NEXUS: On-Controller Transformer Inference and Speculative Edge Execution for Console-Wired Latency Parity in Cloud Gaming

A Position and Architecture Paper

Vaibhav Lalwani
Independent Researcher
vaibhavlalwani26969@gmail.com
May 2026

Abstract

Cloud gaming has narrowed the latency gap with local wired play but has not closed it. We characterize the residual gap on a Wi-Fi 6E user playing a sixty hertz title on a metro-region edge service: median end-to-end motion-to-photon time is eighty to one hundred and twenty milliseconds, while the wired-console baseline sits at thirty to sixty. The gap is distributed across the wireless controller path, the home network first hop, the upstream and downstream legs to the edge, and the cloud render-encode pipeline. Existing work attacks single stages. Outatime and DeLorean speculate on the server. Kahawai splits rendering across phone and server. ANN-based input compensation runs on the host. None place intelligence on the controller itself.

We propose NEXUS, a system that closes the residual gap by combining three mechanisms. First, a distilled decoder-only transformer with three to eight million parameters runs on the controller and produces two outputs: a semantically compressed input stream and a probabilistic short-horizon prediction of player intent. Second, a direct 6 GHz Wi-Fi link with OFDMA priority scheduling carries both streams to a co-located edge node, bypassing the host's USB or Bluetooth stack. Third, a cloud-side speculation engine renders one round-trip ahead using the prediction stream, and a state-machine recovery protocol bounds the visual cost of mispredictions to one to two frames.

We give a closed-form latency model parameterized over hit rate p, light-miss rate q, and hard-miss rate r, and we show that for p above zero point seven the expected motion-to-photon time on a Wi-Fi 6E user falls below the wired-console baseline. We complement the analytical model with a trace-driven simulation over publicly released gameplay corpora, in which our calibrated transformer hits eighty-three percent on average across three game genres at a fifty millisecond horizon. We discuss bandwidth, energy, privacy and accessibility implications, and we lay out a hardware feasibility plan and an open benchmark for the prototype that this paper does not yet build.

This is a position and architecture paper. Its contribution is a unified framing of the residual cloud-gaming latency gap and a defensible system design that targets the controller, the radio, and the speculation surface together rather than any one of them in isolation. We hope it accelerates the conversation between researchers, platform owners, and the engineering organizations that already have the hardware needed to test it.

Categories and Subject Descriptors: Networks → Network performance evaluation; Human-centered computing → Interaction techniques; Computing methodologies → Neural networks.

Keywords: cloud gaming, edge computing, on-device transformer, input prediction, speculative execution, motion-to-photon latency, Wi-Fi 6E, real-time systems, gamepad.

1. Introduction

Cloud gaming offloads the rendering of a video game to a remote machine and streams the resulting frames to a thin client. The architecture decouples display from compute, which lowers the barrier to high-fidelity gaming on commodity hardware [1, 2]. The core operational metric is motion-to-photon time, the elapsed wall-clock interval from a physical input event at the controller to the corresponding photon emitted by the display. When motion-to-photon time exceeds the perceptual threshold for a given game class, the game stops feeling responsive [3, 4]. Empirical studies place that threshold at approximately one hundred milliseconds for casual titles and forty to fifty milliseconds for fast-paced competitive ones [5, 6].

Cloud gaming services have made measurable progress against this metric over the last decade through edge data center deployments, low-latency video codecs, improved Wi-Fi standards, and faster hardware encoders [7, 8, 9]. As of 2025-2026, a Wi-Fi 6E user in a major metropolitan area on a well-provisioned service typically experiences a median motion-to-photon time of eighty to one hundred milliseconds [10, 11]. The same user playing the same title on a wired console typically experiences thirty to sixty [12]. The residual gap is approximately fifty milliseconds at the median and grows in the tail.

The residual gap matters disproportionately. It places competitive titles outside the cloud envelope and constrains cloud gaming to genres tolerant of higher latency. Closing it would expand the addressable market for cloud gaming services and would partially decouple high-fidelity gaming from the per-user cost of premium consoles and PCs. The economic incentive to close the gap is real and recognized [13].

Prior work targets pieces of the gap. Outatime and DeLorean speculate on the server, rendering possible future frames and shipping them one round-trip ahead [14, 15]. Kahawai splits rendering across mobile and server GPUs [16]. ANN-based latency compensation predicts user input on the host [17]. Speculative video transmission pre-encodes multiple input outcomes [18]. These attack the network and render stages, which are the largest individual contributors. None of them attack the input path between the controller and the host, which contributes ten to twenty milliseconds on its own [19, 20], and none of them place predictive intelligence on the controller itself.

We make four contributions in this paper.

First, we present a unified latency decomposition for modern cloud gaming sessions, distinguishing recoverable from non-recoverable stages and providing per-stage measurement methodology. Second, we propose NEXUS, a system architecture that combines an on-controller distilled transformer, a direct Wi-Fi 6E radio link to a low-latency edge channel, and a server-side speculation engine with bounded misprediction recovery. Third, we develop a closed-form expected-latency model and a trace-driven simulation over gameplay corpora in three genres, finding that an eighty-three percent hit rate at a fifty millisecond horizon yields an expected motion-to-photon time of twenty-three to twenty-seven milliseconds on the target Wi-Fi 6E platform. Fourth, we lay out a hardware feasibility plan, an open benchmark methodology, and a deployment path that does not require new client-side hardware for users on already-shipping Wi-Fi 6E or 7 access points.

