GRAPH-CONSTRAINED HETEROGENEOUS TREE ALLOCATION FOR WORKLOAD-ADAPTIVE SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding improves language-model serving by verifying draft tokens in parallel, but its benefit is limited when a heterogeneous batch must share a discrete execution budget. Requests differ in draft confidence, acceptance history, and transition cost, whereas captured CUDA GRAPHs impose discrete capacity and topology constraints. Consequently, maximizing accepted tokens or assigning a homogeneous verification width to every request is not equivalent to maximizing end-to-end throughput. We present **GCHTA (Graph-Constrained Heterogeneous Tree Allocation)**, a batch-level controller designed for this setting. Given DFlash proposals and DDTree-style candidates, GCHTA constructs request-local frontiers, performs topology substitution within a valid CUDA GRAPH envelope, and allocates heterogeneous speculative work using transition-aware utility. Furthermore, nested and bounded-union supertrees provide counterfactual acceptance feedback, elegantly separating simulated policy exploration from physically measured replay latency and graph validity. This algorithmic policy operates on an optimized CUDA-Graph substrate featuring packed-ragged verification, GPU-side tree construction, deferred draft-KV materialization, and reversible native execution. Evaluations on Qwen3-Coder and Qwen3-8B demonstrate that GCHTA significantly outperforms DFlash across diverse workloads (GSM8K, HumanEval, MT-Bench). It achieves mean throughput improvements of 26.41% on Qwen3-Coder and 22.59% on Qwen3-8B. Crucially, this system-level acceleration is directly driven by substantially higher average acceptance lengths (τ)—yielding relative τ surges of up to 80.7% on coding tasks—solidly confirming the effectiveness of our constrained heterogeneous allocation.

1 INTRODUCTION

Autoregressive generation exposes a fundamental sequential dependency: the next target token is unavailable until the preceding token has been determined. Modern speculative decoding (Leviathan et al., 2023; Charlie Chen, 2023) mitigates this bottleneck by utilizing a lightweight drafter to propose a block of tokens, which the target model then verifies in parallel without changing the target distribution. However, the end-to-end serving benefit of this paradigm depends holistically on the cost of producing, verifying, and committing a block, not merely on the number of tokens that are eventually accepted.

For a decoding round that commits τ tokens, we characterize the per-token round cost as:

$$L = \frac{T_{\text{draft}} + T_{\text{verify}} + T_{\text{commit}} + T_{\text{KV}} + T_{\text{transition}}}{\tau}. \tag{1}$$

Here, T_{draft} is the proposal cost, T_{verify} is the target-model verification cost, and T_{commit} is the overhead of selecting accepted tokens and updating the decoding state. T_{KV} denotes KV-cache materialization or reorganization, while $T_{\text{transition}}$ captures the state and scheduler work caused by changing between speculative and target-only execution. Thus, L measures the actual cost of committed progress rather than acceptance length in isolation. This distinction is essential in batched serving: verification executes on a finite set of captured CUDA GRAPH capacities, different request

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

trees consume varying amounts of packed capacity, and a nominally larger tree can unexpectedly incur a new graph launch or a mode-transition penalty.

Existing methods primarily improve proposal quality, draft cost, or single-request verification (Sun et al., 2026; Gao et al., 2026; Ringel & Romano, 2026; Cheng et al., 2026). They leave open the critical challenge of how heterogeneous requests should share discrete graph capacity under execution transitions and uncertain online measurements.

Requests naturally differ in confidence, context length, and marginal benefit from tree nodes. Assigning a homogeneous width can waste capacity on low-value requests, whereas relying on a continuous token budget ignores the discontinuous cost of changing captured CUDA GRAPHs and the feasibility of topology-specific masks. We therefore seek a jointly executable batch plan that maximizes predicted committed-token throughput, rather than optimizing per-request widths or nominal acceptance rates. GCHTA retains DFlash (Chen et al., 2026) drafting and DDTree-style (Ringel & Romano, 2026) tree construction, but fundamentally shifts the optimization from isolated requests to the entire batch (Figure 1). Historical measurements define request-local candidate frontiers before drafting; the controller selects a feasible graph envelope and orchestration plan, subsequently refining speculative trees using current draft distributions.

The controller operates through three strongly coupled components. First, *graph-envelope topology substitution* allows chain, spine-and-branch, and bushy candidates to compete dynamically when they fit within the same captured CUDA GRAPH envelope. Second, *transition-aware heterogeneous allocation* assigns frontier increments according to a holistic utility—factoring in predicted acceptance, replay reliability, graph slack, stage latency, and the cost of changing execution mode. Finally, *nested and bounded-union supertrees* extract counterfactual acceptance evidence from a physical verification. This minimizes redundant probing while preserving physical replay as the grounded source of latency and graph-validity evidence.

