Giugno 2026

Sicurezza e performance nel path MTP

Quando accelerare un LLM diventa anche un problema di affidabilità

AGID | Agenzia per l'Italia Digitale

CERT-AGID

Sicurezza e performance nel path MTP

Quando accelerare un LLM diventa anche un problema di affidabilità

Giugno 2026

Abstract

Il **Multi-Token Prediction (MTP)**[1] è una tecnica recente in cui un modello viene addestrato o configurato per prevedere più token futuri contemporaneamente, anziché uno solo alla volta. In fase di inferenza, l'MTP viene spesso abbinato al **decoding speculativo**[2]: un modello secondario più leggero (*draft*) propone una sequenza di token futuri e il modello principale (*target*) li convalida in un unico passaggio, velocizzando il calcolo. Questa distinzione è fondamentale: l'MTP definisce la capacità di predire più token, mentre il decoding speculativo definisce il meccanismo di verifica e accettazione.

Questo paper analizza l'intersezione tra questi due concetti nei runtime locali. Sebbene l'argomento sembri legato alle sole prestazioni, l'analisi evidenzia implicazioni critiche per la **sicurezza operativa**. L'introduzione di questa tecnica influisce infatti sulla coerenza dello stato interno, sulla prevedibilità dei tempi di risposta, sulla disponibilità del servizio, sui meccanismi di fallback e replay, e sul comportamento del sistema nei casi peggiori. La tesi sostenuta è che una tecnica di accelerazione può definirsi sicura solo se migliora i tempi di calcolo senza rendere il comportamento del sistema opaco, instabile o difficile da verificare.

Introduzione

Immaginiamo una persona che sta scrivendo una frase e ha appena digitato: *Il processore usa la cache per...*". Accanto a lei c'è un assistente veloce che prova ad anticipare le parole successive proponendo: *'ridurre i tempi di accesso'*. La persona controlla. Se la proposta è corretta, la accetta in blocco. Se è corretta solo in parte, tiene la parte buona e corregge il resto. Se è del tutto sbagliata, la rifiuta.

Questa analogia descrive l'unione tra MTP e decoding speculativo. Il modello draft propone più token futuri, mentre il modello target verifica: il draft propone, ma il target decide. Il punto fondamentale per la sicurezza è che **il draft non deve mai diventare l'autorità finale**. Il suo

[1] Fast and Expressive Multi-Byte Prediction with Probabilistic Circuits https://arxiv.org/pdf/2511.11346
[2] Speculative Speculative Decoding https://arxiv.org/pdf/2603.03251

compito è accelerare il processo, non modificare in modo opaco le decisioni del modello principale.

Di conseguenza, l'MTP non è solo una questione di prestazioni, ma di sicurezza operativa. Dal momento che questa tecnica entra direttamente nel percorso (*path*) di generazione dei token, **influisce su proprietà sistemiche fondamentali**: l'integrità dell'output, la disponibilità, la prevedibilità dei tempi di risposta, la robustezza nei casi peggiori e la verificabilità del processo decisionale. Nei runtime locali, specialmente in ambienti consumer, prestazioni e sicurezza convergono inevitabilmente sullo stesso fattore critico: il controllo rigoroso dello stato interno del sistema.

Perché MTP può accelerare

Nel decoding classico, il modello principale genera un token alla volta (una parola, una parte di essa o un simbolo). Il processo è strettamente sequenziale: si predice un token, si aggiorna lo stato e si passa al token successivo. È un metodo robusto, ma vincolato dai tempi di calcolo di ogni singolo passaggio seriale.

L'MTP, abbinato al decoding speculativo, rompe questa sequenza: il modello draft propone un blocco di più token futuri e il target li verifica in un'unica operazione. Se la proposta è valida, il sistema avanza di più token contemporaneamente. Durante l'inferenza si possono verificare tre scenari:

- **Full accept (Caso migliore):** Il draft propone tre token e il target li accetta tutti. Il sistema avanza di tre passi in un colpo solo.
- **Full reject (Caso peggiore):** Il draft propone token che il target rifiuta immediatamente. Il sistema non guadagna tempo e deve rigenerare il token corretto.
- **Partial accept (Caso critico):** Il target accetta solo una parte della proposta.

