
Token Generation Control Method Based on Deterministic Logical Timeline

Achieving Cryptographic-Level Determinism in Large Model
Outputs

Shanghai Huapait Technology Co., Ltd.

Qiu Yong Wang Zhehao

Corresponding author: Qiu Yong

Email: qiuyong@huapait.com.cn

October 2026

Abstract

Text generation by large language models is essentially a probabilistic sampling process: the model computes a probability distribution over the next token based on the input context, and then samples a token from that distribution. This probabilistic design gives large models creativity but also introduces a fundamental engineering problem: the same input may produce different outputs at different times, in different environments, or on different hardware. For AI agents (automated trading, code generation, industrial control) that require stability and reproducibility, this non-determinism is fatal.

This paper proposes a token generation control method based on a deterministic logical timeline. The method uses the logical state $S(n)$ output by a Protocol Security State Machine (PSSM) as the initialization seed of a pseudo-random number generator (PRNG), replacing random sources based on physical time or hardware noise, thereby transforming the token sampling process from “physical non-determinism” to “logical determinism”. The paper provides the complete token generation flow, two working modes (deterministic mode and divergent-but-reproducible mode), a cryptographically bound record format, and, building on the independent verification and replay mechanism disclosed in the applicant’s prior patent, an auditing method for token generation records. The method does not change the model’s native probability distribution (intelligence representation); it only controls the token sampling path, achieving cryptographic-level determinism across environments, hardware, and frameworks.

Keywords: deterministic logical timeline; token generation control; cryptographic entropy source; cross-environment consistency; AI output determinism

1 Introduction

1.1 Non-Determinism in Large Model Token Generation

Text generation by large language models is essentially a probabilistic sampling process. The model computes a probability distribution over the next token (logits) based on the input context, and then samples a token from that distribution using sampling algorithms such as Top-K, Top-p, or Temperature sampling. This probabilistic design gives large models creativity but also introduces a fundamental engineering problem: the same input may produce different outputs at different times, in different environments, or on different hardware.

For AI agents (automated trading, code generation, autonomous driving decisions, industrial control) that require stability and reproducibility, this non-determinism is fatal—developers cannot guarantee that “an Agent behavior that passed tests today will still behave consistently in production tomorrow,” and regulators cannot trace “why a decision was made in a particular way at a particular moment.”

1.2 Limitations of Existing Deterministic Control Schemes

Scheme 1: Setting a random seed

Existing schemes control the initial state of the random number generator by setting a **seed** parameter in the sampling configuration. Fundamental limitations: the seed only controls the sampling layer and cannot eliminate non-determinism introduced in the logits computation stage before sampling (including CUDA kernel execution order, floating-point accumulation order, and algorithm differences selected by cuDNN auto-tuning); cross-environment reproducibility is not guaranteed; the seed source cannot be verified; and there is no cryptographic binding.

Scheme 2: Greedy decoding

By setting Temperature to 0, the model always selects the token with the highest probability. Fundamental limitations: non-determinism at the hardware and operator layers is still not eliminated; the model’s creativity is sacrificed; and it cannot prove that “a given output was indeed produced under greedy decoding.”

Scheme 3: Batch-invariant kernels

Thinking Machines Lab proposed eliminating non-determinism in LLM inference by constructing “batch-invariant kernels.” Fundamental limitations: it only addresses non-determinism at the computational layer and does not involve entropy source control in the token sampling process; it does not establish cryptographic binding; and it cannot support auditing and verification.

Scheme 4: Post-hoc verification

Inference Receipts / AI Integrity Receipts generate lightweight cryptographic commitment records after AI inference is completed. Fundamental limitations: post-hoc verification rather than pre-hoc control; no entropy source control; explicitly does not guarantee universal deterministic replay across hardware or quantization methods; and the security model is based on the “honest emitter” assumption.