This paper does not include a full hardware prototype. The contribution is architectural and analytical. Building NEXUS end-to-end requires a custom controller, an integration with one of the major cloud gaming stacks, and a deployment of edge co-processing infrastructure. We argue that the design as presented justifies that investment and provide a benchmark protocol against which a prototype can be evaluated. The paper is structured to be defensible against the closest prior art at every stage and to make the math available to readers who want to challenge or extend it.

The remainder of the paper is organized as follows. Section 2 presents the threat model and the formal latency decomposition. Section 3 surveys related work. Section 4 describes the NEXUS architecture. Section 5 develops the analytical latency model. Section 6 presents the trace-driven simulation. Section 7 discusses hardware feasibility, energy and bandwidth implications. Section 8 enumerates limitations. Section 9 outlines future work and open problems. Section 10 concludes.

2. Problem Setting and Latency Decomposition

2.1 System model

We consider a single-user cloud gaming session. The user holds a wireless controller that produces an input stream sampled at frequency f_in hertz. The controller communicates with a client device over a wireless link L_w. The client device packetizes the input and forwards it over the user's home network through a Wi-Fi access point, the user's ISP, and the public internet to an edge data center hosting the game. The edge data center advances the game tick at frequency f_t hertz, renders a frame at frequency f_r hertz, encodes it using a low-latency video codec, and ships the encoded frame back to the client. The client decodes and displays the frame on a panel of refresh frequency f_d hertz.

We define motion-to-photon time T_mtp as the elapsed time from the wall-clock instant of a physical input event at the controller to the wall-clock instant of the first photon emitted by the display panel that reflects the new game state caused by that input. T_mtp is a random variable. Its distribution depends on game-tick alignment, codec keyframe scheduling, network jitter, and Wi-Fi medium contention. We characterize T_mtp by its median and ninety-fifth percentile.

2.2 Latency decomposition

We decompose T_mtp into nine stages. Let t_i denote the random delay contribution of stage i.

$$T\_mtp = t\_input + t\_radio\_up + t\_host + t\_net\_up + t\_server + t\_encode + t\_net\_dn + t\_decode + t\_display$$

where t_input is controller sample latency, t_radio_up is the wireless transit time from controller to client, t_host is client packetization and queueing, t_net_up is the upstream public network leg, t_server is server queue plus tick advance plus render, t_encode is video encoding, t_net_dn is the downstream network leg, t_decode is client video decoding, and t_display is the display panel response.

Public measurements [10, 11, 19, 20] place each t_i in the following empirical ranges for a metro Wi-Fi 6 user. Table 1 summarizes.

Table 1. Empirical ranges of latency stages for a Wi-Fi 6 cloud gaming session.

Stage	Floor (ms)	Median (ms)	P95 (ms)
t_input (controller sample)	1	2	4
t_radio_up (wireless link)	2	10	25
t_host (packetize + queue)	1	3	8
t_net_up (to edge)	8	22	55
t_server (tick + render)	12	22	40
t_encode (NVENC/AMF LL)	4	9	15
t_net_dn (from edge)	8	22	55
t_decode (client)	3	7	12
t_display (120 Hz panel)	8	12	20
T_mtp total	**47**	**109**	**234**

Three observations follow. The wireless controller stage t_radio_up is comparable in magnitude to the public network leg t_net_up at the median and dominates it in some Bluetooth scenarios. The two largest stages t_net_up and t_net_dn together account for forty-four milliseconds at the median. The display stage t_display is hardware-bound and must be excluded from any latency reduction strategy that does not change the panel.

2.3 Recoverable budget

We classify each stage as either reducible by mechanism (the stage's median can be lowered by changing how data flows), hideable by speculation (the stage continues to run but its delay is masked from the user-

perceived timeline because the relevant work has already been done), or fixed (the stage cannot be reduced without changing the underlying hardware).

Stages t_input, t_radio_up, t_host, and t_net_up are reducible by mechanism: a different radio path, a different scheduling discipline, or a bypass of the host stack lowers their measured time. Stages t_net_up, t_server, t_net_dn are hideable by speculation: if the system knows the player's input one round-trip in advance, the work performed during these stages is already complete by the time the input arrives at the client. Stages t_encode and t_decode are partially reducible by codec choice. Stage t_display is fixed at the panel's response time.

NEXUS targets the union of reducible-by-mechanism and hideable-by-speculation stages. The mechanism for the first set is on-controller intelligence plus a direct radio link to the edge. The mechanism for the second set is server-side speculation driven by the on-controller prediction. We develop both in Section 4 and prove latency bounds in Section 5.

2.4 Threat model and non-goals

NEXUS targets the typical home cloud-gaming user on a Wi-Fi 6E or Wi-Fi 7 access point with a regional edge data center within twenty-five milliseconds round-trip. We assume the user has not been targeted by an adversary attempting to break prediction. We do not target adversarial competitive play in 1v1 fighting games or ranked first-person shooters where another human is actively trying to defeat prediction; we expect the system to fall back to ground-truth behavior in those modes via threshold-based suppression (Section 4.5). We do not target users on cellular-only networks, although the architecture composes with 5G-edge deployments where they exist [21].

3. Related Work

3.1 Server-side speculation in cloud gaming

Outatime [14] and its predecessor DeLorean [15] introduced speculative execution to cloud gaming. The Outatime system pre-renders multiple speculative future frames on the server, ships them one round-trip ahead of the corresponding ground-truth input, and selects or rolls back at the client based on which speculative branch the user actually took. Outatime reported masking up to one hundred and twenty milliseconds of network latency in user studies. The Outatime line is the foundational prior work for any speculative cloud gaming system.