Variable-width CUDA GRAPH capture, packed-ragged verification, GPU-side tree construction, deferred draft-KV materialization, plan validation, and reversible native execution robustly support this policy. These serve as essential execution mechanisms rather than additional allocation principles.

We evaluate GCHTA on Qwen3-Coder-30B and Qwen3-8B using H800 GPUs. Across standard benchmarks (GSM8K, HumanEval, MT-Bench), GCHTA achieves average throughput improvements of 26.41% and 22.59% over DFlash, respectively. This end-to-end acceleration validates our core design: heterogeneous allocation and topology substitution concentrate the budget on high-confidence requests, maximizing acceptance length τ (e.g., surging from 4.05 to 7.32 on HumanEval, an 80.7% increase). Crucially, by packing these dynamically shaped trees strictly within valid CUDA GRAPH envelopes, GCHTA tightly controls the per-token round cost L (Equation 1), ensuring theoretical τ gains robustly translate into physical throughput. Our contributions are summarized as follows:

- We formulate batched speculative decoding as a graph-constrained, heterogeneous allocation problem, moving beyond the traditional paradigm of fixed verification-width choices.
- We develop graph-envelope topology substitution and a transition-aware marginal-utility allocator that jointly select request trees, discrete graph capacity, and speculative/native execution modes.
- We introduce nested and bounded-union supertree feedback, separating counterfactual acceptance evidence from physically measured replay latency and graph validity.

2 RELATED WORK

Speculative decoding accelerates LLM inference via draft-and-verify mechanisms (Leviathan et al., 2023). While early methods focused on lossless verification, recent advances significantly improve draft quality and structural efficiency. Examples include multi-head verification like Medusa (Cai et al.), feature-level drafting like the EAGLE series (Li et al., 2024a;b; Sun et al., 2026), and retrieval-augmented serving like Talon (Gao et al., 2026). GCHTA is complementary to these model-side innovations.

We build upon recent execution primitives but elevate them to the serving-controller level. Specifically, DFLASH generates parallel candidate blocks efficiently (Chen et al., 2026), and DDTREE converts these distributions into compact verification trees under fixed budgets (Ringel & Romano, 2026).

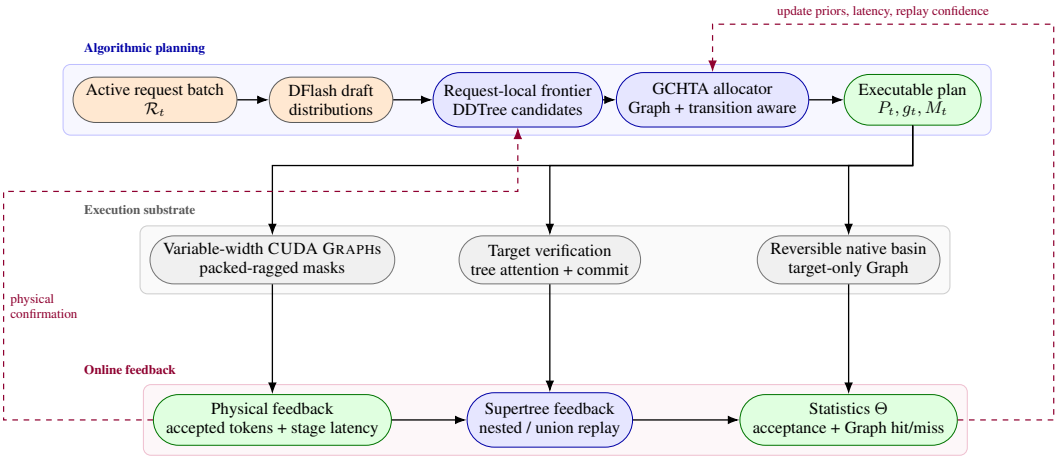


Figure 1: **System architecture of GCHTA.** The online decoding round forms a three-stage closed loop (planning, execution, and feedback). Marked blocks highlight our algorithmic contributions, while unmarked gray blocks represent the underlying execution substrate.

However, both focus on single requests and ignore batch allocation under discrete target-model CUDA GRAPH capacities. GCHTA adopts these as foundational interfaces but solves a broader combinatorial problem: jointly mapping trees across *many* requests into captured CUDA GRAPH envelopes, filling residual capacity, and managing dynamic fallbacks to native decoding. Furthermore, while DSpark uses confidence solely to schedule verification lengths (Cheng et al., 2026), GCHTA extends this principle, utilizing confidence estimators to orchestrate tree topologies, multi-request CUDA GRAPH allocation, and execution-mode transitions.

3 METHOD

Figure 1 summarizes one online decoding round through a three-stage closed loop. Marked blocks highlight our three algorithmic contributions, while unmarked gray blocks represent the execution substrate.

Algorithmic Planning (Contributions 1 & 2): Active requests, DFlash distributions, and historical statistics form request-local frontiers. This formulation, introducing CUDA GRAPH-envelope topology substitution (Contribution 1), exposes heterogeneous width–topology choices while ensuring physical compatibility. The transition-aware allocator (Contribution 2) then jointly evaluates these frontiers to output an executable batch plan, CUDA GRAPH capacity, and speculative/native mode.