The root of these limitations is that existing schemes address “post-hoc verifiability” (what result was produced) rather than “pre-hoc determinism” (how to ensure that the same input produces the same result). These are two fundamentally different technical problems requiring different technical paths.

1.3 Contributions

This paper proposes a token generation control method based on a deterministic logical timeline. The core contributions are:

1. **Entropy source replacement:** replacing the PRNG seed from a “physical-layer non-deterministic source” (system time, hardware noise) with a “cryptographically driven deterministic logical state $S(n)$,” achieving cross-environment consistency;
2. **Path locking:** after $S(n)$ is used as the PRNG seed, the probabilistic walking path of token sampling is permanently locked; the model’s native probability distribution remains unchanged, but the result of “rolling the dice” is cryptographically anchored;
3. **Two working modes:** a deterministic mode (same input + same $S(n)$ $\rightarrow$ exactly the same output) and a divergent-but-reproducible mode (same input + same $S(n)$ + same user random entropy $\rightarrow$ exactly the same output);
4. **Cryptographic binding and auditing:** the token sequence is cryptographically bound to $S(n)$ and n , generating a verifiable generation record; building on the independent verification and replay mechanism disclosed in the applicant’s prior patent, a verifier holding the same DSF can independently replay the token generation process;
5. **Systematic comparison:** the essential differences between this method and existing schemes such as random seed setting, batch-invariant kernels, Inference Receipts, TEE, and ZKML are clarified.

2 Background: Deterministic Logical Timeline

2.1 DSF Triple and PSSM Evolution

The deterministic logical timeline is built on the Dynamic Security Foundation (DSF) triple:

$$DSF = (K_{sec}, Init_Anchor, Rule_ID)$$

The Protocol Security State Machine (PSSM) evolves according to the following formulas:

$$S(0) = H(K_{sec} || Init_Anchor || "INIT")$$

$$K_e = H(K_{sec} || e || "SUBKEY"), \quad e = \lfloor n / N_{epoch} \rfloor$$

$$S(n+1) = H(K_e || S(n) || "CHAIN_TAG")$$

where H is a cryptographic hash function and N_{epoch} is the number of logical ticks per epoch.

$S(n)$ has the following properties: unforgeability (cannot be predicted without K_{sec}), independent verifiability (can be recomputed by anyone holding the DSF), tamper-evidence (hash chain breaks on modification), forward security, and backward security. The detailed mechanisms of this logical timeline have been disclosed in Paper 1, “Cryptography-Driven Deterministic Logical Timeline”[7].

2.2 Relationship between Logical State and Physical Anchor

In this method, the deterministic logical timeline and the local physical clock form two independent but related layers:

- **Logical layer (independent evolution):** the generation of the logical state $S(n)$ is entirely determined by the DSF-driven PSSM and is independent of the precision, drift, and error of the physical clock;

- **Physical layer (instantaneous sampling):** at logical moment n , the node instantaneously reads the local physical clock to obtain the physical anchor $T_{anchor}(n)$, which is used to determine the physical execution moment of the token generation behavior but does not participate in the computation of $S(n)$.

The errors of the two layers do not affect each other: a single sampling error of the physical clock only affects the absolute value of $T_{anchor}(n)$, does not propagate to $S(n)$, and does not accumulate.

3 Token Generation Control Method

3.1 Core Idea: Entropy Source Replacement and Path Locking

The core idea of this method consists of two aspects:

First aspect—entropy source replacement: the randomness of existing large-model token sampling comes from the PRNG seed, which is usually derived from system time, hardware noise, or a user-specified value, belonging to a physical-layer non-deterministic source. This method replaces the PRNG seed with a cryptographically driven deterministic logical state $S(n)$ —a state generated by a deterministic logical timeline decoupled from physical clocks, independent of any hardware state, operating system state, or framework version.