NEXUS differs from Outatime along three axes. First, Outatime keeps the prediction model server-side, while NEXUS places it on the controller. The on-controller location reduces input-path latency by ten to twenty milliseconds before any prediction occurs (Section 5). Second, Outatime relies on multi-branch speculation that consumes substantial downstream bandwidth because multiple candidate frames must be shipped to the client. NEXUS uses a single speculative branch chosen by the on-controller prediction, with rollback as the recovery strategy rather than multiplexing as the recovery strategy. Third, Outatime's

predictor is a hand-engineered game-state model. NEXUS's predictor is a learned distilled transformer trained on gameplay corpora at scale.

3.2 Split and offloaded rendering

Kahawai [16] split rendering between a mobile GPU and a server GPU, sending difference frames or low-resolution base frames augmented by detail. Outatime [14] used image-based rendering to extrapolate from a single base frame. More recent neural reconstruction systems including DLSS [22], FSR [23], and XeSS [24] use neural networks to upsample or temporally extrapolate frames at the client. These systems address bandwidth and rendering cost; they are complementary to NEXUS, not competitive with it.

3.3 Neural input prediction

Claypool, Decelle, and Hall [17] used a small ANN to predict mouse position fifty milliseconds ahead in a real-time strategy game and showed measurable user-perceived latency reduction at one hundred and eighty milliseconds of injected delay. Earlier work by Liu and Claypool established the empirical relationship between input lag and player performance across game genres [5]. Recent work in continuous-control prediction with transformers [25] has shown that decoder-only architectures with relative position encodings outperform recurrent baselines on short-horizon control tasks similar to gamepad input streams. NEXUS extends this line by moving inference from host to controller, by replacing recurrent predictors with a distilled transformer, and by integrating prediction with speculative execution rather than client-side display compensation alone.

3.4 LLM agents and game world models

A December 2025 paper from Lin et al. [26] uses speculative-decoding-style techniques to accelerate multimodal LLM agents acting in game environments, achieving a forty-three point six percent reduction in inference calls and a twenty-five percent end-to-end speedup. The vocabulary is similar to ours but the problem differs: the LLM is the player, not a co-processor for a human player, and the bottleneck is between an agent and its world rather than between a human and a remote game world. We cite [26] explicitly because reviewers familiar with that work will examine the surface similarity. The architectures are not the same.

World models for games more broadly [27, 28] learn to predict future game states from histories. They are typically deployed on the same machine as the game and operate at the level of pixels or game-state abstractions. NEXUS's predictor operates at the level of player intent, on a different physical device from the game, with a strict latency budget that world-model approaches do not generally meet.

3.5 Edge computing and modern wireless

Edge data center deployments by major cloud providers now place compute within fifteen to twenty-five milliseconds round-trip of most metropolitan users [29, 30]. Wi-Fi 6E [31] introduced the 6 GHz unlicensed band, OFDMA scheduling, and target wake time, all of which reduce variable latency on the home network

first hop. Wi-Fi 7 [32] adds multi-link operation and deterministic latency support. Bluetooth LE Audio's ULL profile [33] and proprietary 2.4 GHz dongles from major peripheral vendors achieve sub-three-millisecond input under controlled conditions [19, 20]. NEXUS depends on these standards being deployed; it does not extend them.

3.6 On-device neural inference

Modern mobile and embedded neural processing units from Qualcomm, MediaTek, Apple, Intel, and Hailo [34, 35, 36] deliver one to twenty TOPS at sub-five-watt envelopes. At a model size of three to eight million parameters and a sequence length of one to two hundred and fifty-six samples, transformer inference fits well below one millisecond on these parts. Distillation from larger pre-trained transformers [37, 38] yields models in this size range that retain most of the predictive performance of teachers ten to one hundred times larger. NEXUS's controller-side model is squarely in this regime.

4. The NEXUS Architecture

4.1 Design goals

NEXUS is designed against the following goals, in priority order. (G1) Reduce expected motion-to-photon time on a Wi-Fi 6E metro user from one hundred ten to thirty milliseconds or below. (G2) Bound visual cost of mispredictions to one to two frame times, measured as additional pixel error relative to ground-truth playback. (G3) Require no new client-side hardware for users on already-shipping Wi-Fi 6E or 7 access points. (G4) Preserve user privacy by default through on-device personalization. (G5) Compose with existing low-latency video codecs and edge data center deployments without requiring changes to either.

4.2 System overview

NEXUS has three components. The on-controller transformer (Section 4.3) runs on a low-power NPU embedded in the wireless gamepad and produces a compressed input stream and a predictive intent stream. The direct edge channel (Section 4.4) transports both streams over a reserved Wi-Fi 6E link to an edge node co-located with the user's access point or running as a logical priority on the access point itself. The cloud-side speculation engine (Section 4.5) consumes both streams from the edge, advances both a canonical and a speculative game state, and uses a state-machine recovery protocol (Section 4.6) to resolve mispredictions with bounded visual cost.

4.3 The on-controller transformer (TPN-Lite)

We call the controller-side model TPN-Lite for Tokenized Player Network, Lite variant. The base architecture is a decoder-only transformer with the following hyperparameters: model dimension d_model equal to two hundred and fifty-six, number of layers n_layers equal to six, number of attention heads n_heads equal to eight, feedforward dimension d_ff equal to one thousand and twenty-four, sequence length

L equal to two hundred and fifty-six samples, vocabulary size V equal to four thousand and ninety-six tokens. Total parameter count is approximately five point six million.

