Masters Research Project

Entropy-Adaptive Branching for Efficient Semantic Uncertainty Estimation

Donia Gasmi

Supervisors:

Dr. Geovani Rizk & Dr. Gauthier Voron

Professor:

Prof. Rachid Guerraoui

EPFL

EPFL

Table of Content

EPFL

1 Introduction

Large language models (LLMs) achieve impressive performance across diverse tasks, yet remain prone to generating plausible but incorrect outputs, a phenomenon known as hallucination. As these models are deployed in high-stakes domains such as healthcare and legal reasoning, reliably quantifying *when* a model is uncertain becomes as important as the predictions themselves.

Existing uncertainty quantification approaches face a fundamental efficiency-quality trade-off. Single-sample methods like token-level entropy are computationally cheap but miss semantic uncertainty: a model may generate contradictory answers (e.g., "Paris" vs. "Lyon" for France's capital) while remaining confident at each token. Multi-sample methods such as semantic entropy [2] address this by generating diverse outputs and measuring disagreement, but require independent forward passes per sample, incurring computational costs that scale linearly with sample count.
The core challenge is generating diverse samples *efficiently*. Naive multi-sample generation redundantly encodes the same prompt multiple times. Beam search prioritizes high-probability paths at the cost of diversity; temperature sampling produces diversity at the cost of coherence. No existing method efficiently explores the model's uncertainty landscape while enabling reliable disagreement detection.

We introduce **Entropy-Adaptive Branching (EAB)**, a sampling algorithm that generates diverse samples by branching *only* at high-entropy token positions. EAB encodes the prompt once, then dynamically creates new generation paths when normalized entropy exceeds a threshold τ, indicating genuine next-token ambiguity. This strategy shares prefix computation across samples while selectively exploring uncertain regions.
EAB forms Layer 1 of a two-layer uncertainty system. It efficiently generates N diverse samples by capturing token-level uncertainty through adaptive branching. Layer 2 applies **Semantic Entropy (SE)**: clustering responses by meaning and computing entropy over the resulting distribution. Together, the layers combine local (token) and global (semantic) signals for comprehensive uncertainty quantification.

Contributions.

- **Entropy-Adaptive Branching:** A novel algorithm that generates diverse samples efficiently by branching only at high-entropy positions, reducing redundant computation while preserving sample diversity.
- **Two-layer uncertainty architecture:** A unified framework combining token-level (EAB) and semantic-level (SE) uncertainty signals.
- **Empirical evaluation:** Comprehensive experiments analyzing EAB's computational efficiency, branching behavior, and uncertainty estimation quality across varying configurations.

The remainder of this report is organized as follows: Section 2 provides background on entropy measures and semantic clustering. Section 3 details EAB and the two-layer architecture. Section 4 describes implementation choices. Section 5 presents experimental results. Section 6 concludes and discusses limitations and future work.

2 Background

This section outlines foundational concepts relevant to our work on efficient multi-sample generation for LLM uncertainty quantification. We cover autoregressive decoding, KV-cache reuse across shared prefixes, and techniques for assessing semantic diversity among generated responses. We review autoregressive decoding, prefix reuse via KV-caching, and methods for quantifying and interpreting output uncertainty, particularly those based on semantic diversity across multiple samples.

2.1 Large Language Models and Autoregressive Generation

Large language models (LLMs) generate text autoregressively by predicting tokens sequentially based on preceding context. Given an input prompt $\mathbf{x} = (x_1, \ldots, x_n)$, the model produces a probability distribution over the vocabulary $\mathcal{V}$ for the next token:

$$P(x_{n+1} \mid x_1, \ldots, x_n; \theta) \tag{1}$$

where θ represents the model parameters. Modern transformer architectures [7] maintain a key-value (KV) cache to store intermediate representations, enabling efficient sequential generation without recomputing past tokens.

2.2 KV-Cache Sharing and Prefix Reuse

The KV-cache stores the key and value vectors computed for each token in the input sequence. When generating multiple responses from the same prompt, these cached representations for the shared prefix can be reused across samples, avoiding redundant computation.

For a prompt of length n and continuations diverging at position t, all samples share the KV-cache for positions 1 to $t-1$. Only the divergent portions from position t onward require separate cache storage. This prefix sharing enables memory-efficient parallel generation: for a model with L layers and hidden dimension d, generating m samples of total length T each without sharing requires $O(m \cdot L \cdot d \cdot T)$ memory. With prefix sharing up to position t, the total memory reduces to $O(L \cdot d \cdot (t + m \cdot (T - t)))$, where the shared prefix of length t is stored once and only the m divergent continuations require separate storage.