Second aspect—path locking: after $S(n)$ is used as the initialization seed of the PRNG, the random number sequence $u_1, u_2, \dots, u_k$ produced by the PRNG is completely determined. Since the token sampling algorithm is a deterministic function $Token = F(\text{probability distribution, random number})$, as long as $S(n)$ is fixed, the probabilistic walking path of token sampling is permanently locked—the model’s native probability distribution remains unchanged, but the result of “rolling the dice” is cryptographically anchored.

This method does not change the model’s ability to “choose what is smarter” (the probability distribution is unchanged); it fixes the execution path of “which token to draw from the probability distribution” (the PRNG seed is controlled by $S(n)$).

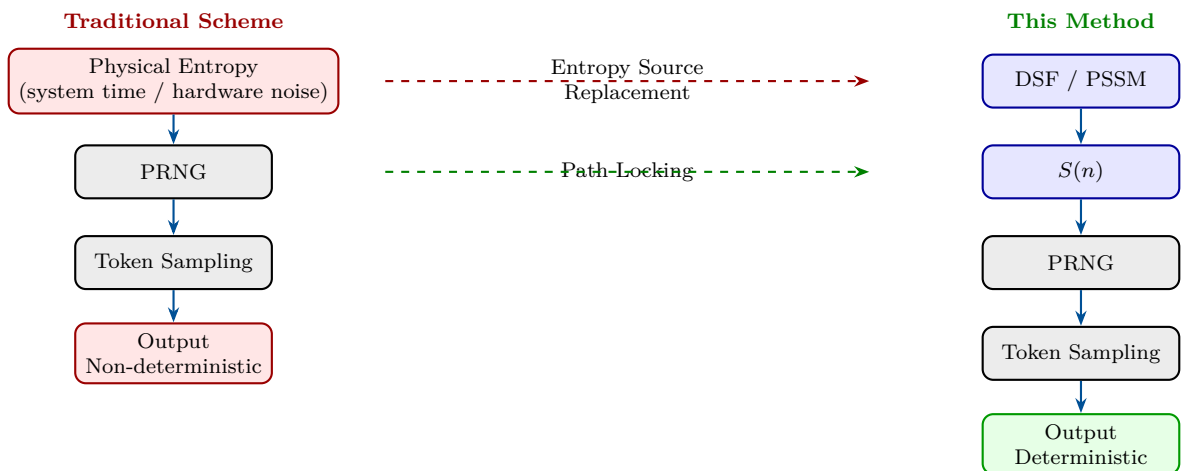


Figure 1: Token generation control architecture and entropy source replacement. Left: traditional scheme. Right: this method.

As shown in Figure 1, this method replaces the PRNG seed from a physical entropy source with the cryptographically driven $S(n)$, thereby locking the token sampling path and changing the output from “non-deterministic” to “deterministic.”

3.2 Token Generation Flow

1. **Build the deterministic logical timeline:** jointly constructed by the DSF and PSSM; the logical counter n is driven by token generation events, periodic ticks, inference frames, or user-defined events;
2. **Output the deterministic logical state:** at logical moment n , the PSSM outputs $S(n)$, whose generation does not depend on any physical-layer random source;
3. **Use $S(n)$ as the PRNG seed:** use $S(n)$ as the initialization seed of the PRNG in the token sampling process, replacing random sources based on physical time or hardware noise;
4. **Execute token sampling:** based on the probability distribution P and the random number sequence output by the PRNG, determine the selected token using direct sampling, Gumbel-Max, Top-K, or Top-p algorithms;
5. **Cryptographic binding and record generation:** cryptographically bind the generated token sequence to $S(n)$ and n , generating a verifiable generation record;
6. **Behavior chain linking:** generation records at each logical moment are linked via a hash chain, forming a tamper-evident behavior chain.

