Technical Report: Routing, Confidence, Verification, and Routing Overhead in EAGLE and HASS

Codebase Analysis Report

February 25, 2026

Abstract

This report documents how routing is implemented in the EAGLE and HASS repositories, how draft confidence is computed, how verifier acceptance works, and how routing overhead can be quantified. The report is based on direct code inspection of both repositories and analysis of generated benchmark artifacts.

1 Scope and Repositories

This report compares:

- EAGLE codebase: `Eagle-Code`
- HASS codebase: `Hass-Code`

Primary focus:

- Step-level dual-head confidence routing
- Confidence definition and extraction
- Verifier accept/reject logic
- Overhead introduced by routing

2 Code Map (What Was Audited)

2.1 EAGLE

- Core generation: `eagle/model/ea_model.py`
- Dual routing model: `eagle/model/ea_model_dual.py`
- Draft tree and confidence: `eagle/model/cnets1.py` (EAGLE-2), `eagle/model/cnets.py`
- Verifier/update logic: `eagle/model/utils.py`
- Dual eval driver: `eagle/evaluation/gen_dual_head_eval.py`
- Routing eval and scripts: `eagle/evaluation/gen_routing_eval.py`, `evaluate/run_dual_head_eval.sh`, `evaluate/run_routing_eval.sh`
- Confidence analysis: `eagle/evaluation/gen_confidence_analysis.py`

2.2 HASS

- Core generation: `model/ea_model.py`
- Dual routing model: `model/ea_model_dual.py`
- Draft tree and confidence: `model/cnets.py`
- Verifier/update logic: `model/utils.py`
- Dual eval driver: `evaluation/gen_dual_head_eval.py`
- Routing eval and scripts: `evaluation/routed_eval.py`, `scripts/run_dual_head_eval.sh`, `scripts/eval_routed.sh`
- Confidence analysis: `evaluation/gen_confidence_analysis.py`

3 Shared Decoding Mechanics

Both systems use speculative decoding with:

1. A **draft model** that proposes a tree of candidate continuations.
2. A **verifier model** (base LLM) that scores/accepts candidates.
3. KV-cache updates to avoid recomputing accepted prefixes.

Core functions (both repos):

- Tree proposal: `topK_genrate(...)`
- Verifier forward on tree: `tree_decoding(...)`
- Acceptance decision: `evaluate_posterior(...)`
- Commit accepted tokens and refresh tree: `update_inference_inputs(...)` (single-head path)

3.1 Verifier Acceptance Rule

In greedy mode (`logits_processor is None`), both repos implement:

- Compare candidate tokens against verifier argmax at each position.
- Accept the longest matching prefix.
- If no match, fall back to candidate 0.

In sampling mode:

- Apply processed verifier distribution.
- Sequential stochastic accept/reject over candidate path tokens.
- Renormalize when rejecting a token.

4 How Confidence Is Computed

4.1 Tree-level confidence used by dual routing

In both EAGLE and HASS draft tree code:

- `scores_list` stores cumulative log-probability scores along draft paths.
- Selected top paths are indexed by `top_scores_index`.
- Confidence is computed as:

$$c_i = \exp(s_i), \quad s_i \in \text{selected cumulative log-probabilities}$$

and returned as `draft_confidences`.

This is implemented in:

- EAGLE: `eagle/model/cnets1.py` (and `cnets.py`)
- HASS: `model/cnets.py`

4.2 Accepted vs rejected confidence logging

For analysis runs:

- EAGLE uses `eagenerate(..., confidence_log=True)` and labels draft tokens accepted vs rejected using verifier acceptance length and `retrieve_indices`.
- HASS confidence analysis extracts per-token draft logprobs (`return_logprobs=True`) and maps them to confidence via $\exp(\log p)$, then labels accepted vs rejected similarly.

5 How Routing Works

5.1 EAGLE dual-head routing

Implemented in `eagle/model/ea_model_dual.py`.

At each decoding step:

1. Both heads call `topK_genrate(..., return_confidence=True)`.
2. Per-head mean confidence is computed over proposed draft tokens.
3. Router chooses the head with larger mean confidence.
4. Verifier accepts/rejects from the chosen tree.
5. Both heads stay synchronized via accepted hidden states.

Decision rule:

$$h^* = \arg\max_{h \in \{1,2\}} \left(\frac{1}{N} \sum_{i=1}^{N} c_{h,i} \right)$$

5.2 HASS dual-head routing

Implemented in `model/ea_model_dual.py`, algorithmically aligned with EAGLE dual routing:

1. Generate draft tree+confidence from both heads.
2. Compare mean confidences.
3. Verify chosen tree and commit accepted tokens.
4. Continue with synchronized accepted hidden states.

5.3 Other routing modes present

Secondary routing flows also exist:

- EAGLE entropy router: `eagle/evaluation/gen_routing_eval.py`, choosing lowest draft entropy head.
- HASS keyword router: `evaluation/routed_eval.py`, using heuristic math keyword/pattern matching.

These are separate from the step-level dual confidence router.

6 Verification Procedure (What Is Actually Verified)

At each speculative iteration:

1. Verifier computes logits on tree candidates.
2. `evaluate_posterior` selects best candidate and accepted prefix length.
3. Accepted tokens are appended to `input_ids`.
4. Corresponding KV cache slices are copied into the active prefix region.
5. Draft generation restarts from accepted hidden states plus next sampled token.

This means routing affects **which draft tree is proposed**, but verifier logic remains unchanged.