In questo scenario parziale il sistema deve integrare la parte corretta, scartare quella errata e riallineare lo stato interno per ripartire dal punto esatto. È precisamente in questa gestione del riallineamento che emergono le complessità tecniche e i rischi per la sicurezza operativa.

Il punto critico: la coerenza dello stato interno

Un LLM non genera testo basandosi solo sulla stringa visibile. Durante la generazione mantiene e aggiorna costantemente uno stato interno composto da due elementi chiave:

- **Il frontier logico:** Rappresenta il punto esatto fino a cui è arrivata la generazione ufficiale. È la porzione di testo considerata definitiva e consolidata.
- **La KV cache:** È la memoria interna del Transformer che memorizza le chiavi e i valori dei token passati per evitare di ricalcolare l'intero contesto a ogni nuovo passaggio. Funziona come un segnalibro: permette al modello di riprendere la lettura dal punto esatto in cui si era fermato, senza dover rileggere il libro dall'inizio.

Nel percorso MTP la gestione di questo stato si complica drasticamente, perché è necessario mantenere coerenti e sincronizzati sia lo stato del **modello target** che quello del **modello draft**. Durante la validazione speculativa, il frontier logico può trovarsi temporaneamente più avanti rispetto allo stato consolidato della KV cache, in questo caso, le parole suggerite dall'assistente sono già scritte sul foglio, ma l'autore deve ancora verificarle.

Questa asincronia non è un problema se il runtime tratta quelle parole rigorosamente come candidate. Il rischio concreto per la sicurezza emerge se il runtime confonde i token proposti con quelli già verificati. In caso di disallineamento, il sistema può validare i token nella posizione sbagliata, corrompere la KV cache con dati incoerenti o essere costretto a ricalcolare passaggi già effettuati.

Le tre dimensioni della sicurezza operativa

Nelle architetture MTP, la sicurezza non si limita al blocco di contenuti dannosi (moderazione), ma coincide con l'affidabilità, la prevedibilità e la verificabilità dell'intero sistema.

Integrità del comportamento

In un sistema ideale, attivare MTP dovrebbe cambiare solo una cosa: la velocità. Il modello dovrebbe arrivare alla stessa risposta, ma in meno tempo.

Nella pratica, però, i modelli non sempre scelgono la continuazione più probabile. Quando è attivo il sampling, il modello sceglie il prossimo token tra più candidati possibili, seguendo una distribuzione di probabilità. Questo significa che, anche con lo stesso prompt, due generazioni possono prendere strade leggermente diverse.

Questa variazione non è automaticamente un bug.

È accettabile se il modello ha semplicemente scelto una continuazione diversa tra quelle possibili.

È come una persona che completa la frase "*Il processore usa la cache per...*" con "*migliorare le prestazioni*" oppure con "*ridurre i tempi di accesso*".

Entrambe le frasi possono essere valide. Il percorso cambia, ma la scelta è coerente con il comportamento del modello.

Il caso critico è diverso. Il problema nasce quando l'output cambia non perché il modello ha scelto una strada alternativa, ma perché il runtime ha perso il controllo del proprio stato interno.

È come se l'assistente suggerisse alcune parole, la persona ne accettasse solo metà, ma il foglio, il segnalibro e la memoria dell'assistente non venissero aggiornati nello stesso modo. A quel punto la frase può continuare da un punto sbagliato, non per una scelta consapevole, ma per un disallineamento.

Nel runtime questo può succedere quando non restano coerenti elementi come:

- KV cache;
- sampler;
- frontier logico;
- token accettati e rifiutati.

Questa è una violazione dell'integrità del comportamento.

Disponibilità del servizio

La disponibilità riguarda una domanda molto semplice: il sistema riesce a rispondere in modo stabile quando serve?

Un percorso MTP instabile può mettere a rischio questa proprietà. Non perché il modello smetta necessariamente di funzionare, ma perché può iniziare a impiegare molto più tempo del previsto.