3.3 Conversion of $S(n)$ into a PRNG Seed

The conversion method for using $S(n)$ as the PRNG initialization seed depends on the type of $S(n)$:

- **Hash chain value (Rule B):** if $S(n)$ is a 256-bit hash value, it can be directly used as the PRNG seed input (e.g., Python `random.seed(S(n).to_bytes(32, 'big'))` or PyTorch `torch.manual_seed(int.from_bytes(S(n), 'big'))`);
- **Logical counter value (Rule C):** if $S(n)$ is a logical counter value, it is concatenated into a byte string and hashed for normalization before being used as the PRNG seed;
- **Mixed with user random entropy:** for a user-specified external random entropy $User_R$, it is concatenated with $S(n)$ and hashed as a whole, and the hash value is used as the PRNG seed.

The PRNG generates a random number sequence $u_1, u_2, \dots, u_k$ based on $S(n)$, where k equals the total number of tokens to be generated in this inference. Each token position consumes one or more random numbers u_i , depending on the sampling algorithm: direct sampling and Top-K/Top-p sampling consume one random number per token, while Gumbel-Max sampling consumes a number of random numbers equal to the candidate set size at that position.

3.4 Token Sampling Algorithms

At logical moment n , the token sampling process is:

1. The large model computes the probability distribution $P(token \mid context)$ of the next token based on the input context;
2. The sampling module determines the selected token based on the probability distribution P and the random number sequence output by the PRNG, using at least one of the following sampling algorithms:
 - Direct sampling: $token_i = Sample(P, u_i)$;

- Gumbel-Max sampling: $token_i = \arg \max(\log(P) + G(u_i))$, where $G(u_i)$ is Gumbel noise generated from the random number u_i ;
- Top-K sampling: truncate P to obtain the K tokens with the highest probabilities as the candidate subset, then sample from the subset using u_i ;
- Top-p sampling (nucleus sampling): truncate P to obtain the smallest token set whose cumulative probability does not exceed p as the candidate subset, then sample from the subset using u_i .

3. Repeat the above process until the complete token sequence is generated.

Since the random number sequence of the PRNG is completely determined by $S(n)$, the token generation path of any of the above sampling algorithms is deterministic at the logical level.

3.5 Cryptographic Binding and Record Generation

The generated token sequence is cryptographically bound to $S(n)$ and n , generating a verifiable generation record $Record(n)$:

$$Record(n) = \{n, S(n), Hash(S(n)), Action_Type, Input_Hash, Token_Hash, Model_Version, Auth_Tag\}$$

where:

- n : logical moment identifier, determining the position of the generation behavior in the logical sequence;
- $S(n)$: logical state value (plaintext), used by the verifier to initialize the PRNG to replay the token sampling process;
- $Hash(S(n))$: hash of the logical state, used to verify the binding between $S(n)$ and logical moment n ;
- $Action_Type$: behavior type identifier, marked as "TOKEN_GEN";
- $Input_Hash$: hash of the input data (prompt/context), used for logical replay;
- $Token_Hash$: hash of the output token sequence, used for logical replay;
- $Model_Version$: model version number, ensuring that the same model version is used during replay;
- $Auth_Tag$: authentication tag, used for tamper-evidence verification.

3.6 Two Working Modes

Mode 1: Deterministic Mode

$S(n)$ is the sole entropy source of the PRNG, and no external randomness is introduced. Same input + same $S(n) \rightarrow$ exactly the same token sequence. Applicable scenarios: AI agents (code generation, automated trading, industrial control), mathematical problem solving, and any scenario requiring stable output.

Mode 2: Divergent-but-Reproducible Mode

$S(n)$ and a user-specified external random entropy $User_R$ are jointly used as the PRNG entropy source. $User_R$ is recorded together with $S(n)$ in the generation record. Same input + same $(S(n), User_R) \rightarrow$ exactly the same token sequence. Applicable scenarios: literary

creation, marketing copy, artistic drawing—users can reproduce a specific output that “struck a chord.”