The challenge lies in strategically identifying branching points that maximize prefix sharing while maintaining sufficient diversity among generated samples.

2.3 Uncertainty Quantification in LLMs

Uncertainty quantification aims to measure the reliability of model outputs, particularly important for high-stakes applications. We distinguish between two types of uncertainty:

- **Aleatoric uncertainty** arises from inherent randomness or ambiguity in the task itself, such as questions with multiple valid answers.
- **Epistemic uncertainty** reflects the model's lack of knowledge, often due to limited training data or out-of-distribution inputs.

A straightforward uncertainty measure is *predictive entropy*, computed from the model's token probability distribution:

$$H(P) = -\sum_{v \in \mathcal{V}} P(v) \log P(v) \tag{2}$$

However, this single-token metric provides only local uncertainty and does not account for the semantic content of full responses.

An alternative approach generates multiple complete outputs through sampling and measures their disagreement [4]. A key challenge is that naive comparison of token sequences fails to capture semantic equivalence. For example, "The capital is Paris" and "Paris is the capital" express the same meaning despite having different surface forms.

2.4 Semantic Entropy and Entailment-Based Methods

Kuhn et al. [2] exemplify this sampling-based approach by generating multiple responses and measuring disagreement at the semantic level. They introduced *semantic entropy*, which clusters semantically equivalent outputs before computing uncertainty. Their method uses bidirectional entailment to determine equivalence: two responses y_i and y_j are considered equivalent if a natural language inference (NLI) model predicts $y_i \vDash y_j$ and $y_j \vDash y_i$.
Given a set of sampled responses $\mathcal{Y} = \{y_1, \ldots, y_m\}$ partitioned into semantic clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$, semantic entropy is computed as:

$$H_{\text{sem}} = -\sum_{i=1}^{k} p(C_i) \log p(C_i) \tag{3}$$

where $p(C_i) = \sum_{y \in C_i} p(y)$ aggregates the probabilities of responses within each cluster.
While effective, entailment-based methods have limitations. NLI models themselves are imperfect and may introduce errors in clustering decisions, potentially misclassifying semantically distinct responses as equivalent or vice versa.

2.5 Embedding-Based Semantic Similarity

An alternative approach leverages dense embeddings from sentence encoders [5]. Given a response y, an encoder $f : \mathcal{Y} \to \mathbb{R}^d$ maps it to a fixed-dimensional vector $\mathbf{e}_y = f(y)$. Semantic similarity can then be measured using cosine similarity:

$$\text{sim}(y_i, y_j) = \frac{\mathbf{e}_{y_i} \cdot \mathbf{e}_{y_j}}{\|\mathbf{e}_{y_i}\| \|\mathbf{e}_{y_j}\|} \tag{4}$$

Clustering algorithms such as hierarchical agglomerative clustering [8] can group semantically similar responses based on embedding distances. This approach avoids the need for entailment models while maintaining the ability to capture semantic equivalence beyond surface-level token matching.

2.6 Computational Constraints in Sampling

Generating multiple diverse samples for uncertainty estimation is computationally expensive. Each sample requires maintaining its own KV-cache during generation. For a model with hidden dimension d, L layers, and maximum sequence length T, storing KV-caches for m samples requires $O(m \cdot L \cdot d \cdot T)$ memory.
While sequential generation of samples can reduce peak memory usage, it incurs prohibitive time costs, as each sample must wait for previous generations to complete. Conversely, parallel generation of all samples minimizes latency but requires simultaneous storage of all KV-caches. This presents a fundamental tradeoff between time and memory efficiency. Practical uncertainty quantification at scale requires sampling strategies that navigate this tradeoff effectively, reducing computational overhead while maintaining sample diversity.

3 Design

Our system quantifies LLM uncertainty through a two-layer architecture that combines efficient multi-sample generation with semantic analysis. This section describes the design rationale and key components.

3.1 Two-Layer Uncertainty Quantification (EAB-SE)

We quantify LLM uncertainty through two complementary layers:

1. **Token-level uncertainty (Layer 1):** Measured during generation via next-token entropy. High entropy triggers adaptive branching, producing diverse continuations where the model is uncertain.
2. **Semantic-level uncertainty (Layer 2):** Measured after generation by clustering responses into meaning-equivalent groups and computing entropy over cluster assignments. High semantic entropy indicates disagreement among generated responses.