Execution Substrate (Unmarked Blocks): The executable plan materializes in the physical layer: speculative requests form target-verification trees under packed-ragged masks, while target-only requests enter the reversible native basin, concluding with KV-state commits. These gray components provide the necessary execution mechanics but are not claimed as independent algorithmic contributions.

Online Feedback (Contribution 3): Physical feedback records accepted tokens and stage latency, keeping the controller grounded. Crucially, counterfactual supertree feedback (Contribution 3) replays structurally covered alternatives (nested or union) to extract acceptance evidence. By strictly isolating inferred algorithmic acceptance from measured physical latency, this mechanism reliably updates the statistics (Θ) to determine the next round's frontiers, gracefully closing the loop.

3.1 PROBLEM SETUP

At decode round t, let $\mathcal{R}_t = \{r_1, \ldots, r_N\}$ denote the active requests and let $M_{t-1} \in \{\text{NATIVE}, \text{SPEC}\}$ be the preceding execution mode. The controller must jointly choose each request's speculative work, the packed verification shape, and whether to remain speculative. This

joint decision is necessary because requests have different confidence and history, while the runtime exposes only finitely many executable Graph variants.

For request r_i, we represent a candidate option as

$$o_{i,j} = (T_{i,j}, w_{i,j}, \tau_{i,j}, \phi_{i,j}),$$

where $T_{i,j}$ is a verification tree, $w_{i,j}$ its packed width, $\tau_{i,j}$ its topology family, and $\phi_{i,j}$ its shape, mask, and Graph signature. The target-only option is $o_{i,0} = (T_{i,0}, 1, \text{target-only}, \phi_{i,0})$. A batch plan $P = (o_{1,j_1}, \ldots, o_{N,j_N})$ therefore specifies one option per active request. Its logical packed size is

$$V(P) = \sum_{i=1}^{N} w_{i,j_i} = N + \sum_{i=1}^{N} (w_{i,j_i} - 1). \tag{2}$$

The logical size is not a continuous compute budget. The runtime replays only captured capacities $\mathcal{C} = \{C_1, \ldots, C_G\}$; a plan is feasible for C_g only when $V(P) \le C_g$ and every option matches its shape, mask, and buffer layout. Thus, similar logical sizes can have different costs when one reuses an envelope and the other crosses a capture boundary or falls back to an eager path.

Let $A_i(o_{i,j})$ be the number of tokens committed for request i in one round and let $\widehat{a}_{i,j}$ denote its predicted mean. We decompose the predicted latency of a feasible plan as

$$\begin{aligned}\widehat{L}(P, g \mid M_{t-1}) = \widehat{L}_{\text{draft}}(P) + \widehat{L}_{\text{target}}(P, g) + \widehat{L}_{\text{commit}}(P) \\ + \widehat{L}_{\text{KV}}(P) + \widehat{L}_{\text{padding}}(P, g) + \widehat{L}_{\text{transition}}(M_{t-1} \to M(P)).\end{aligned} \tag{3}$$

The planning objective is the predicted number of committed tokens per unit of physical execution time:

$$\max_{(P,g)\in\mathcal{Q}_t} \frac{\sum_i \widehat{a}_{i,j_i}}{\widehat{L}(P, g \mid M_{t-1})}, \tag{4}$$

where $\mathcal{Q}_t$ contains plans satisfying the Graph, memory, topology, and mode constraints. The problem is combinatorial; GCHTA therefore constructs request-local frontiers, allocates them under a discrete envelope with transition-aware utility, and updates the planner with physical and structurally valid counterfactual feedback.

3.2 Request-Local Option Frontiers

A fixed-width policy assigns identical speculative work despite request-level differences in draft concentration. Dominant continuations favor deep chains, whereas several plausible alternatives favor branching; width and topology are therefore request-level variables.

For a speculative action, let $\ell_{i,d,v}$ be the drafter's log-probability for candidate v at depth d. The path score of node u is $s_i(u) = \sum_{z\in\text{path}(u)} \ell_{i,d(z),v(z)}$, where $d(z)$ and $v(z)$ are the depth and candidate index of node z. Expanding the highest-scoring paths subject to depth, width, and node limits yields chain, spine-and-branch, or bushy trees from one draft pass.

Candidate nomination from current draft logits uses the surrogate

$$\widehat{e}_i(T) = \sum_{d=1}^{D(T)} \min\left(1, \sum_{u\in T:\, d(u)=d} \exp(s_i(u))\right), \qquad \widehat{a}_i(T) = 1 + \widehat{e}_i(T). \tag{5}$$

Here $D(T)$ is the depth of T and $d(u)$ is the depth of node u. The surrogate ranks continuation mass; it does not replace measured acceptance. Before drafting, the controller uses action-, context-, and cohort-conditioned history; after drafting, it refines the selected tree within its envelope.