Dimension	Deterministic Mode	Divergent-but-Reproducible Mode
Entropy source	$S(n)$ only	$S(n) + User_R$
Result for same input	Exactly the same	May differ (depending on $User_R$)
Reproducibility	Fully reproducible	Reproducible given $(S(n), User_R)$
Applicable scenarios	Agents, trading, industrial control	Creation, marketing, art

3.7 Pause and Resume

The token sampling process can be paused at any logical moment n . When paused, the system saves the current logical moment identifier n and the corresponding logical state $S(n)$, and the advancement of the logical timeline is also paused. When resumed, token sampling restarts from the saved $S(n)$ and n , and the subsequently generated token sequence is exactly the same as if there had been no interruption.

This feature is particularly important for long-text generation, streaming output, and break-point resume scenarios: even if the generation process is interrupted, the output after resumption is exactly the same as if there had been no interruption, and no non-determinism is introduced by the interruption.

Implementation mechanism:

1. **Pause:** upon receiving a pause command, the system records the current logical moment n_{pause} and the corresponding logical state $S(n_{pause})$, and stops PSSM evolution and token sampling;
2. **State saving:** n_{pause} and $S(n_{pause})$ are persistently stored, optionally protected by encryption;
3. **Resume:** upon receiving a resume command, the system reads n_{pause} and $S(n_{pause})$, continues running the PSSM from that state, and executes token sampling;
4. **Consistency guarantee:** since the PSSM is a deterministic state machine, after resuming from the same $S(n_{pause})$ and n_{pause} , the subsequently generated $S(n)$ sequence is exactly the same as if there had been no interruption, and therefore the token sampling path is also exactly the same.

Relationship with the behavior chain: pause and resume events can be written as behavior records into the behavior chain, marked as $Action_Type = "PAUSE"$ and $Action_Type = "RESUME"$. The auditor can verify the temporal legitimacy of pause and resume through these records.

4 Independent Verification and Replay

4.1 Source of the Verification and Replay Mechanism

The authenticity verification of token generation records adopts the independent verification and replay mechanism disclosed in the applicant’s prior patent (Application No. 2026110706945,

“System and Method for Deterministic Behavior Recording, Auditing, and Replay Based on Logical-Physical Dual Anchoring”). This mechanism has been systematically described in Paper 1, “Cryptography-Driven Deterministic Logical Timeline”[7].

It should be clarified that the token generation patent (Application No. 2026111239235) focuses on **entropy source replacement and path locking** (i.e., how to make the token sampling path deterministic); the verification and replay patent (Application No. 2026110706945) focuses on **independent verification of behavior records** (i.e., how to verify the authenticity of generation records). The two are complementary and together form a complete AI output determinism system. This paper applies the verification mechanism to token generation records, and the verification flow is as follows.

4.2 Verification Flow

Any verifier holding the same initial state and evolution rules can independently verify the authenticity of a token generation record at any time:

Step 1: Logical state recomputation and binding verification

Independently evolve to the target logical moment n , compute $S(n)$, and verify that it is consistent with $Hash(S(n))$ in $Record(n)$ —confirming that the binding between $S(n)$ and logical moment n is authentic and valid.

Step 2: Path replay

Extract the plaintext $S(n)$ from $Record(n)$, use it to initialize the PRNG, and replay the token sampling process using the same model version and input data. Verify that the hash of the output token sequence is consistent with $Token_Hash$ in $Record(n)$ —confirming the correctness of the token generation path.

Step 3: Chain integrity verification

Verify that all hash links from the initial anchor to the target moment are continuous—confirming that the behavior history is intact and untampered.