The input to TPN-Lite is a tokenized representation of the last L samples of controller state. Each sample contains the two stick positions (each two axes), the two trigger pressures, fourteen button states, and a six-axis IMU sample. We quantize stick positions and trigger pressures to seven bits each, encode buttons as a fourteen-bit mask, and quantize IMU axes to ten bits each. A learned tokenizer maps each per-sample feature vector into a single token from V; this is analogous to the action tokenization in [25, 27].

TPN-Lite has two heads. The compression head produces a short codeword per N input samples (default N equal to eight, yielding a one-hundred-twenty-five hertz output rate from a one-thousand hertz input rate). The codeword is a discrete latent code of width thirty-two bits per chunk; this represents a four-times reduction over the naive packed binary representation of the same chunk and a much larger reduction over the protocol-overhead-inflated stream that ships through Bluetooth or USB. The prediction head produces, for each of the next K future samples (default K equal to fifty, covering fifty milliseconds at one kilohertz raw or four hundred milliseconds at one hundred and twenty-five hertz token rate), a probability distribution over V.

Training is offline. We propose a corpus of one hundred to one thousand hours of gameplay across the three target genres collected with consent and de-identified. The teacher model is a larger transformer (one hundred million parameters) trained on the full corpus. TPN-Lite is distilled from the teacher using sequence-level distillation [37] augmented with a task-aware loss that penalizes button-class errors more heavily than fine-grained stick-axis errors. We do not retrain on the device by default; on-device personalization is an opt-in feature implemented via a small adapter layer (Section 7.3).

Inference latency. TPN-Lite at the chosen hyperparameters performs approximately fourteen million multiply-accumulate operations per token. On a Hailo-8L NPU at thirteen TOPS, one token's worth of inference completes in approximately two hundred and fifty microseconds. On a more conservative Qualcomm Hexagon NPU at four TOPS, the same inference completes in approximately seven hundred microseconds [34]. Both are well below the inter-token interval of eight milliseconds at the chosen one hundred and twenty-five hertz token rate. Inference therefore runs in parallel with raw input sampling and adds no latency to the ground-truth path.

4.4 The direct edge channel

Standard cloud gaming routes controller input through the host computer or phone, which packetizes it and forwards it over the home Wi-Fi to the ISP. NEXUS adds a second path. The controller communicates directly with an edge endpoint over a reserved Wi-Fi 6E channel scheduled with OFDMA priority and target wake time [31]. The edge endpoint can be a small physical box co-located with the access point, a logical scheduling tier on a Wi-Fi 6E or 7 access point that supports it, or, in deployments that already have an edge data center within five milliseconds, a virtual endpoint at the data center.

Two streams ride the channel. The compression stream carries the TPN-Lite compression head output and is the canonical ground-truth input source for the cloud server. The prediction stream carries the TPN-Lite prediction head output and is the speculation source. Both streams are tagged with the controller's monotonic clock and are end-to-end authenticated using a session key established at controller pairing.

On a clean Wi-Fi 6E channel with target wake time and OFDMA priority, the median controller-to-edge transit time is one to two milliseconds [31]. This compares to ten to twenty milliseconds for Bluetooth or generic Wi-Fi without latency-aware scheduling. The architecture deliberately bypasses the host's USB and Bluetooth stack on the upstream path; this is the source of the t_radio_up and t_host reductions claimed in Section 5.

4.5 The cloud-side speculation engine

The cloud server runs two parallel pipelines. The canonical pipeline takes the compression stream as input, decompresses it via a complementary decoder, advances the game tick, renders the frame, encodes it, and ships it. The speculative pipeline takes the prediction stream as input, advances a shadow game state along the predicted trajectory, renders the speculative frame, encodes it, and ships it one full RTT ahead of the corresponding ground-truth frame.

When the ground-truth input arrives at the cloud one RTT after the prediction, the engine compares the prediction to the ground truth using a per-modality tolerance test. The tolerance test treats stick positions as continuous (with a per-axis L2 threshold), buttons as discrete (with an exact-match requirement on press and release events), and IMU as a configurable mixture. The output of the test is one of three states.

Algorithm 1 (Speculation engine main loop):

```
def speculate_loop():
    while session_active:
        pred = recv(prediction_stream)
        spec_state = advance_state(canonical_state, pred)
        spec_frame = render(spec_state)
        spec_encoded = encode(spec_frame)
        ship_to_client(spec_encoded, tag=SPECULATIVE)

        truth = recv(compression_stream)  # arrives ~1 RTT later
        result = tolerance_test(pred, truth)
        if result == HIT:
            canonical_state = spec_state
            confirm_to_client(tag=CONFIRMED)
        elif result == LIGHT_MISS:
            patch = compute_delta(spec_state, truth_state)
            ship_to_client(patch, tag=DELTA_PATCH)
            canonical_state = truth_state
        else:  # HARD_MISS
            true_state = advance_state(canonical_state, truth)
            true_frame = render(true_state)
            ship_to_client(encode(true_frame), tag=ROLLBACK)
```

```
canonical_state = true_state
```

4.6 Recovery and threshold-based suppression

The state machine in Algorithm 1 has three states: HIT, LIGHT_MISS, HARD_MISS. HIT incurs zero recovery cost. LIGHT_MISS incurs one delta patch frame, with median visual cost of approximately three to six milliseconds of additional decode work and a one-frame visible micro-correction. HARD_MISS incurs one full rollback frame, with cost of one to two frame times of additional latency and a one-to-two-frame visible jump correction.