Riprendiamo l'immagine della persona che scrive con un assistente accanto.

Se l'assistente suggerisce frasi utili, la scrittura accelera. Ma se continua a suggerire parole sbagliate, la persona deve fermarsi, cancellare, correggere e ripartire. Il testo finale può anche essere corretto, ma il processo diventa lento e faticoso.

Nel runtime succede qualcosa di simile. Quando il path MTP non riesce a mantenere lo stato interno coerente, entrano in gioco meccanismi di recupero.

Il primo è il replay. Significa che il sistema deve rifare una parte del lavoro già svolto. È come tornare indietro di qualche riga perché il segnalibro non indica più il punto giusto.

Il secondo è il fallback. Significa che il runtime abbandona il percorso più veloce e torna a un metodo più conservativo, di solito il decoding classico. È come dire all'assistente: "*fermati, da qui continuo da solo*".

Replay e fallback non sono necessariamente errori. In alcuni casi sono meccanismi utili, perché permettono al sistema di recuperare da uno stato incerto. Il problema nasce quando diventano frequenti.

Se il sistema deve continuamente tornare indietro o abbandonare il percorso veloce, la disponibilità peggiora. Le risposte possono diventare lente, irregolari o arrivare in timeout. Nei casi peggiori possono comparire assert, cioè controlli interni che falliscono perché il programma trova uno stato che non dovrebbe esistere.

La disponibilità non dipende solo dal fatto che il modello sappia rispondere. Dipende anche dal fatto che il runtime riesca a farlo entro tempi controllabili, senza entrare continuamente in percorsi di recupero.

Prevedibilità dei costi computazionali e DoS applicativo

Un sistema affidabile non deve solo produrre una risposta corretta. Deve anche farlo con un costo ragionevolmente prevedibile. Nel path MTP questo aspetto è importante, perché il costo dipende molto da quanto il draft riesce ad anticipare bene il target.

È come usare un navigatore durante un viaggio. Se il navigatore suggerisce strade corrette, si arriva prima. Se invece continua a proporre svolte sbagliate, bisogna fermarsi, tornare indietro, ricalcolare il percorso e ripartire. La destinazione può essere ancora raggiunta, ma il viaggio diventa molto più lungo e meno prevedibile.

Nel runtime succede qualcosa di simile. Un prompt difficile può portare il draft a sbagliare più spesso. Questo può causare più rifiuti, più partial accept, più riallineamenti dello stato interno e, in alcuni casi, più replay. Il rischio non è necessariamente un crash. Il modello può continuare a funzionare. Il problema è che il costo computazionale cresce molto rispetto al caso normale.

Se un input viene costruito apposta per spingere il sistema verso questi casi costosi, si può creare una forma di **Denial of Service** applicativo. Non è un exploit classico basato su corruzione della memoria. È più sottile: il sistema resta formalmente funzionante, ma viene spinto a fare molto più lavoro.

Per questo, quando si valuta MTP, bisogna misurare anche i casi difficili. Non basta sapere che il sistema è veloce quando il draft indovina. Bisogna sapere cosa succede quando il draft sbaglia spesso. Ovvero:

- quanti token vengono rifiutati?
- quanti partial accept si verificano?
- quanti replay vengono attivati?
- quanto cresce la latenza nei casi peggiori?
- il costo resta entro limiti accettabili?

Se il costo computazionale resta controllato anche nei prompt difficili, MTP è più sicuro da usare. Se invece alcuni input fanno esplodere i tempi, allora l'ottimizzazione introduce una superficie di rischio.

Policy del draft: Perché essere prevedibili è una scelta di sicurezza

Il draft non deve generare il testo finale, ma formulare ipotesi che il modello target dovrà convalidare. In questo contesto, la proposta più prevedibile è la più utile, la più efficiente e anche la più sicura.

Immaginiamo che il sistema debba analizzare un comando sospetto:

powershell -enc ...

