen, T. Wu, K. Shu, L. Cheng, H. Liu, From generation to judgment: Opportunities and challenges of llm-as-a-judge, 2025. URL: <https://arxiv.org/abs/2411.16594>. arXiv:2411.16594.