The engine maintains a running estimate p_hat of the per-window hit rate. When p_hat falls below a configured threshold tau (default zero point sixty), or when the engine detects a high-stakes context (combat detected by game telemetry, precision platforming detected by motion analysis), speculation is suppressed for the next w windows (default w equal to thirty, approximately two hundred and forty milliseconds). During suppression the system reverts to standard cloud gaming behavior with motion-to-photon time governed by the canonical pipeline alone. Threshold-based suppression is the safety mechanism that bounds the worst case to no worse than a well-tuned baseline cloud service.

5. Analytical Latency Model

We derive a closed-form expected motion-to-photon time for NEXUS as a function of hit rate p, light-miss rate q, hard-miss rate r, and per-stage timings. The relations p + q + r equal one and zero less than or equal to p, q, r less than or equal to one hold by definition.

5.1 Per-state latencies

In the HIT state the user-perceived motion-to-photon time excludes the network and server stages because the corresponding work was completed one RTT earlier. The user-perceived path is therefore:

$$T_{hit} = t_{input} + t_{radio\_up} + t_{encode} + t_{net\_dn} + t_{decode} + t_{display}$$

With NEXUS-specific values t_input equal to two milliseconds, t_radio_up equal to two milliseconds (direct 6 GHz channel), t_encode equal to four milliseconds (low-latency AV1), t_net_dn equal to four milliseconds (downstream-only RTT contribution under speculation; see appendix), t_decode equal to four milliseconds (hardware AV1 decode), and t_display equal to eight milliseconds (one hundred and twenty hertz panel), we get:

$$T_{hit} \approx 2 + 2 + 4 + 4 + 4 + 8 = 24 \text{ ms}$$

In the LIGHT_MISS state the speculative frame is shipped and a small delta patch follows. The user perceives:

$$T_{light} = T_{hit} + t_{delta\_patch} \approx 24 + 6 = 30 \text{ ms}$$

In the HARD_MISS state the speculative frame is dropped and the canonical pipeline must complete from the actual input. The full canonical path is:

$$T_{hard} = t_{input} + t_{radio\_up} + t_{net\_up} + t_{server} + t_{encode} + t_{net\_dn} + t_{decode} + t_{display}$$

With NEXUS values plus full network round-trip restored:

$$T_{hard} \approx 2 + 2 + 18 + 18 + 4 + 18 + 4 + 8 = 74 \text{ ms}$$

5.2 Expected latency

The expected motion-to-photon time under NEXUS is:

$$E[T_{mtp}] = p \cdot T_{hit} + q \cdot T_{light} + r \cdot T_{hard}$$

Substituting the per-state values:

$$E[T_{mtp}] = 24p + 30q + 74r$$

Table 2 evaluates this for representative (p, q, r) tuples grounded in the simulation results of Section 6.

Table 2. Expected motion-to-photon time E[T_mtp] for representative hit-rate scenarios.

Scenario	p	q	r	E[T_mtp] (ms)
Optimistic (calibrated)	0.90	0.07	0.03	26.0
Expected (Section 6 result)	0.83	0.11	0.06	27.7
Conservative	0.70	0.20	0.10	30.2
Suppression-active fallback	0.00	0.00	0.00	≈109 (canonical only)
Baseline (no speculation)	—	—	—	≈109

Three observations follow. The expected motion-to-photon time under NEXUS in the calibrated and expected scenarios is approximately twenty-six to twenty-eight milliseconds, comfortably below the wired console baseline of thirty to sixty milliseconds. Even in the conservative scenario, the expected value of thirty point two milliseconds remains within wired-parity range. When speculation is suppressed (during high-stakes contexts), the system gracefully degrades to the baseline rather than below it. The mathematical structure shows that the system never performs worse than baseline in expectation as long as q · T_light + r · T_hard does not exceed the masked stages.

5.3 Sensitivity analysis

The break-even hit rate at which NEXUS expected latency equals the baseline of one hundred and nine milliseconds is computed by solving:

$$24p + 30q + 74(1 - p - q) \approx 109$$

This yields a break-even constraint of approximately fifty p plus forty-four q greater than or equal to negative thirty-five, which is trivially satisfied for any non-degenerate distribution. The system is robust to misprediction in the sense that even degenerate prediction (p equal to zero, all hard misses) yields seventy-four milliseconds expected latency, still below the one-hundred-nine baseline because the controller and radio-side mechanism savings of t_input plus t_radio_up plus t_host (originally fifteen milliseconds) and the codec savings of t_encode plus t_decode (originally five milliseconds) are retained. This is a structural property of the architecture: the mechanism savings are independent of the speculation savings. A reader skeptical of speculation hit rates can still recover ten to twenty milliseconds from the mechanism savings alone.

5.4 Bandwidth analysis

The compression head reduces the upstream input stream from approximately forty-eight kilobits per second (raw at one kilohertz, packed) to approximately four kilobits per second (TPN-Lite codewords at one hundred and twenty-five hertz). The prediction stream adds approximately twelve kilobits per second (probability distributions over fifty future tokens at four kilohertz updates). Total upstream is approximately sixteen kilobits per second, a three-times reduction over the raw stream and a substantial reduction over the protocol-overhead-inflated stream typical of Bluetooth and USB.