Una continuazione prudente potrebbe essere:

questo comando è sospetto perché usa una stringa codificata; va analizzato in un ambiente sicuro

Questa è una risposta da triage. Segnala il rischio, invita alla cautela e resta dentro un contesto difensivo.

Una continuazione più libera potrebbe invece essere:

per capire cosa fa, si può decodificare la stringa e poi eseguire il risultato in PowerShell

Qui il problema non è che la frase sia tecnicamente impossibile. Il problema è la direzione che prende. La risposta passa dall'analisi prudente a un suggerimento operativo che, se seguito fuori da un ambiente isolato, può essere pericoloso.

Nel path MTP, se il draft esplora continuazioni troppo libere, aumenta la probabilità di spingere il target a intraprendere traiettorie più delicate da controllare.

Una policy *top1-greedy* riduce questa variabilità. Il draft sceglie la continuazione più probabile, non quella più originale. Questo non rende automaticamente il sistema più sicuro, ma rende il comportamento più stabile, più misurabile e più facile da verificare.

Metodo sperimentale: misurare la coda e l’effetto barriera

Per valutare l'MTP con rigore scientifico è obbligatorio misurare in parallelo sia le prestazioni sia la sicurezza operativa. Quando si analizzano i tempi, fermarsi alla velocità media è un grave errore metodologico: le prestazioni reali si misurano analizzando la latenza tipica rispetto a quella dei casi peggiori, utilizzando i percentili (p75 e p95).

Mentre la mediana (p50) mostra l'andamento del caso normale, il p75 indica dove le condizioni iniziano a degradare e il p95 traccia il comportamento quasi nel caso peggiore. Una modifica al codice può migliorare la mediana ma peggiorare drasticamente il p95. In laboratorio, guardando la media, l'ottimizzazione sembra efficace; in produzione, un p95 instabile introduce picchi di latenza inaccettabili che distruggono l'affidabilità del servizio.

Nelle architetture locali CPU-only si aggiunge la complessità hardware della gestione dei thread e del rischio dell'**effetto barriera**. Immaginiamo sei persone che devono sollevare insieme un carico pesante: se cinque sono pronte ma una è in ritardo, le prime cinque rimangono ferme ad aspettare. Nei calcoli paralleli accade lo stesso: i thread che completano la propria elaborazione in anticipo si bloccano in corrispondenza di una *barrier* (punto di sincronizzazione obbligatorio) in attesa che l'ultimo thread termini il suo lavoro.

Per questa ragione, nei benchmark su macchine non performanti, è fondamentale impostare rigorosamente l'affinità di CPU attraverso strumenti come ***taskset***, vincolando i thread a core fisici specifici per impedire lo spostamento arbitrario dei processi da parte del sistema operativo. Senza questo rigore, si rischia di attribuire all'architettura MTP un'instabilità o una variazione di performance che dipendono unicamente dalle decisioni dello scheduler della CPU.

Un framework di test completo deve monitorare contemporaneamente entrambe le dimensioni, garantendo la ripetibilità dei parametri hardware e algoritmici (thread, batch, ubatch, prompt, modelli e filtri del sampler):

Metriche di Performance	Indicatori di Sicurezza Operativa
Tempo totale di generazione: Latenza complessiva del processo.	**Integrità dell'output:** Verifica dell'equivalenza dei testi o tracciabilità logica della divergenza.
Frequenza di chiamata: Numero di invocazioni atomiche verso target e draft.	**Stabilità del runtime:** Assenza totale di crash, blocchi o fallimenti di *assert*.
Efficienza predittiva: Conteggio esatto dei token accettati e rifiutati (*acceptance rate*).	**Meccanismi di recupero:** Monitoraggio della frequenza degli eventi di *fallback* e *replay*.
Analisi della coda: Calcolo dei percentili p75 e p95.	**Coerenza degli stati:** Sincronizzazione della KV cache, del sampler e del frontier logico.
Costo nei prompt difficili: variazione della latenza nei casi peggiori.	**Stress testing:** Comportamento in presenza di prompt "difficili" (*worst-case scenario*).

