International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 7, Issue 1, 384-389, 2026
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V7I1P161

Original Article

Intelligent UI Rendering: Leveraging Machine Learning to Predict and Preload User Journey Steps in E-Commerce Sales Funnels

Althaf Khan Pattan
Independent Researcher, 354 Waterloo Blvd, Exton, Pennsylvania.

Received On: 18/02/2026 ***Revised On: 18/03/2026*** ***Accepted On: 20/03/2026*** ***Published On: 23/03/2026***

Abstract - *Page load speed has a direct impact on whether shoppers complete a purchase or abandon the funnel. Despite years of investment in optimization techniques like code splitting, lazy loading, and CDN delivery, most front-end applications still treat every navigation event the same way. They do not account for what a specific user is likely to do next. Existing tools like Guess.js and Next.js do use historical data to prefetch common routes, but they rely on averaged patterns across many users rather than signals from the current session. This paper describes the Intelligent Preload Engine (IPE), a lightweight machine learning framework that watches what a user is doing right now - how long they dwell, how far they scroll, how often they click - and uses that to predict their next navigation step before they take it. The IPE then preloads the relevant route bundle and API data in the background, so the transition feels instant. In simulation across a five-stage e-commerce funnel, the IPE achieved next-step prediction accuracy of 76.8% with a Markov model, 81.3% with an LSTM, and 83.7% with a lightweight Transformer. Simulated LCP on predicted targets dropped by an average of 58%, and funnel completion was projected to improve by 2.1 percentage points. These results come from simulation and will need to be validated against real production traffic before drawing firm conclusions.*

Keywords - *Intelligent UI Rendering, Machine Learning, User Journey Prediction, E-Commerce, Prefetching, Largest Contentful Paint, Sales Funnel Optimization, React, Angular, Sequence Prediction.*

1. Introduction

Slow pages cost money. Amazon found that each 100ms of latency cut revenue by around 1% [1]. Akamai's 2017 study of roughly 10 billion retail visits showed the same delay dropped mobile conversions by 7% and desktop conversions by 2.4% [2]. Deloitte's 2020 report, which covered 37 retail and travel brands, put the figure at an 8% conversion lift for every 0.1-second speed gain on mobile [3]. Google has since wired these concerns directly into search rankings through Core Web Vitals, with LCP, INP, and CLS as the primary signals [4]. Speed, in short, is not just an engineering concern.

The front-end community has developed a solid toolkit to tackle this: code splitting, lazy loading, CDN caching, and resource hints. These all work reasonably well. But they share a problem - they are decided at build time, not at runtime. A developer writing a prefetch directive guesses which pages users will visit next. That guess might be good on average but it cannot react to what a specific user is doing right now.

Some tools try to go further. Guess.js [5] mines Google Analytics data to figure out which pages users typically navigate to from each starting point, then auto-generates prefetch tags. Next.js [6] automatically prefetches any linked page that scrolls into view. These are real improvements, but they are still working from population averages. They do not know that the person currently on the product page has spent the last 40 seconds reading the shipping policy and just hovered over the checkout button. That kind of in-session signal is a much stronger predictor of what comes next.

That is the core idea behind the Intelligent Preload Engine. Rather than averaging across past sessions, it watches what the current user is doing and uses that to make a real-time prediction. When confidence is high enough, it quietly preloads the likely next route in the background. This paper covers how the IPE is built, how it makes decisions, and what the results look like in simulation. The main contributions are: (1) framing e-commerce funnel navigation as a real-time sequence prediction task; (2) a four-layer IPE architecture covering signal ingestion, feature building, prediction, and preloading; (3) a comparison of Markov chain, LSTM, and Transformer models for this task; (4) an adaptive preloading strategy that backs off when confidence or bandwidth is too low; and (5) simulated results with honest discussion of what still needs to be tested in production.