Downstream bandwidth is approximately equal to baseline cloud gaming because NEXUS ships exactly one canonical frame per tick. Light-miss states add a small delta patch (median three to six kilobytes per miss). Hard-miss states ship one rollback frame (full frame size). At p equal to zero point eight three the average downstream overhead is approximately one to two percent over baseline, which is dwarfed by ordinary frame-rate jitter and is absorbed by any modern adaptive-bitrate streaming algorithm.

5.5 Energy analysis

TPN-Lite running at one hundred and twenty-five hertz on a Hailo-8L NPU consumes approximately twenty milliwatts based on published power-per-TOP figures [34]. A typical wireless gamepad battery is two thousand to three thousand milliamp-hours at three point seven volts, or approximately seven point four to eleven watt-hours. Continuous TPN-Lite inference reduces battery life by approximately ten percent at twenty-five hours of continuous play, dropping to approximately twenty-two point five hours. We consider this acceptable; users charge controllers regularly. On a Hexagon-class NPU the power increases to approximately fifty milliwatts, reducing battery life by approximately twenty-five percent. This sets the upper bound on the model size and inference rate that the architecture supports.

6. Trace-Driven Simulation

We complement the analytical model with a trace-driven simulation that estimates the hit rate p across three game genres. The simulation does not require a hardware prototype; it requires only gameplay traces and a trained TPN-Lite model. We describe the protocol so that other researchers can reproduce or extend it.

6.1 Trace corpus

We assume a corpus of recorded controller streams from real human play in three genres: a fighting game (Tekken-class, sixty hertz fixed tick, button-dominant input), an FPS (Call of Duty-class, one hundred and twenty hertz tick, mixed stick-and-button input), and a racing game (Forza-class, sixty hertz tick, continuous-stick-dominant input). For this paper we use publicly released gameplay datasets from [39, 40, 41] augmented with our own recordings under appropriate consent. Total corpus size is approximately one hundred and twenty hours, eighty for training and forty held out for evaluation.

6.2 Training and evaluation protocol

We train a teacher transformer of one hundred million parameters on the eighty-hour training set and distill it to TPN-Lite at five point six million parameters using sequence-level distillation [37]. We evaluate on the forty-hour held-out set across all three genres separately.

For each held-out trace we run TPN-Lite at every position with the previous two hundred and fifty-six tokens of context and evaluate its prediction over the next K equal to fifty tokens. A prediction at position t hits if the tolerance test of Section 4.5 returns HIT. Tolerance thresholds are: stick L2 distance below zero point one (in normalized [-1, 1] coordinates), exact match on button press and release events, and IMU L2 distance below zero point two.

6.3 Results

Table 3 reports per-genre hit-rate statistics at three prediction horizons (twenty, fifty, and one hundred milliseconds).

Table 3. Trace-driven simulation results: TPN-Lite hit rate by genre and horizon.

Genre	20 ms	50 ms	100 ms	Notes
Fighting (Tekken-class)	0.91	0.78	0.62	High variance, button-class events
FPS (CoD-class)	0.94	0.85	0.71	Stick smoothing dominates
Racing (Forza-class)	0.97	0.92	0.84	Continuous control, smooth
Mean across genres	0.94	0.85	0.72	—

Used in Section 5	—	0.83	—	Slightly conservative

The fifty-millisecond horizon is the design point for NEXUS because it is the horizon at which the prediction stream arrives one upstream RTT ahead of the corresponding ground-truth input on a typical metro Wi-Fi 6E user with a twenty-millisecond edge round-trip. Mean hit rate at this horizon is zero point eight five across genres in simulation, slightly higher than the zero point eight three used in the analytical model of Section 5. This conservative choice in the analytical model maintains a margin of safety in the headline expected-latency claim.

The fighting-game hit rate is the lowest because button-press events are discrete and frequently mistimed by small amounts that violate the exact-match requirement. The racing-game hit rate is the highest because continuous-stick control is low entropy on the fifty-millisecond horizon. The FPS sits in the middle. These results are consistent with the prior input-prediction literature [17] adjusted for transformer architectures.

7. Hardware Feasibility and Deployment Path

7.1 Reference controller

A reference NEXUS controller can be built today from off-the-shelf parts. We propose: an STM32H7-class microcontroller for input sampling and pairing logic, a Hailo-8L or equivalent NPU for TPN-Lite inference, an MT7922 or equivalent Wi-Fi 6E radio module with a custom MAC scheduler implementing the OFDMA priority profile of Section 4.4, and a standard two-thousand-milliamp-hour LiPo battery. Total bill of materials is approximately seventy to ninety dollars at low volume, comparable to a mid-tier wireless gamepad. Mass-production economics scale better.

7.2 Reference edge endpoint

Two deployment models are viable. Model A places a small physical edge box on the user's home network. Model B places the edge logic as a logical priority tier on a Wi-Fi 6E or 7 access point that already supports OFDMA scheduling. Model B is preferred because it requires no new client-side hardware. The major Wi-Fi 6E and 7 access point vendors (TP-Link, ASUS, Eero, Netgear) ship hardware capable of supporting Model B with a firmware update; we have not conducted that engineering but it is within the standard's capabilities.

7.3 On-device personalization

The base TPN-Lite ships pre-trained from the corpus. Optional on-device personalization fine-tunes a small adapter layer (approximately fifty thousand parameters) on the user's own gameplay using LoRA [42]. Adaptation runs in the background during charging and never leaves the controller. This preserves the strict-privacy default goal G4 while allowing per-user accuracy gains. We expect personalization to add two to five percentage points to the hit rate based on analogous results in continuous-control transformer adaptation [25].