Una singola esecuzione non basta. Le misurazioni vanno ripetute più volte, alternando sistematicamente baseline e variante, valutandone l'impatto sulla latenza di coda.

Risultati osservati in laboratorio

Per verificare il riscontro pratico di queste considerazioni, abbiamo analizzato un percorso MTP sperimentale in ambiente CPU-only, vincolando l'esecuzione con: **taskset -c 0-5**.

In questo modo il processo viene limitato alle CPU logiche da 0 a 5, riducendo la possibilità che il sistema operativo sposti i thread da un core all'altro durante il calcolo. Questo vincolo non rende il modello più veloce in sé, ma rende il benchmark più pulito e più ripetibile.

I test hanno mostrato tre risultati principali.

In primo luogo, l'ambiente di esecuzione conta molto. Senza affinità CPU, la latenza era più instabile. Con *taskset*, lo spread del tempo totale di generazione si è ridotto da circa 3448 ms a circa 1716 ms, e il tempo perso alla barrier è sceso di circa 613 ms. Questo conferma che, su CPU-only, una parte della variabilità può dipendere dalla macchina e non dall'MTP.

Come secondo risultato, l'esperimento ha mostrato che un branch sperimentale del path MTP ha migliorato il caso *response medium*, cioè uno scenario con risposta di lunghezza intermedia usato come confronto principale:

Scenario: risposte medie	Baseline	Branch Sperimentale	Delta
Mediana generation time	11320 ms	10628 ms	-692 ms
p75 generation time	11674 ms	10643 ms	-1031 ms

Durante questi run non sono stati osservati *replay*, *fallback* o *assert*. Il miglioramento, quindi, non è stato ottenuto pagando più recuperi o instabilità evidente.

Infine, non tutte le modifiche apparentemente corrette migliorano il sistema. Abbiamo testato una variante in cui il draft usava il token campionato dal sampler invece del candidato *top-1*. Nel caso *response medium*, il tempo è peggiorato da circa 9949 ms a circa 14092 ms, cioè circa +41.6%.

Questo risultato conferma che, nel path MTP, il draft deve essere prevedibile più che creativo. Il suo compito è proporre token facili da verificare dal target, non esplorare traiettorie più varie.

Conclusioni

L'MTP promette di accelerare l'inferenza anticipando una sequenza di token futuri. Ma il suo valore non sta solo nella velocità, bensì anche nella capacità di accelerare senza perdere controllo. Nel path MTP, il draft deve restare un assistente. Deve proporre. anticipare e suggerisce. Il target, però, deve restare l'autorità finale. Se questa gerarchia si rompe,

l'ottimizzazione smette di essere solo una questione di performance e diventa un problema di sicurezza.

In contesti di cybersecurity questo punto è centrale. Un runtime che analizza comandi sospetti, alert, log o codice potenzialmente malevolo non deve essere solo veloce, deve anche essere stabile, verificabile e prevedibile. Una risposta corretta ottenuta attraverso uno stato interno incoerente, replay continui o divergenze non spiegate non è una base solida per un sistema di sicurezza.

L'analisi del path MTP mostra quindi una doppia natura. Da un lato è performance engineering: ridurre i passaggi seriali, migliorare l'uso della KV cache, aumentare l'acceptance rate e stabilizzare p75 e p95. Dall'altro è sicurezza operativa: preservare l'integrità del comportamento, mantenere disponibile il servizio, controllare i costi computazionali e resistere ai casi peggiori.

Il criterio finale di scelta non può essere solo: *è più veloce?* Deve essere anche:

- *è ancora controllato?*
- *è verificabile?*
- *mantiene il target come arbitro finale?*
- *resta stabile con input difficili?*
- *ha costi prevedibili?*

Il punto di equilibrio è questo: veloce, ma non opaco; ottimizzato, ma verificabile; speculativo, ma controllato dal target; efficiente, ma robusto nei casi peggiori.

Performance e sicurezza non sono due capitoli separati, sono due modi di misurare la stessa proprietà: la capacità del sistema di mantenere il controllo del proprio stato interno mentre genera testo.