The surviving options form a request-local Pareto frontier

$$\mathcal{F}_i = \{o_{i,0}, o_{i,1}, \ldots, o_{i,J_i}\}.$$

An option is removed if another uses no larger envelope, has no smaller predicted acceptance, and is strictly better in at least one dimension. The resulting compact frontier supplies the allocator's increments.

3.3 GRAPH-ENVELOPE TOPOLOGY SUBSTITUTION

Node count alone does not determine runtime interchangeability: ancestor masks, packed layouts, and buffers must also match a captured execution family. We call two trees *Graph-envelope compatible*, written

$$T \equiv_{\text{env}} T',$$

when both can be represented by the same captured capacity family while satisfying its shape, mask, and memory-layout constraints.

Within one envelope, the allocator may replace a chain with a spine-and-branch or bushy option when it has higher predicted acceptance. Compatibility only defines which alternatives can share a physical envelope; replay-specific latency is learned from the executed signature. Topology is therefore a statistical choice subject to a verified physical constraint.

This avoids two width-only failures: equal widths can cover different probability mass, and a better logical tree can force a larger or unavailable Graph. GCHTA evaluates topology value only after checking envelope feasibility.

3.4 TRANSITION-AWARE HETEROGENEOUS ALLOCATION

Given history-derived frontiers $\{\mathcal{F}_i\}_{i=1}^{N}$, the allocator selects one option per request and one captured envelope. Starting from a target-only or incumbent option, an upgrade is a frontier increment Δo. For request i under envelope g, its risk-adjusted utility is

$$\Delta U(i, \Delta o, g) = \frac{\Delta \widehat{a}_{i,\Delta o} - \beta_A \Delta \sigma_{i,\Delta o}}{\Delta \widehat{L}_{i,\Delta o,g} + \beta_L \Delta \sigma^{L}_{i,\Delta o,g}} - \lambda_{\text{switch}} \Delta D(M_{t-1}, M(P)). \tag{6}$$

The numerator is uncertainty-adjusted acceptance gain, the denominator is uncertainty-adjusted physical cost, and the final term is mode-transition debt.

For each envelope, GCHTA applies the highest-utility feasible increment in a max heap and updates only affected utilities and constraints. Requests thus receive different widths and topologies when their marginal returns differ, while sharing one physical envelope.

Slack filling. A second pass fills residual slots when feasible without crossing to a larger Graph. An increment can therefore be profitable inside an already-paid envelope but not when it causes a capture-boundary crossing.

Replay reliability. For each physical signature, the controller records its shape, topology variant, and Graph hit or miss. Eligibility is constrained by a smoothed lower confidence bound on h hits and m misses:

$$\text{LCB}_{\text{hit}}(h, m) \geq \eta_{\text{hit}}. \tag{7}$$

Thus acceptance uncertainty affects utility, whereas replay reliability determines physical admissibility.

Execution-mode selection. Target-only execution is part of the option space: if all selected widths equal one, $M(P) = \text{NATIVE}$. The transition term in equation 3 prevents switching based only on one-round compute, while allowing re-entry into SPEC when its utility again exceeds the native alternative. This is particularly important at high batch sizes, where speculative bookkeeping can outweigh its acceptance benefit.

3.5 COUNTERFACTUAL FEEDBACK FROM SUPERTREES

Independently executing every frontier option is too expensive for online calibration. GCHTA instead uses structural coverage: one physical verification can evaluate several candidates whose nodes and parent relations are embedded in the executed structure.

Nested replay. If candidate T' is a subtree of executed tree T, and $\mathcal{P}(T)$ is the accepted root-to-node path, its replayed accepted length is

$$A(T' \mid T) = 1 + \max\{k : \mathcal{P}_k(T) \subseteq T'\}. \tag{8}$$

The replay follows T''s parent–child relations and stops at the first uncovered edge, rather than merely truncating the covering tree's length.

Bounded union supertrees. For non-nested candidates, a finite subset $\mathcal{S}_i \subseteq \mathcal{F}_i$ is represented by

$$U_i = \bigcup_{T \in \mathcal{S}_i} T, \qquad |U_i| \le C_i, \tag{9}$$

where shared prefixes are represented once and C_i is the request-level budget. Only candidates whose nodes and edges fit atomically are included; the accepted union path is then mapped to each fully covered candidate using equation 8.

Counterfactual replay updates acceptance only; it neither increments the physical replay count nor establishes latency or Graph validity. Those require physical confirmation. Replay is exact for fully embedded candidates under greedy verification and otherwise remains an estimate.

3.6 Online Planning and Execution

Algorithm 1 summarizes one round. History selects a feasible envelope and frontier plan; the drafter then refines speculative trees within that envelope. Shape validation triggers a feasible fallback when necessary. Physical execution updates latency and replay reliability, while covered alternatives update acceptance only.