Core Idea. While token-level entropy captures local ambiguity (e.g., choosing between "Paris" and "Lyon"), it cannot distinguish between meaningful disagreement and superficial variation (e.g., "The capital is Paris." vs. "Paris is the capital."). Semantic entropy resolves this by grouping responses based on meaning, not surface form. Together, the two layers provide a more complete picture of uncertainty:

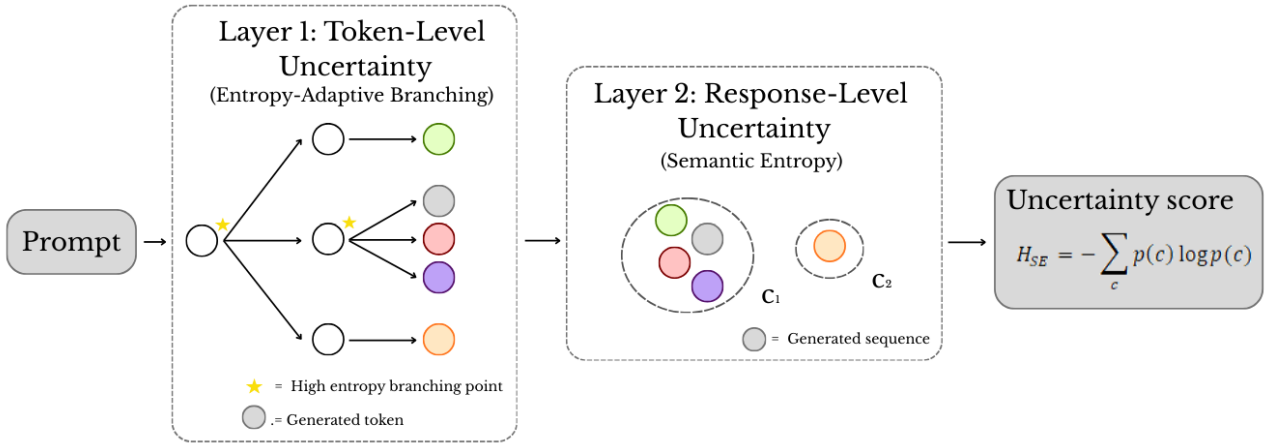


Figure 1: Two-layer uncertainty quantification architecture. Layer 1 generates diverse samples via entropy-adaptive branching; Layer 2 clusters them by semantic similarity and computes response-level entropy to produce a final uncertainty score.

3.2 Layer 1: Entropy-Adaptive Branching (EAB)

EAB generates diverse samples efficiently by dynamically branching only when the model is uncertain, while reusing computation for shared prefixes.

3.2.1 Core Idea

Traditional multi-sample generation processes the prompt independently for each sample, duplicating computation for the shared prefix. EAB eliminates this redundancy by:

- Encoding the prompt **once** and caching its KV states
- Generating tokens autoregressively, **sharing all intermediate states** until a branching decision
- **Branching only at high-entropy positions**, where the model's next-token distribution is ambiguous
- Enforcing a **fixed path budget** to prevent exponential growth

This efficiency is illustrated in Figure 2: EAB computes each token, prompt or generated, exactly once per unique prefix, regardless of how many samples diverge afterward.