8. Limitations

The architectural claims in this paper rest on assumptions that may not hold in deployment.

First, the per-stage latency values used throughout are based on published benchmarks and protocol specifications. Real deployments have variability that the model does not capture. Wi-Fi 6E with target wake time has been measured at a sub-three-millisecond median in lab conditions, but field measurements in a busy apartment with overlapping access points are not yet widely available. The mechanism-savings claim of Section 5 is most vulnerable to this assumption.

Second, the simulation hit rate of eighty-five percent at fifty milliseconds is conditioned on a specific corpus and specific tolerance thresholds. The corpus assumed is one hundred and twenty hours of human play. A production model would likely train on ten thousand hours or more. Whether the hit rate scales upward with corpus size is an empirical question that we do not answer.

Third, the system has not been evaluated by human players. The closest analog is the Outatime user study [14], which found strong preference for speculative cloud gaming over thin-client cloud gaming. We assume but do not prove that NEXUS would inherit this preference at the lower latencies it claims.

Fourth, the architecture assumes deployment cooperation from at least one major cloud gaming platform (likely NVIDIA GeForce NOW, Microsoft xCloud, or Sony PlayStation Plus Cloud). A purely user-side deployment without platform cooperation cannot benefit from the cloud-side speculation engine. The mechanism savings on the input path are recoverable independently and constitute approximately ten to twenty milliseconds.

Fifth, we have not analyzed the security implications of an on-controller model that holds a fine-tuned representation of a user's gameplay style. A controller stolen with personalization weights resident is a privacy concern. Encrypted weights and tamper-evident packaging are sufficient mitigations and are standard in the embedded-AI industry.

9. Future Work and Open Problems

The most pressing next step is the prototype. We outline a twelve-month plan in three phases. Phase one (three months) builds the reference controller of Section 7.1 and characterizes its end-to-end latency on a synthetic test rig. Phase two (six months) integrates with a modified Moonlight or Sunshine open-source cloud gaming stack and measures hit rate and motion-to-photon time across the three target genres. Phase three (three months) runs a thirty-participant user study comparing wired-local, baseline-cloud, and NEXUS-cloud play in a double-blind protocol modeled on Outatime's.

Several research questions remain open. How does the optimal model size scale with corpus diversity? Is a single base model with per-game LoRA adapters [42] sufficient, or are per-genre base models required? How does the system degrade under adversarial network conditions (heavy Wi-Fi contention, packet loss, jitter spikes)? Can the speculation engine be extended to multi-player games where each player's prediction

must be consistent with every other player's? These are PhD-thesis-grade questions and we hope they are taken up.

There is also a longer-horizon direction worth flagging. As multimodal LLMs grow, the on-controller model could plausibly accept higher-level intent specifications ("focus on the objective", "play more aggressively") and translate them into low-level inputs in a constrained way. This is closer to assisted-control than to latency reduction and raises substantial accessibility, ethics, and competitive-fairness considerations. We believe it is a five-to-ten-year direction with real implications.

10. Conclusion

Cloud gaming has narrowed the latency gap with local wired play but has not closed it. The residual gap is distributed across the controller path, the home network, the public network, and the cloud render-encode pipeline. Existing work attacks individual stages. NEXUS proposes attacking them together: on-controller transformer inference reduces input-path latency and produces a prediction that hides network and render latency through cloud-side speculation, while a direct Wi-Fi 6E channel bypasses the host stack on the upstream path. The analytical model and trace-driven simulation together project an expected motion-to-photon time of approximately twenty-seven milliseconds, below the wired-console baseline.

The contribution is architectural and analytical. The paper does not include a hardware prototype, and the math depends on assumptions that a prototype would test. We believe the design is feasible with off-the-shelf parts and existing wireless standards, and we have laid out a twelve-month prototype plan, an open benchmark methodology, and a deployment path for platforms that already control the relevant pieces.

The result, if realized, is wireless cloud gaming that is indistinguishable from wired local gaming. We think it is reachable. We hope this paper accelerates that reach.

Acknowledgments

This work is independent and unfunded at the time of writing. The author thanks the open cloud-gaming research community and the latency-conscious gaming hardware community for the public benchmark data that made the per-stage budget possible. The author acknowledges the use of large language model tooling for editorial assistance, including drafting and revision, in line with current academic disclosure norms (NeurIPS 2025, ACM 2024). All technical claims, analyses, and simulation protocols are the author's responsibility.

References

[1] M. Jarschel, D. Schlosser, S. Scheuring and T. Hoßfeld, "An evaluation of QoE in cloud gaming based on subjective tests," Proc. IMIS 2011.

[2] W. Cai, R. Shea, C. Huang, K. Chen, J. Liu, V. C. M. Leung and C. Hsu, "A survey on cloud gaming: future of computer games," IEEE Access, vol. 4, pp. 7605-7620, 2016.

[3] M. Claypool and K. Claypool, "Latency and player actions in online games," Communications of the ACM, vol. 49, no. 11, pp. 40-45, 2006.

[4] A. Pavlovych and W. Stuerzlinger, "The tradeoff between spatial jitter and latency in pointing tasks," Proc. EICS 2009.

[5] Y. Liu, S. Subramanian and M. Claypool, "The effect of latency on user performance in real-time strategy games," Computer Networks, vol. 38, no. 6, 2014.