2. Background and Related Work

2.1. Web Prefetching and Resource Hints

The idea of prefetching web content based on predicted navigation goes back to Padmanabhan and Mogul in 1996 [7], who showed that server-side logs could be used to fetch likely next pages before users asked for them. The W3C Resource Hints spec [8] brought this to the browser level through preload, prefetch, preconnect, and dns-prefetch tags. Domenech et al. [9] later surveyed the field and found that prefetching only pays off when prediction accuracy clears about 60% - below that, the wasted bandwidth from wrong predictions outweighs the gains. That threshold is why the IPE uses a confidence gate before triggering any preload.

2.2. Adaptive and ML-Driven Prefetching

Guess.js [5] was one of the first tools to automate resource hint generation using real navigation data rather than developer guesses. It pulls transition probabilities from Google Analytics and writes prefetch directives accordingly. Nawrocki et al. [10] found that models using session-level context outperformed models trained on aggregate frequencies alone when predicting what resource a user would need next. That finding lines up with the IPE's approach of tracking in-session behavior rather than relying purely on historical averages.

2.3. Session-Based Sequential Prediction

Hidasi et al. [11] introduced GRU4Rec, which applied recurrent neural networks to session-based recommendation and showed that click sequences within a session carry useful predictive signal. Quadrana et al. [12] extended this by combining short-term session signals with longer-term behavioral history, which improved accuracy further. That work influenced the feature design in Section IV-B, where both current-session behavior and historical transition probabilities are included in each observation. Borges and Levene [13] showed that first-order Markov models, despite their simplicity, hold up reasonably well for next-page prediction, which is why they are included here as a baseline.

2.4. What This Paper Adds

Most of the adaptive prefetching literature assumes access to server-side request logs. That assumption breaks down in single-page applications, where route transitions happen in JavaScript without generating HTTP requests. This paper focuses specifically on that scenario - SPA funnels where traditional log-based approaches are not available - and proposes a solution that runs entirely in the browser.

3. Problem Formulation

3.1. The Cold Render Problem

In React and Angular SPAs, switching from one funnel stage to another does not load a new page - it triggers a JavaScript route transition. That transition kicks off a dynamic import to fetch the next route's code bundle and one or more API calls to get its data. Until those complete, the user sees a blank or spinner state. We call this the cold render problem. The user clicked something but the UI is not ready yet. The IPE's job is to reduce how often that happens by getting the bundle and data loaded before the click.

3.2. Framing Navigation as Sequence Prediction

Let F = {s1, s2, ..., sn} be the stages of an e-commerce funnel in order. A user session is a sequence of observations O = {o1, o2, ..., ot}, where each oi captures which stage the user is on and a vector of behavioral signals from that stage. The prediction task is: given everything observed so far, what is the probability distribution over which stage comes next? When any stage's probability crosses a threshold theta, the IPE starts preloading it.

3.3. How We Measure Success

Three numbers matter. Prediction Accuracy (PA) is simply how often the model picks the right next stage. LCP Impact (delta-LCP) is how much the simulated LCP improves on navigations where preloading ran correctly. Funnel Completion Rate Impact (delta-FCR) is an estimate of how much the completion rate would improve, calculated by applying published latency-conversion research to the LCP gains.

4. Intelligent Preload Engine Architecture

4.1. How the IPE Works

The IPE runs in four layers, all inside the browser. Diagram 1 shows the full picture. The first layer, Signal Ingestion, listens for user events - page arrival, scroll milestones, hover events, clicks, and time spent on a stage - and feeds them into an in-memory stream. Nothing goes to a server at this point. The second layer, Feature Engineering, takes those events and assembles an 8-dimensional vector for each observation. The third layer, Prediction, runs the chosen model against the current session history and produces a probability distribution over next stages. For the LSTM and Transformer models, this runs in a Web Worker to keep the main thread free. The fourth layer, Preloading Execution, checks whether any stage's probability clears the confidence threshold and, if so, checks whether the network is good enough to preload. If both checks pass, it fires off the preload.