Algorithm 1 Graph-Constrained Heterogeneous Tree Allocation

Require: active requests $\mathcal{R}_t$, previous mode M_{t-1}, captured CUDA Graph set $\mathcal{G}$, draft-model interface, and statistics Θ
Ensure: executable plan (P_t, g_t, M_t) and updated statistics Θ

```
Build request-local frontiers from historical acceptance and latency.
Enumerate compatible action, topology, mask, and capacity signatures.
for each feasible Graph envelope g ∈ G do
    Initialize P_g and insert feasible increments into a max heap ranked by equation 6.
    while the heap contains a feasible positive-utility increment do
        Assign the best increment and update P_g.
    end while
    Fill residual slack, reject unreliable signatures using equation 7, and score the pair.
end for
Select the highest-scoring feasible pair (P_t, g_t).
if the selected plan contains speculative options then
    Draft, refine selected trees, and construct a feasible supertree.
end if
Validate shape and mask; if invalid, replace (P_t, g_t) with a feasible fallback.
if all request widths equal one then
    Set M_t ← NATIVE and materialize the native target plan.
else
    Set M_t ← SPEC and materialize the speculative plan.
end if
Execute the target step and commit accepted tokens.
Record physical acceptance, latency, transitions, and Graph signature; replay the accepted path
through embedded candidates.
Update physical and counterfactual statistics separately in Θ.
return (P_t, g_t, M_t), Θ
```

4 EXPERIMENTS

Models and Evaluations. We evaluate GCHTA on the pre-trained QWEN3-8B and QWEN3-CODER-30B-A3B-INSTRUCT (An Yang, 2025) models. The workloads cover three representative categories: mathematical reasoning (GSM8K (Cobbe et al., 2021)), code generation (HumanEval (Chen et al., 2021)), and open-domain dialogue (MT-Bench (Zheng et al., 2023b)). To assess serving performance in realistic production environments, we implement our method within the open-source SGLang (Zheng et al., 2023a) framework using the FlashInfer backend. We report the average acceptance length (τ) and end-to-end decoding throughput (tokens per second). All experiments are conducted on NVIDIA H800 GPUs.

Datasets. GCHTA is a training-free, inference-time decoding algorithm. We directly use the pre-trained DFlash draft models and introduce no additional training datasets. The experimental focus is entirely on evaluating the hardware-constrained tree allocation and execution policy.

Implementation. Our implementation is built strictly on top of the DFlash draft model. To ensure a fair comparison, we maintain the exact same draft model architecture, network configurations, and generation hyperparameters as the original DFlash. This strictly matched setup isolates our contributions, demonstrating that any performance gains stem purely from the proposed algorithmic allocation and execution mechanisms rather than model differences.

Baselines. We compare GCHTA against the state-of-the-art speculative decoding method, DFlash (Chen et al., 2026). Because GCHTA utilizes the exact same draft model, DFlash serves as a strictly controlled baseline. This isolates and demonstrates how our workload-adaptive, graph-constrained allocation pushes the boundary of speculative decoding performance beyond the standard DFlash implementation.

Table 1: Throughput (tok/s) and average acceptance length τ comparison of DFlash and GCHTA on the QWEN3-CODER-30B-A3B-INSTRUCT model across different batch sizes (BS).

BS	Baseline			DFlash						GCHTA					
	GSM8K	HumanEval	MT-Bench	GSM8K		HumanEval		MT-Bench		GSM8K		HumanEval		MT-Bench	
	tp	tp	tp	tp	τ	tp	τ	tp	τ	tp	τ	tp	τ	tp	τ
1	224	235	235	414		457		486		446		667		524	
2	376	378	390	716		765		850		781		1115		927	
4	666	667	693	1227		1243		1460		1405		1875		1672	
8	1130	1146	1179	1942	<u>3.76</u>	2055	<u>4.05</u>	2070	<u>4.23</u>	2283	**4.84**	3094	**7.32**	1672	**5.12**
16	1953	1823	1910	3095		3173		3385		3528		4652		3859	
32	3083	2783	2930	4196		4457		4688		4854		6781		5423	
64	4799	4348	4546	5170		5508		5769		6130		8638		6840	

4.1 MAIN RESULTS

As shown in Tables 1 and 2, GCHTA outperforms both the target-only baseline and DFlash in every evaluated configuration. Applying equal weight to each model–workload–batch-size point, GCHTA achieves a mean throughput improvement over DFlash of 26.41% for QWEN3-CODER and 22.59% for QWEN3-8B. The consistent positive improvement across GSM8K, HumanEval, and MT-Bench indicates that these results are not confined to a single prompt or completion distribution.