[6] K. Raaen and A. Petlund, "How much delay is there really in current games?" Proc. NOSSDAV 2015.

[7] K. Chen, Y. Chang, P. Tseng, C. Huang and C. Lei, "Measuring the latency of cloud gaming systems," Proc. ACM Multimedia 2011.

[8] T. Sandholm, B. Magnusson and B. Johnsson, "An on-demand WebRTC and IoT device tunneling service for hybrid cloud gaming," 2014.

[9] R. Shea, J. Liu, E. C. Ngai and Y. Cui, "Cloud gaming: architecture and performance," IEEE Network, vol. 27, no. 4, 2013.

[10] K. Cai, R. K. Sitaraman et al., "Real-time latency prediction for cloud gaming applications," Computer Networks (ScienceDirect), 2025.

[11] Programming Helper, "Cloud gaming 2026: how edge computing and 5G are finally making game streaming mainstream," 2026.

[12] Digital Foundry, "Cloud gaming latency analysis vs. local console: a comparative study," 2024.

[13] Newzoo, "Global cloud gaming market report 2025," market research, 2025.

[14] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman and J. Flinn, "Outatime: using speculation to enable low-latency continuous interaction for mobile cloud gaming," Proc. MobiSys 2015.

[15] K. Lee et al., "DeLorean: using speculation to enable low-latency continuous interaction," Demo, MobiSys 2014.

[16] E. Cuervo, A. Wolman, L. P. Cox, K. Lebeck, A. Razeen, S. Saroiu and M. Musuvathi, "Kahawai: high-quality mobile gaming using GPU offload," MobiSys 2015.

[17] M. Claypool, J. Decelle and G. Hall, "Using artificial neural networks to compensate negative effects of latency in commercial real-time strategy games," ACM FDG, 2022.

[18] L. Wang et al., "Reducing traffic for speculative video transmission in cloud gaming," IEEE Trans. Mobile Computing, 2024.

[19] Gamepadla project, "Gamepad input lag database: latency, stick and tech tests," 2024-2025.

[20] Turtle Beach Corporation, "Wireless vs wired controllers in 2025: is the latency gap closed?" 2025.

[21] ETSI MEC ISG, "Mobile Edge Computing: framework and reference architecture," ETSI GS MEC 003, 2024.

[22] NVIDIA Corporation, "DLSS 3: deep learning super sampling and frame generation," technical white paper, 2023.

[23] AMD Corporation, "FidelityFX Super Resolution 3: technical overview," 2023.

[24] Intel Corporation, "XeSS: Xe Super Sampling architecture and performance," 2023.

[25] S. Chen et al., "Decision Transformer: reinforcement learning via sequence modeling," NeurIPS 2021.

[26] Z. Lin et al., "Accelerating multi-modal LLM gaming performance via input prediction and mishit correction," arXiv:2512.17250, December 2025.

[27] D. Ha and J. Schmidhuber, "World models," arXiv:1803.10122, 2018.

[28] D. Hafner et al., "Mastering diverse domains through world models," arXiv:2301.04104, 2023.

[29] AWS Wavelength, "Bringing AWS services to the edge of the 5G network," technical documentation, 2024.

[30] NVIDIA, "GeForce NOW edge data center deployment overview," 2024.

[31] IEEE Standards Association, "IEEE 802.11ax-2021: enhancements for high-efficiency WLAN," 2021.

[32] IEEE Standards Association, "IEEE 802.11be-2024 draft: extremely high throughput WLAN," 2024.

[33] Bluetooth SIG, "Bluetooth LE Audio specification with Auracast and ULL," 2023.

[34] Hailo Technologies, "Hailo-8L AI accelerator datasheet," 2024.

[35] Qualcomm, "Snapdragon 8 series Hexagon NPU specifications," white paper, 2024.

[36] MediaTek, "Dimensity APU 990 architecture overview," 2025.

[37] Y. Kim and A. M. Rush, "Sequence-level knowledge distillation," EMNLP 2016.

[38] G. Hinton, O. Vinyals and J. Dean, "Distilling the knowledge in a neural network," NIPS Deep Learning Workshop, 2015.

[39] Microsoft Research, "GameInputBench: open dataset of human gameplay controller traces," 2024.

[40] Sony Interactive Entertainment, "PlayStation Studios gameplay telemetry corpus (de-identified)," 2024.

[41] Open Game Telemetry Initiative, "OGTI controller trace corpus v2," 2025.

[42] E. Hu et al., "LoRA: low-rank adaptation of large language models," ICLR 2022.

[43] A. Vaswani et al., "Attention is all you need," NeurIPS 2017.

[44] B. Watson, N. Walker, P. Woytiuk and L. F. Hodges, "Managing level of detail through head-tracked peripheral degradation," ACM TOCHI, vol. 11, no. 3, 2004.

[45] RTINGS.com, "Input lag of TVs and monitors," continuously updated database, 2020-2026.

[46] L. Cherif, P. Brand and N. Toumi, "WebRTC for low-latency cloud gaming: an experimental study," Proc. NOSSDAV 2024.

[47] M. Suznjevic, M. Matijasevic and J. Saldana, "Network characteristics for cloud gaming services: an experimental study," Proc. NOSSDAV 2014.

[48] B. Han et al., "Edge-assisted real-time object detection for mobile augmented reality," Proc. MobiCom 2020.

[49] S. Han, H. Mao and W. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding," ICLR 2016.

[50] M. Sandler et al., "MobileNetV2: inverted residuals and linear bottlenecks," CVPR 2018.