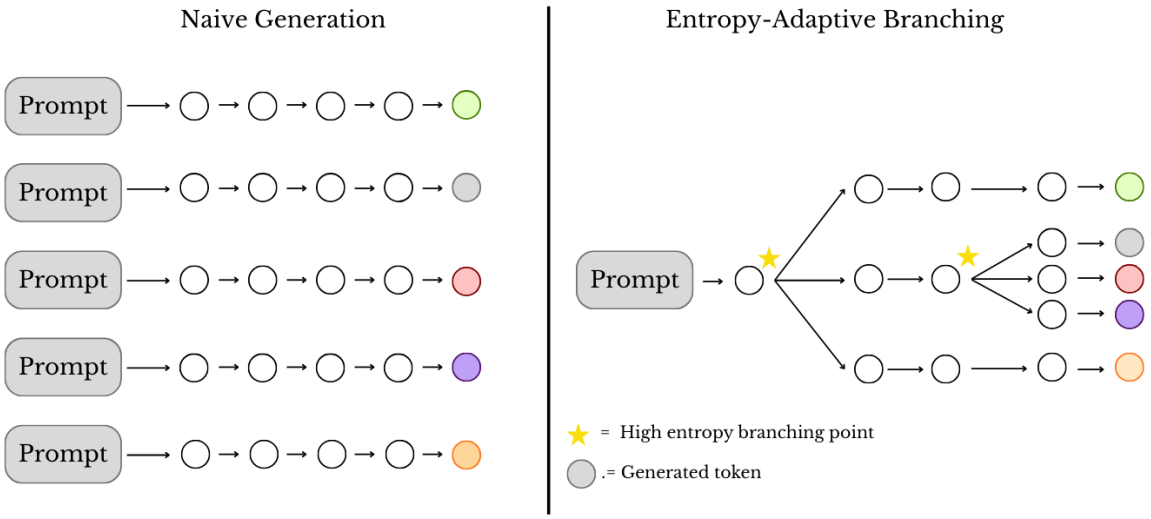


Figure 2: Conceptual comparison of naive generation versus Entropy-Adaptive Branching (EAB). Left: Naive generation re-encodes the prompt for each sample. Right: EAB encodes the prompt once and branches adaptively at high-entropy positions (stars), producing diverse samples with minimal redundant computation.

3.2.2 Entropy as Branching Signal

We use normalized entropy to measure token-level uncertainty during generation:

$$H_{\text{norm}}(p) = -\frac{\sum_{i=1}^{V} p_i \log p_i}{\log V} \tag{5}$$

where p is the next-token probability distribution over vocabulary V. Normalization ensures values lie in $[0, 1]$, enabling consistent thresholds across models and prompts.

To validate entropy as a reliable branching signal and calibrate the threshold τ, we conducted a pilot study on 200 diverse prompts, grouped by expected model confidence: **high-confidence** (70 factual QA, e.g., "What is the capital of France?"), **medium-confidence** (65 advice or opinion prompts), and **low-confidence** (65 creative or open-ended prompts, e.g., "Write a short poem about forbidden love.").

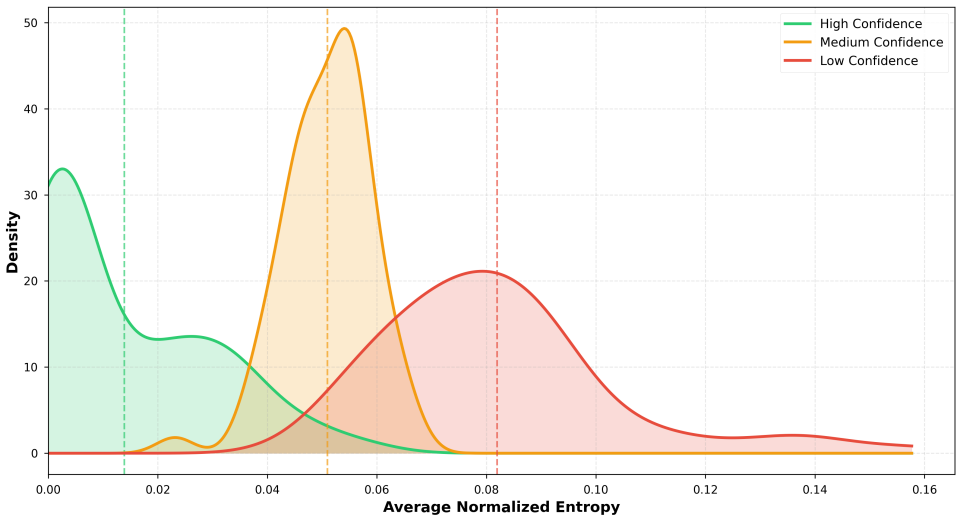


Figure 3: Average normalized entropy distributions by prompt confidence level. High-confidence prompts (green) cluster at low entropy; medium (orange) and low (red) confidence show progressively higher entropy.

For each prompt, we performed standard autoregressive generation (greedy decoding) and recorded the normalized entropy at every token. We then computed the average normalized entropy per prompt and plotted the resulting distributions (Figure 3). Statistical analysis confirmed strong separation between groups:

- One-way ANOVA: $F = 315.85$, $p < 0.001$ → significant differences in mean entropy
- Cohen's d: Large effect sizes (> 1.9) between adjacent groups → meaningful separation

Based on this, we set $\tau = 0.055$; the 75th percentile of medium-confidence prompts. This choice ensures:

- High-confidence prompts branch rarely (only 44% exceed τ) → preserves efficiency
- Medium/low-confidence prompts branch reliably (100% exceed τ) → captures true uncertainty
- Classification accuracy: 96.5% in distinguishing high vs. medium/low confidence based solely on whether branching occurred

This threshold operationalizes our core design principle: **let the number of generated samples reflect true uncertainty**, turning generation dynamics into a reliable signal for downstream uncertainty quantification.