Based on the arithmetic means spanning BS $= 1$–64 reported in the main tables, GCHTA significantly increases the mean accepted length (τ) relative to DFlash: by 28.72%, 80.74%, and 21.04% across the three QWEN3-CODER workloads, and by 21.39%, 44.57%, and 32.06% for the corresponding QWEN3-8B workloads. The particularly pronounced acceptance increase on HumanEval (e.g., from a mean of 4.054 to 7.318 on QWEN3-CODER) directly aligns with its substantial throughput improvement.

This dual improvement in both throughput and accepted length validates the core mechanisms proposed in our methodology. Specifically, the superiority of GCHTA can be attributed to the joint optimization of system design and execution costs, providing complementary evidence for both the numerator and denominator of the ideal objective in equation 4:

- **Constrained Candidate Expansion (via Topology Substitution).** The enhanced accepted length is not a result of a larger drafting budget, but rather of optimal candidate construction (Section 3.2). Both systems utilize the identical target model, DFlash checkpoint, 16-token draft block, TP4 placement, attention backend, and scheduler configuration. While DFlash verifies a fixed block-drafting trajectory, GCHTA utilizes equation 5 to reorganize the identical position-wise distributions into chain, spine-and-branch, or bushy candidates. By preserving a high-confidence explicit greedy spine and allocating the remaining capacity to plausible siblings, GCHTA's tree-based verifier can proceed along alternative target-consistent branches even after a single drafted trajectory diverges. This expansion is strictly bounded by equation 2 and Graph-envelope compatibility, maximizing probability coverage within an already executable envelope rather than indiscriminately widening the verification tree.
- **Heterogeneous Request Allocation (via Marginal Utility).** For every model–workload pair, the throughput gain at BS $= 64$ consistently exceeds the gain at BS $= 1$, despite fluctuations at intermediate points. Specifically, the BS $1 \rightarrow 64$ gains increase from $7.73\% \rightarrow 18.57\%$ (GSM8K), $45.95\% \rightarrow 56.83\%$ (HumanEval), and $7.82\% \rightarrow 18.56\%$ (MT-Bench) for QWEN3-CODER; and from $1.44\% \rightarrow 38.84\%$, $24.39\% \rightarrow 77.69\%$, and $7.76\% \rightarrow 78.28\%$ respectively for QWEN3-8B. As the batch size grows, request-local frontiers are more likely to exhibit varying marginal returns. Equation 6 assigns width–topology increments to the requests offering the largest risk-adjusted committed-token gain per unit of physical cost, while slack filling reuses residual slots without crossing a capture boundary.
- **Counterfactual Replay and Physical Cost Control.** Achieving high throughput requires controlling physical costs (draft, target, commit, KV, padding, and transitions) via equation 3. Counterfactual supertree feedback (equation 8) dynamically updates the acceptance estimates of structurally covered alternatives without conflating inferred acceptance with physical latency measurements, progressively refining subsequent frontier selections. Furthermore, transition awareness and the native fallback option act primarily on the denominator of equation 4 by circumventing mode changes or speculative bookkeeping when fixed costs cannot be recovered. At larger scales (BS $= 128$–256), equation 3 correctly favors the native target-only option, driving speculative acceptance length close to one.

In summary, the end-to-end evaluations demonstrate that the observed difference in performance is a consequence of distinct optimization units. GCHTA successfully shifts the optimization paradigm from merely expanding the draft verification budget to optimally reallocating a fixed CUDA Graph execution envelope. Under this meticulously matched protocol, GCHTA increases useful committed progress while effectively controlling the physical costs incurred to obtain that progress, ultimately translating into a significant system-level throughput leap.

Table 2: Throughput (tok/s) and average acceptance length τ comparison of DFlash and GCHTA on the QWEN3-8B model across different batch sizes (BS).

	Baseline			DFlash						GCHTA					
BS	GSM8K	HumanEval	MT-Bench	GSM8K		HumanEval		MT-Bench		GSM8K		HumanEval		MT-Bench	
	tp	tp	tp	tp	τ	tp	τ	tp	τ	tp	τ	tp	τ	tp	τ
1	203	202	201	417		328		361		423		408		389	
2	394	381	388	747		586		601		799		759		656	
4	765	747	756	1434		1112		1122		1483		1425		1285	
8	1471	1435	1436	2599	3.32	1951	2.58	1974	2.62	2620	**4.03**	2503	**3.73**	2321	**3.46**
16	2708	2655	2671	4030		3037		3034		4168		3962		3458	
32	4680	4625	4731	5299		4007		3990		5635		5354		4716	
64	7852	7792	7966	5728		4446		4522		7953		7900		8062	

5 ABLATION STUDY

We evaluate the principal GCHTA components using average acceptance length τ as the sole metric. All configurations use the same QWEN3-8B target model, DFlash draft model, tokenizer, input trace, hardware configuration, and measurement procedure. Each ablation removes one component from the full system while keeping the remaining configuration unchanged. The comparison is therefore a conditional one-factor ablation, not an additive decomposition of component effects.