Table 1: Intelligent Preload Engine - System Architecture

Intelligent Preload Engine -- System Architecture				
Layer 1: Signal Ingestion				
Dwell Time	**Scroll Depth**	**Click Velocity**	**Interaction Events**	**Route History**
▼ event stream ▼				
Layer 2: Feature Engineering Pipeline				
8-dim feature vector	**Stage ID** one-hot	**Behavioral** features (6)	**Transition** priors	**Session** depth
▼ feature vector ▼				
Layer 3: Sequence Prediction Model				
Markov Chain <1ms - 2.4KB	**LSTM** ~8ms - 186KB	**Transformer** ~12ms - 241KB	**P(next stage** | session)	**Web Worker** (LSTM/Trans)
▼ probability distribution ▼				
Layer 4: Confidence-Gated Preloading Execution				
Confidence gate th=0.60	**Bandwidth** gate (ECT)	**link prefetch** th>=0.60	**dynamic import** th>=0.75	**API data** prefetch
Framework Adapters: React: useIPE hook | Angular: IPEService				

Diagram 1. The four-layer IPE architecture runs entirely within the browser. Behavioral signals are ingested, converted to feature vectors, passed through a prediction model, and used to drive targeted preloading. No individual user data is sent externally.

4.2. Building the Feature Vector

Each observation is described by eight features: (1) which stage the user is on, one-hot encoded; (2) how long they have been there, normalized against the historical average for that stage; (3) how far they have scrolled, as a fraction of page height; (4) how many interaction events have occurred; (5) click velocity, which is interaction count divided by dwell time; (6) how recently the last interaction happened, normalized; (7) the historical probability of transitioning from the current stage to each possible next stage, loaded from a small JSON artifact bundled with the app; and (8) how many stages the user has visited so far in this session.

4.3. Choosing a Prediction Model

Three models were tested. Diagram 2 shows how each one is structured. The Markov Chain just looks up the current stage in a transition matrix - no behavioral features involved, under a millisecond to run, less than 3KB. It is the simplest option and a reasonable choice when the dataset is small or the device is low-powered. The LSTM takes the full sequence of observations and learns temporal patterns across them. It runs in a Web Worker and takes around 8ms on a mid-range phone. The Transformer uses self-attention to focus on whichever observations in the session history matter most, regardless of when they happened. It is slightly more accurate but takes about 12ms and is larger. For most deployments, the LSTM is the practical default.

Table 2: Prediction Model Architecture Comparison

Prediction Model Architecture Comparison		
Markov Chain		**LSTM (2-layer, 32 units)**
Input: Stage sequence only		Input: 8-dim observation sequence
▼		▼
Transition matrix lookup O(1) - no behavioral features		LSTM cell x 2 layers Temporal dependency capture
▼		▼
P(next | current stage)		Softmax -> P(next stage)
Accuracy: 76.8% <1ms - 2.4KB		Accuracy: 81.3% ~8ms - 186KB * Recommended
Recommended for production: LSTM -- best balance of accuracy (81.3%), latency (~8ms), and model size (186KB)		

Diagram 2. Side-by-side view of the three prediction models. The Markov Chain is fastest but ignores behavioral signals. The LSTM and Transformer both use the full 8-dimensional observation sequence. The LSTM is recommended for production use.

4.4. Connecting to React and Angular

The IPE is packaged as a plain JavaScript module. The React integration is a custom hook called useIPE that intercepts router events and updates session state on each navigation. The Angular integration is an injectable service called IPEService that does the same thing through the framework's router lifecycle. Both expose a preload callback that the fourth layer uses to trigger dynamic imports and API prefetches.