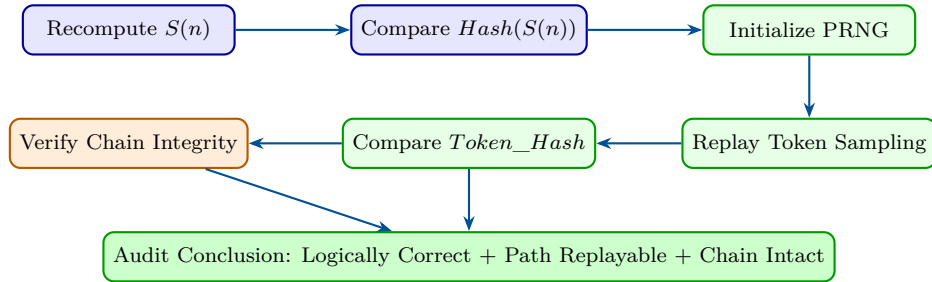


Figure 2: Independent verification and replay flow. The verifier only needs to hold the same DSF to complete all steps. This mechanism originates from Patent 2026110706945.

The verification flow is shown in Figure 2. This flow completely reuses the behavior record auditing mechanism described in Paper 1, with the only difference being that the behavior being audited is token generation.

4.3 Audit Conclusion Example

“At logical moment n (corresponding to physical moment $T_{anchor}(n)$), the AI model, under model version $Model_Version$, used $S(n)$ as the PRNG seed to perform inference on input $Input_Hash$, and the hash of the generated token sequence is $Token_Hash$. Logical state recomputation verification passed, path replay verification passed, and the behavior chain is intact and untampered.”

5 Essential Differences from Existing Schemes

5.1 Difference from Random Seed Setting

Dimension	Random Seed Setting	This Method
Seed source	User-specified or system time	Cryptographically driven $S(n)$
Cross-environment consistency	Not guaranteed	Cryptographically guaranteed
Seed source verifiable	No	Yes (independently recomputable from DSF)
Cryptographic binding	None	Yes ($S(n)$ bound to n)
Auditing and replay	Not supported	Supported

5.2 Difference from Batch-Invariant Kernels

Dimension	Batch-Invariant Kernels	This Method
Layer addressed	Computational layer (operator execution order)	Entropy source layer (PRNG seed)
Entropy source control	Not involved	$S(n)$ as PRNG seed
Cryptographic binding	None	Yes
Cross-environment consistency	Limited (within the same environment)	Cryptographically guaranteed
Auditing and replay	Not supported	Supported

Batch-invariant kernels are proposed by Thinking Machines Lab and belong to existing technology. Their core is to eliminate differences in floating-point reduction order caused by changes in batch size. This method further locks the token sampling path from the entropy source layer. The two can be combined: batch-invariant kernels ensure determinism at the computational layer, and this method ensures determinism at the sampling layer. Combined, they achieve end-to-end cross-environment determinism.

5.3 Difference from Inference Receipts

Dimension	Inference Receipts	This Method
Timing	After inference completes	Anchored before inference starts
Anchoring object	Record write moment	Inference initiation moment
Entropy source control	Not involved	$S(n)$ as PRNG seed
Cross-environment consistency	Explicitly not guaranteed	Cryptographically guaranteed
Security model	Honest emitter assumption	Cryptographic enforcement
Independent verification	Requires external ledger	Hold DSF

The two schemes complement each other: one manages the process, the other manages the result. Together they form a complete AI output determinism system. Inference Receipts address “post-hoc verifiability,” while this method addresses “pre-hoc determinism.”

5.4 Difference from TEE/ZKML

TEE relies on a trust basis from hardware vendors, cross-domain verification is difficult, and time binding uses physical timestamps. ZKML has extremely high computational overhead (1000–10000 times that of inference) and does not provide time binding. This method is based on cryptographic rules, natively supports cross-domain verification, uses a logical timeline plus physical anchor for time binding, and has low computational overhead (hashing + HMAC).