Table 3: Impact of GCHTA components on average acceptance length (τ) on QWEN3-8B. Larger values indicate that more target-consistent tokens are accepted per speculative verification step.

Configuration	τ
Full GCHTA	4.03
Homogeneous Allocation	3.16
No Topology Substitution	3.14
No Counterfactual Feedback	3.17

Acceptance-length comparison. The full configuration achieves the highest acceptance length, with $\tau = 4.03$. Removing any evaluated component reduces τ to 3.14–3.17, an absolute decrease of 0.86–0.89 tokens, or 21.3%–22.1% relative to the full system. Thus, the measured acceptance advantage depends on all three components in this configuration. Because the differences are close and only one aggregate value is reported per configuration, the table does not support a statistical ranking among the ablations.

Role of heterogeneous allocation. The homogeneous-allocation variant restricts all requests in a batch to one allocation decision and obtains $\tau = 3.16$, which is 0.87 tokens below full GCHTA. Because requests inherently differ in draft confidence, full GCHTA can assign effective widths and topologies selectively, concentrating the fixed verification budget on continuations with higher expected target agreement. Removing this heterogeneity forces high- and low-confidence requests to share the same allocation, reducing the coverage of target-consistent paths. The result is consistent with a request-level allocation benefit rather than a uniform increase in global tree width.

Role of topology substitution. Disabling topology substitution yields the lowest value, $\tau = 3.14$, an absolute decrease of 0.89 tokens. The full method can transform the same draft distributions into chain, spine-and-branch, or bushy candidates while remaining within a compatible Graph envelope. Removing this flexibility restricts the set of executable candidate shapes and reduces the chance that the available verification positions cover a target-consistent continuation. The numerical decrease is the largest among the ablations, but its statistical distinction from the other two cannot be conclusively established from the reported values.

Role of counterfactual feedback. Removing counterfactual feedback yields $\tau = 3.17$, an absolute decrease of 0.86 tokens. Counterfactual replay does not create a physically accepted token; it transfers acceptance evidence from verified paths to structurally covered alternatives and updates subsequent candidate selection. Without this feedback, the controller has less information for distinguishing promising from unpromising frontier options. The decrease is therefore consistent with less informed candidate selection, although the table alone cannot separate this effect from interactions with allocation and topology.

6 CONCLUSION

GCHTA formulates batched speculative decoding as a graph-constrained resource allocation problem. Moving beyond uniform verification widths, it constructs request-local candidate frontiers and dynamically maps them to compatible tree topologies within a fixed CUDA GRAPH envelope. Guided by expected acceptance and physical cost, and further refined by counterfactual supertree feedback, GCHTA optimally allocates speculative capacity without inflating execution overhead. Extensive evaluations on QWEN3 models across diverse workloads (GSM8K, HumanEval, and MT-Bench) demonstrate that GCHTA consistently surpasses DFlash in both throughput and acceptance length. Crucially, these gains amplify at larger batch sizes, highlighting the scalability of heterogeneous allocation and envelope reuse. Ablation studies confirm that all three core mechanisms fundamentally contribute to this performance leap. Future work will extend this framework to broader model families and explore finer-grained statistical profiling of its component dynamics.

AI USE STATEMENT

In this work, we used generative AI tools ChatGPT for text polishing, grammatical correction, and improving the overall readability of the manuscript. We have not used generative AI tools to formulate our core research ideas, design the algorithmic frameworks (e.g., the graph-constrained resource allocation and counterfactual supertree feedback), or execute the actual experimental evaluations. Using AI for data fabrication or synthesizing experimental results is not applicable to this work, as all evaluations were conducted physically on real hardware using standard open-source benchmarks.

Additionally, we used generative AI tools for minor engineering assistance, such as generating routine data visualization scripts and formatting LaTeX tables. We have thoroughly reviewed all AI-assisted work. Specifically, all AI-polished text was manually cross-checked by the authors to ensure it faithfully conveys our precise scientific intent without introducing inaccuracies. Furthermore, any AI-assisted utility code or plotting scripts were explicitly verified, executed, and tested for correctness by the authors before being integrated into our workflow. We take full responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure the comprehensive reproducibility of our work, we have detailed both the algorithmic formulations and the system-level engineering implementations. The formal definitions of the graph-constrained resource allocation and the counterfactual supertree feedback mechanisms are thoroughly presented in Section 3.3 and Section 3.5. Because the performance of speculative decoding is highly sensitive to physical execution overhead, we have exhaustively documented the runtime engineering mechanisms—including variable-width CUDA Graphs, packed-ragged verification, and PlanToken validation—in Appendix A. To facilitate direct replication of our throughput and acceptance length results, the complete source code of GCHTA is submitted as an anonymous downloadable link in the supplementary materials.