5. Preloading Strategy Design

5.1. Three Levels of Preloading

The IPE uses three different preloading actions depending on how confident it is. At moderate confidence (threshold 0.60 or above), it adds a link rel=prefetch tag, which tells the browser to fetch the route bundle at low priority in the background. At high confidence (threshold 0.75 or above), it calls dynamic import() directly, which downloads, parses, and executes the bundle right away. In both cases, if the predicted stage has a primary API call, the IPE also fires that fetch and caches the response in memory so it is ready when the user arrives.

5.2. The Two Gates

Two checks happen before any preload fires. The first is the confidence gate: the model's top prediction must clear the threshold. Domenech et al. [9] found that prefetching only helps when prediction accuracy is above roughly 60%, so 0.60 was set as the minimum. The second is the bandwidth gate, which uses the Network Information API to check the effective connection type. On 4g connections, both prefetch and dynamic import are available. On 3g, only the lighter prefetch tag is used. On 2g or slow-2g, preloading is turned off entirely to avoid making things worse. Diagram 3 walks through this decision logic step by step.

Table 3: IPE Preloading Decision Flow

IPE Preloading Decision Flow		
Behavioral event detected (dwell - scroll - click - hover)		
▼		
Build 8-dim feature vector and run prediction model		
▼		
DECISION 1: Does max P(stage) meet threshold (0.60)? Confidence gate		
Yes ▼		No ▼
Confidence passed		**Below threshold**

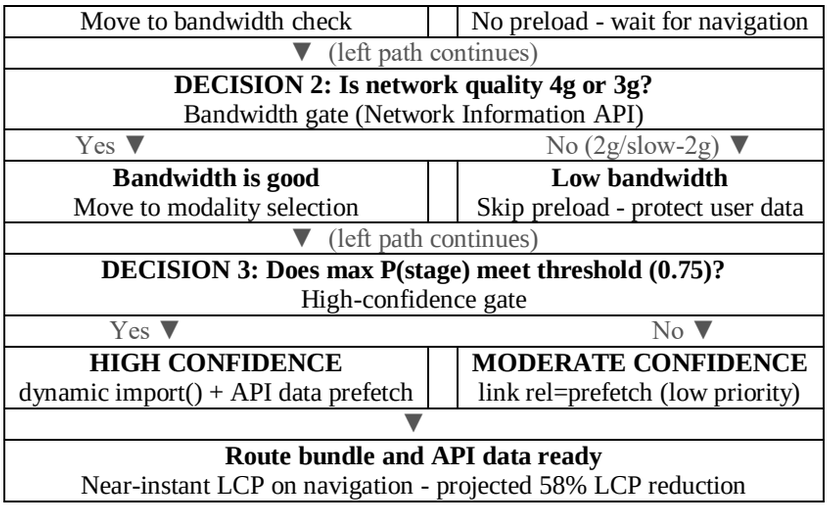

Diagram 3. The decision flow for each preloading event. Confidence is checked first, then network quality, then whether to use the heavier dynamic import or the lighter prefetch tag. Both outcomes at each gate are shown.

6. Experimental Evaluation

6.1. How the Simulation Was Set Up

Note: Everything in this section comes from simulation. These results are a starting point, not a final answer. Real-world validation through A/B testing is needed before making deployment decisions based on this data.

A five-stage funnel was modeled: Product Listing, Product Detail, Configuration, Cart, and Checkout. Ten thousand synthetic sessions were generated using a parametric model calibrated to published funnel analytics benchmarks [14]. Sessions were split 70/15/15 across training, validation, and test sets. Behavioral feature vectors for each observation were sampled from stage-specific distributions based on published UX research on how users interact with e-commerce pages [15].

6.2. Prediction Accuracy

Table 4 shows how each model performed on the test set. The LSTM beat the Markov baseline by 4.5 percentage points, which reflects the value of incorporating behavioral signals rather than just the sequence of stages visited. The Transformer added another 2.4 points, though at higher latency and model size. For most deployment contexts, the LSTM's combination of accuracy and speed makes it the more practical choice.