6 Application Prospects

The token generation control method can serve as a core technology for the following scenarios:

Application Domain	Core Value
AI Agent deterministic behavior control	Same input necessarily produces same output, reproducible across environments
Financial trading compliance auditing	Each trading decision independently verifiable, regulator can trace
Code generation	Same requirement produces same code, supporting version management and regression testing
Industrial control	Deterministic generation of control commands, incidents traceable
AI output determinism	Inference starting point anchoring, supporting version management and retrieval reuse

7 Conclusion

This paper proposes a token generation control method based on a deterministic logical timeline. The method uses the logical state $S(n)$ output by a Protocol Security State Machine as the initialization seed of the PRNG, transforming the token sampling process from “physical non-determinism” to “logical determinism,” and achieving cryptographic-level determinism across environments, hardware, and frameworks.

The core contributions of this paper are:

1. **Entropy source replacement:** replacing the PRNG seed from a physical-layer non-deterministic source with a cryptographically driven deterministic logical state $S(n)$;
2. **Path locking:** the token sampling path is cryptographically anchored, while the model’s native probability distribution remains unchanged;
3. **Two working modes:** deterministic mode and divergent-but-reproducible mode, satisfying the needs of both AI agents and creative scenarios;
4. **Cryptographic binding and auditing:** the token sequence is cryptographically bound to $S(n)$, and the verifier can independently replay using the verification and replay mechanism disclosed in the prior patent;

5. **Boundary delineation:** systematic comparison of this method with random seed setting, batch-invariant kernels, Inference Receipts, TEE, and ZKML.

Do not change the model’s intelligence; only lock the sampling path. Let the same input produce the same output at any time, in any environment, on any hardware.

7.1 Known Limitations and Future Work

Known limitations:

Limitation	Description
Computational-layer non-determinism	This method controls the PRNG seed at the sampling layer; non-determinism in the logits computation stage before sampling (CUDA kernel execution order, floating-point accumulation order, etc.) is not addressed by this method and requires combination with schemes such as batch-invariant kernels
Model version consistency	Replay verification requires the same model version; after a model upgrade, old records can still be verified, but newly generated token sequences may differ from old records
Storage of user random entropy	In the divergent-but-reproducible mode, <i>User_R</i> must be stored with the record; if <i>User_R</i> is lost, the output cannot be reproduced
Experimental scope	The verification in this paper is internal testing and has not been independently reproduced by a third party

Future work:

1. Combine with batch-invariant kernels to achieve end-to-end cross-environment determinism;
2. Extend to multi-modal generation (image, audio, video) determinism control;
3. Study the application of token generation determinism in federated learning and distributed inference scenarios;
4. Invite third parties to independently reproduce the experimental data in this paper.

AI Usage Statement

The LaTeX formatting, layout, and language polishing of this paper were assisted by AI. The research design, technical approach, core mechanisms, experimental validation, and conclusions were completed by the authors. AI did not participate in the formulation of research questions, selection of technical routes, generation of experimental data, or interpretation of results.

Shanghai Huapait Technology Co., Ltd.
October 2026

References

- [1] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.
- [2] Thinking Machines Lab, “Batch-invariant kernels for deterministic LLM inference,” 2026.
- [3] Model Witness: Cryptographic attestation for AI inference, 2026.
- [4] AIIR: AI Integrity Receipts, 2026.
- [5] V. Costan and S. Devadas, “Intel SGX explained,” *IACR Cryptology ePrint Archive*, vol. 2016, p. 086, 2016.
- [6] T. Liu, Y. Xie, and J. Zhang, “ZKML: A survey on zero-knowledge machine learning,” *arXiv preprint arXiv:2403.10326*, 2024.
- [7] Qiu Yong, Wang Zhehao, “Cryptography-Driven Deterministic Logical Timeline: A Temporal Baseline for Machine Collaboration and AI Auditing,” AiraXiv, ID: 2610.0018, 2026.
- [8] Qiu Yong, Wang Zhehao, “Cryptography-Driven Deterministic Logical Timeline: AI Behavior Records: Dual-Timestamp Binding and Trigger Modes,” AiraXiv, ID: 2610.00xx, 2026.