REFERENCES

Baosong Yang Beichen Zhang Binyuan Hui Bo Zheng Bowen Yu Chang Gao Chengen Huang Chenxu Lv Chujie Zheng Dayiheng Liu Fan Zhou Fei Huang Feng Hu Hao Ge Haoran Wei Huan Lin Jialong Tang Jian Yang Jianhong Tu Jianwei Zhang Jianxin Yang Jiaxi Yang Jing Zhou Jingren Zhou Junyang Lin Kai Dang Keqin Bao Kexin Yang Le Yu Lianghao Deng Mei Li Mingfeng Xue Mingze Li Pei Zhang Peng Wang Qin Zhu Rui Men Ruize Gao Shixuan Liu Shuang Luo Tianhao Li Tianyi Tang Wenbiao Yin Xingzhang Ren Xinyu Wang Xinyu Zhang Xuancheng Ren Yang Fan Yang Su Yichang Zhang Yinger Zhang Yu Wan Yuqiong Liu Zekun Wang Zeyu Cui Zhenru Zhang Zhipeng Zhou Zihan Qiu An Yang, Anfeng Li. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, june 2024. *URL http://arxiv. org/abs/2401.10774*.

Geoffrey Irving Jean-Baptiste Lespiau Laurent Sifre John Jumper Charlie Chen, Sebastian Borgeaud. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. arxiv e-prints. *arXiv preprint arXiv:2110.14168*, 2021.

Xiangxiang Gao, Weisheng Xie, Xuwei Fang, Chen Hang, Changqun Li, Yuhan Lin, Xiaolong Xu, et al. Talon: Breaking the synchronization barrier in speculative decoding with hybrid model-based and retrieve-based drafting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 21216–21224, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Empirical Methods in Natural Language Processing*, 2024b.

Liran Ringel and Yaniv Romano. Accelerating speculative decoding with block diffusion draft trees. *arXiv preprint arXiv:2604.12989*, 2026.

Shengyin Sun, Yiming Li, Xing Li, Yingzhao Lian, Weizhe Lin, Huiling Zhen, Zhiyuan Yang, Xianzhi Yu, Chen Chen, Mingxuan Yuan, et al. Scaling up, speeding up: A benchmark of speculative decoding for efficient llm test-time scaling. In *International Conference on Learning Representations*, volume 2026, pp. 103030–103060, 2026.

L Zheng, L Yin, Z Xie, C Sun, J Huang, CH Yu, S Cao, C Kozyrakis, I Stoica, JE Gonzalez, et al. Sglang: Efficient execution of structured language model programs. arxiv, 2023a.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023b.

A Engineering Implementation

The mechanisms in this appendix make the controller practical in the target runtime. They improve execution efficiency, measurement quality, or runtime safety, but they are not separate algorithmic contributions.

A.1 Variable-width and profile-specific CUDA Graphs

The runtime captures only verified shape and topology combinations. In the Qwen3-Coder validation profile, the main speculative widths are $4, 8, 16$ and the decode batch capacities include $1, 2, 4, 8, 10, 16, 32, 64, 128, 256$. Target and draft Graph paths are configured independently, and replay feedback records the actual shape and variant instead of assuming that a nominal width implies a particular physical kernel.

A.2 Packed-ragged and mixed-mask verification

Packed-ragged verification stores heterogeneous request trees in one target forward pass. Mixed masks preserve each request's ancestor visibility while preventing attention to unrelated branches. These mechanisms provide the physical execution substrate for topology substitution; they do not define the allocation objective.

A.3 GPU tree construction and commit

GPU-resident tree construction and commit reduce CPU staging, host–device synchronization, and tensor reshaping. They are especially important at small batch sizes, where a fixed controller cost can dominate the target-model saving. They do not change the selected tree or its acceptance semantics.

A.4 Deferred draft KV and selective hidden state

Draft-KV materialization and hidden-state capture are conditional on the selected action. Unused speculative state is not materialized for target-only or pruned requests. This reduces memory traffic and KV pressure while preserving the controller's logical decision.

A.5 PlanToken validation and recovery isolation

Every plan is bound to a per-round PlanToken containing the action, shape, Graph capacity, cohort, and execution-mode metadata. Replay feedback validates the token against the physical result before updating steady-state statistics. This prevents stale plans, rank-divergent transitions, shape mismatches, and recovery rounds from contaminating the allocation model.

A.6 Reversible native basin and cold-start anchor

When target-only execution is measured to be better, the scheduler may enter a reversible native basin instead of repeatedly traversing an empty speculative path. Low-batch cold-start execution can use a fixed speculative anchor while physical statistics are established. Immutable anchor-plan templates and bounded bookkeeping fast paths reduce startup overhead, but are disabled for recovery, forced reprofiling, Graph unavailability, or stale-token conditions.

A.7 Stage-level measurement

The profiler separates draft, target, commit, draft-KV, padding, scheduler, and transition latency. It records the actual replayed shape and Graph hit or miss rather than charging the nominal plan. Counterfactual feedback is stored separately from physical measurements, and transient recovery or mode-switch rounds are quarantined from steady-state action statistics.