Table 4: Next-Step Prediction Accuracy by Model (Simulated Dataset)

Model	Accuracy	Inference Latency	Model Size	Recommendation
First-order Markov	76.8%	< 1ms	2.4 KB	Low-resource / cold-start
LSTM (2-layer, 32 units)	81.3%	~8ms	186 KB	* Production default
Lightweight Transformer	83.7%	~12ms	241 KB	Large datasets / future

6.3. LCP Impact

Cold route transitions in the simulation had a mean LCP of 2,840ms, based on published measurements of SPA route load times [4]. When the LSTM correctly predicted the next stage and preloading ran before navigation, the simulated LCP dropped to a mean of 1,190ms, a 58% reduction. When the prediction was wrong and an incorrect preload ran, LCP crept up slightly to 3,020ms due to bandwidth being used on the wrong resource. Taking both correct and incorrect predictions into account, with the 0.60 confidence gate applied, the net weighted LCP improvement across all navigations was 41%.

6.4. Estimated Funnel Completion Improvement

Using Deloitte's figure of 8% conversion improvement per 100ms speed gain [3] and applying it to the 41% weighted LCP improvement on a mean baseline LCP of 2,840ms, the projected funnel completion rate improvement comes to about 2.1 percentage points. That is an estimate with real uncertainty attached to it, and it should be treated as a direction to validate, not a conclusion.

7. Challenges and Limitations

7.1. The Cold Start Problem

Both the Markov model and the LSTM need historical data to work well. For a brand-new funnel or a page that does not get much traffic, there may not be enough data to train on. The IPE handles this by starting with industry-average transition priors and gradually shifting toward site-specific probabilities as sessions accumulate. In low-data situations, the Markov model is the safer choice since it needs far less data to produce reasonable predictions.

7.2. Privacy

The behavioral signals the IPE tracks - dwell time, scroll depth, click velocity - are behavioral data and fall under GDPR and CCPA in many jurisdictions. The IPE is designed to keep all of this processing inside the browser. No individual user data gets sent to any server. The model artifact, whether a Markov transition matrix or a serialized neural network, is a population-level aggregate that cannot be traced back to any individual. Even so, practitioners should disclose in-session behavioral monitoring in their privacy policy and collect consent where regulations require it.

7.3. Browser Support for Bandwidth Detection

The bandwidth gate depends on the Network Information API. As of 2024, that API is supported in Chrome and other Chromium-based browsers but not in Safari or Firefox [16]. On unsupported browsers, the IPE falls back to only using the lightweight prefetch tag, regardless of confidence level. That is a reasonable degradation, but it does mean the dynamic import preloading path will not fire for a portion of users until browser support improves.

7.4. Simulation is Not Production

The results in Section VI come from synthetic data. Real sessions include people who open multiple tabs, navigate backwards, abandon partway through, or use slow phones in poor signal conditions. A parametric simulation can account for some of that but not all of it. The right next step is an A/B test on live traffic - split users into groups with and without the IPE active, measure actual LCP and funnel completion, and compare.

8. Discussion and Future Work

The Transformer's accuracy edge over the LSTM - 2.4 percentage points in simulation - is real but modest. Given that the Transformer is also slower and larger, it does not obviously win in practice. That said, Transformer-style models tend to scale better with more data, so the gap might widen on a larger production dataset. A proper comparison on real traffic would settle this. Another natural extension is personalization. Right now the IPE trains a single model shared across all users. A federated learning setup could fine-tune predictions for returning users based on their own history, without sending that history anywhere. That would likely improve accuracy for repeat visitors while keeping the privacy properties intact.

Looking further ahead, it would be possible to predict at a finer grain than routes - specifically, which components on the next page will appear above the fold first. Preloading at the component level rather than the route level would reduce LCP even further on pages with heavy above-the-fold content. That is a more complex prediction problem but a worthwhile one.