7 Empirical Results from Existing Artifacts

All values below are computed from existing JSONL outputs already present in both repositories.

7.1 EAGLE: $\tau = \frac{\textbf{new\_tokens}}{\textbf{steps}}$

7.2 HASS: $\tau = \frac{\textbf{new\_tokens}}{\textbf{steps}}$

7.3 Confidence separation quality

Aggregate accepted vs rejected draft confidence (all 4 benchmarks combined):

Head	mt_bench	gsm8k	math_500	svamp
MathInstruct	2.541	4.923	5.284	4.812
ShareGPT	3.606	3.835	3.968	3.778
MathInstruct+ShareGPT (dual)	3.632	4.751	5.195	4.717
Δ vs best single	+0.026	-0.172	-0.089	-0.095

Table 1: EAGLE single-head vs dual confidence routing.

Head	mt_bench	gsm8k	math_500	svamp
MathInstruct	2.371	4.960	5.420	4.935
ShareGPT	3.889	4.165	3.975	4.092
MathInstruct+ShareGPT (dual)	3.934	4.932	5.377	4.879
Δ vs best single	+0.044	-0.028	-0.043	-0.056

Table 2: HASS single-head vs dual confidence routing.

7.4 Dual-run throughput (from wall_time in dual outputs)

8 Routing Overhead

8.1 Exact algorithmic overhead

Let one draft decision be one call to `_draft_best`, i.e., one confidence comparison event.

- Single-head: one draft head tree per decision.
- Dual-head routing: two draft head trees per decision.

Therefore, draft-side call overhead is exactly +100% per decision.

8.2 Measured overhead proxy from completed dual runs

Using logged dual outputs, extra draft-head calls over single-head equivalent are:

- EAGLE: 110,588 extra head calls for 514,393 generated tokens ($\approx$ 214.99 extra calls / 1k tokens)
- HASS: 105,089 extra head calls for 509,028 generated tokens ($\approx$ 206.45 extra calls / 1k tokens)

8.3 End-to-end runtime overhead formula

If:

- T_d: draft-stage cost per decision for one head
- T_v: verifier-stage cost per decision

then:

$$T_{\text{single}} \approx T_d + T_v, \quad T_{\text{dual}} \approx 2T_d + T_v + \epsilon$$

Model	Mean accepted	Mean rejected	Delta	Counts (acc/rej)
EAGLE MathInstruct	0.534534	0.046742	0.487792	395728 / 6255165
EAGLE ShareGPT	0.496855	0.053631	0.443224	378471 / 7630130
HASS MathInstruct	0.441771	0.178571	0.263200	281528 / 45484
HASS ShareGPT	0.428201	0.224287	0.203914	295899 / 55220

Table 3: Accepted draft tokens consistently have higher confidence than rejected tokens.

System	mt_bench	gsm8k	math_500	svamp
EAGLE Tok/s	53.15	78.69	89.06	83.86
HASS Tok/s	58.31	82.98	92.62	87.86

Table 4: Dual-mode token throughput from recorded `wall_time`.

$$\text{Overhead ratio} \approx \frac{T_{\text{dual}} - T_{\text{single}}}{T_{\text{single}}} \approx \frac{T_d}{T_d + T_v}$$

where ϵ (mean confidence compare) is typically negligible.

Important: exact end-to-end overhead percentage versus single-head wall-clock cannot be computed from current artifacts alone because single-head confidence logs in these folders do not include `wall_time` fields.

9 Reproduction Commands

9.1 EAGLE dual run

```
cd /home/zbibm/MOSS---Mixture-of-Speculative-Samplers/Eagle-Code
BASE_MODEL_PATH=/home/zbibm/MOSS---Mixture-of-Speculative-Samplers/base \
EA_MODEL_PATH1=checkpoints/Eagle-MathInstruct_20epochs \
EA_MODEL_PATH2=checkpoints/Eagle-ShareGPT_20epochs \
bash evaluate/run_dual_head_eval.sh
```

9.2 HASS dual run

```
cd /home/zbibm/MOSS---Mixture-of-Speculative-Samplers/Hass-Code
BASE_MODEL_PATH=/home/zbibm/MOSS---Mixture-of-Speculative-Samplers/base \
EA_MODEL_PATH1=checkpoints/MathInstruct_20epochs/state_final \
EA_MODEL_PATH2=checkpoints/ShareGPT_20epochs \
bash scripts/run_dual_head_eval.sh
```

10 Key Implementation Differences

- EAGLE dual implementation is EAGLE-2 style (`cnets1.py`) with multi-architecture base support (Llama/Qwen2/Qwen3/Mixtral paths in single-head model class).
- HASS dual implementation currently targets Llama architecture in `model/ea_model_dual.py`.

- HASS confidence-analysis path reconstructs token confidence from returned logprobs ($\exp(\log p)$); EAGLE confidence mode uses native draft confidence logging path in `EaModel.eagenerate`.
- HASS dual generation return signature includes `accept_length_list`; EAGLE dual returns `(input_ids, new_token, idx)` in log mode.

11 Conclusion

Both EAGLE and HASS dual routers implement the same core strategy: choose, at each speculative step, the draft head with higher mean cumulative path confidence, then let the verifier accept/reject normally. Confidence is consistently discriminative (accepted $>$ rejected), and routing improves conversational benchmark behavior while remaining close to or below the strongest math-specialist head on pure math benchmarks. Draft-side routing overhead is structurally +100% in draft calls; the practical end-to-end impact depends on the draft-vs-verifier compute split.