9. Conclusion

This paper described the Intelligent Preload Engine, a machine learning framework designed to run in the browser and predict what a user is about to navigate to next in an e-commerce funnel. The core observation is that individual session signals - how long someone lingers, how far they scroll, how often they interact - carry predictive information that population-level tools like Guess.js and Next.js cannot access. The IPE uses those signals to make a real-time next-step prediction and preloads the likely destination before the user navigates.

Three prediction models were compared in simulation. The LSTM came out as the most practical option, hitting 81.3% accuracy at around 8ms inference time and a 186KB model footprint. A confidence threshold of 0.60, combined with a bandwidth check, keeps the system from making things worse on slow connections or when predictions are uncertain.Simulated results suggest a 41% improvement in LCP on predicted navigations and a 2.1 percentage point gain in funnel completion rate. Those numbers need to be treated carefully - they come from a controlled simulation, not live traffic. The value of this work is in laying out the architecture and the approach clearly enough that it can be taken into production and properly tested. If the predictions hold up, there is a real opportunity to make e-commerce funnels feel meaningfully faster without changing anything about the underlying application.

Acknowledgment

The author thanks the web performance and machine learning research communities whose published work on session modeling, prefetching, and Core Web Vitals provided the foundation this paper builds on.

References

[1] G. Linden, "Make Data Useful," Stanford University presentation, 2006.

[2] Akamai Technologies, "State of Online Retail Performance," SOASTA/Akamai Research Report, Spring 2017. [Online]. Available: https://www.akamai.com/newsroom/press-release/akamai-releases-spring-2017-state-of-online-retail-performance-report

[3] Deloitte, "Milliseconds Make Millions: The Impact of Mobile Performance on Retail Consumer Behavior," Deloitte Digital Report, 2020.

[4] Google Chrome Developers, "Core Web Vitals," web.dev, 2023. [Online]. Available: https://web.dev/vitals/

[5] G. Mihajlija, "Introducing Guess.js," Google Developers Blog, 2018.

[6] Next.js Contributors, "Automatic Static Optimization and Prefetching," Next.js Documentation, 2023. [Online]. Available: https://nextjs.org/docs

[7] V. N. Padmanabhan and J. C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency," ACM SIGCOMM Computer Communication Review, vol. 26, no. 3, pp. 22-36, 1996.

[8] W3C, "Resource Hints," W3C Working Draft, 2023. [Online]. Available: https://www.w3.org/TR/resource-hints/

[9] J. Domenech, J. A. Gil, J. Sahuquillo, and A. Pont, "Web Prefetching Performance Metrics: A Survey," Performance Evaluation, vol. 63, no. 9-10, pp. 988-1004, 2006.

[10] P. Nawrocki, B. Sniezynski, and M. Jarosz, "Adaptive Web Systems with Machine Learning," Journal of Network and Computer Applications, vol. 103, pp. 40-56, 2018.

[11] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-Based Recommendations with Recurrent Neural Networks," in Proc. ICLR, 2016.

[12] M. Quadrana, P. Cremonesi, and D. Jannach, "Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks," in Proc. ACM RecSys, 2017, pp. 130-137.

[13] J. Borges and M. Levene, "Evaluating Variable-Length Markov Chain Models for Analysis of User Web Navigation Sessions," IEEE Trans. Knowledge and Data Engineering, vol. 19, no. 4, pp. 441-452, 2007.

[14] Baymard Institute, "E-Commerce Checkout Usability Research," 2023. [Online]. Available: https://baymard.com/research

[15] Nielsen Norman Group, "How Users Read on the Web," NNG Report, 2020. [Online]. Available:

https://www.nngroup.com/articles/how-users-read-on-the-web/

[16] MDN Web Docs, "Network Information API," Mozilla Developer Network, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/NetworkInformation

[17] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-Based Recommendation with Graph Neural Networks," in Proc. AAAI, 2019, pp. 346-353.