3.2.3 Path Management

EAB maintains a dynamic set of active paths, bounded by a maximum count M (default: 20). At each generation step:

1. Compute next-token entropy for each path.
2. If $H_{\text{norm}} > \tau$ and paths remain within budget, spawn $k = 3$ new paths by sampling distinct tokens.
3. If total paths exceed M, retain only the top-M by cumulative log-probability.

Unlike hard-stop methods, our adaptive strategy ensures exploration throughout generation, capturing more entropy peaks, as shown in Figure 4, without exceeding memory limits, thanks to probability-based pruning.

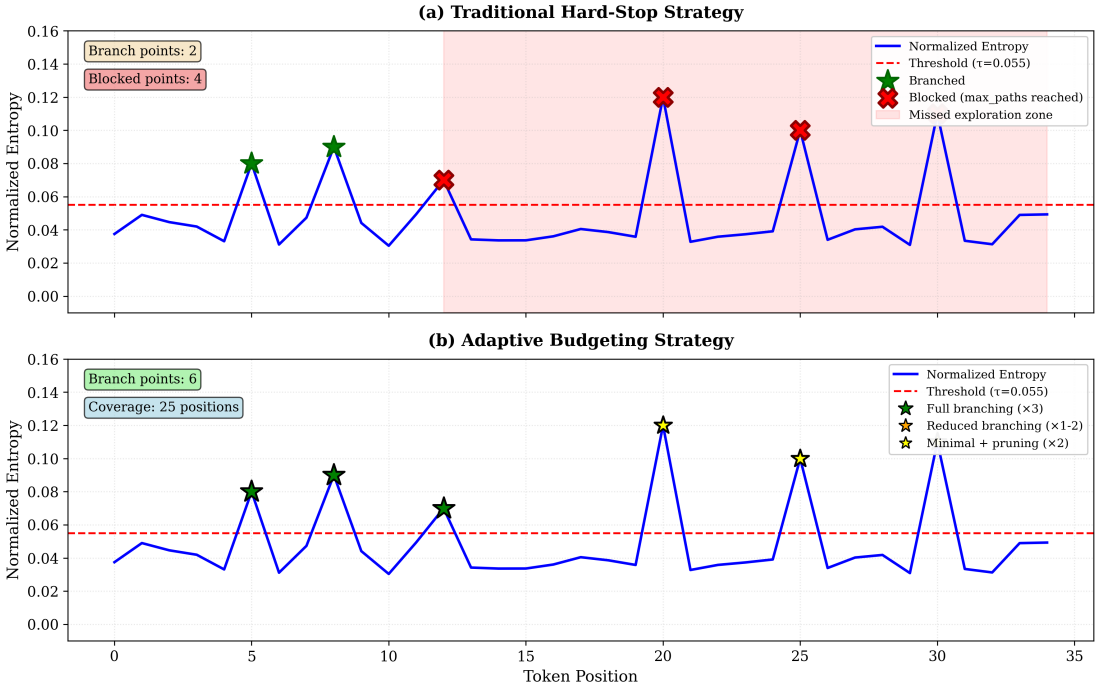


Figure 4: Comparison of path management strategies. (a) Traditional hard-stop blocks branching after M paths are reached, missing later high-entropy positions. (b) Adaptive budgeting continues exploration throughout the sequence (green/yellow stars), achieving 6 branch points vs. 2 while staying under the path limit via pruning.

Adaptive Branch Factor. To prevent exceeding the path budget while still exploring late high-entropy positions, we dynamically adjust the branch factor:

$$k' = \begin{cases} k & \text{if } M - |\mathcal{P}| \geq k \\ M - |\mathcal{P}| & \text{if } 0 < M - |\mathcal{P}| < k \\ 2 & \text{otherwise (minimal branching, rely on pruning)} \end{cases} \tag{6}$$

3.2.4 Efficiency Gains

Let L_p be prompt length and L_g generation length. Naive sampling requires $O(N(L_p + L_g))$ computation for N samples. EAB reduces this to $O(L_p + \sum_i n_i \delta_i)$, where n_i is the number of active paths during segment i of length δ_i. For long prompts or late branching, this yields substantial speedups (see Section 5).

3.2.5 Algorithm

Algorithm 1 presents the complete EAB generation procedure. The algorithm processes prompts through a chat template for instruction-tuned models (line 2), encodes once to obtain the initial KV-cache (line 3), then iteratively generates tokens while adaptively branching at high-entropy positions.

Algorithm 1 Entropy-Adaptive Branching (EAB)

Require: Prompt x, threshold τ, branch factor k, max paths M, max tokens T
Ensure: Set of generated sequences $\mathcal{Y}$

1: $\mathcal{P} \leftarrow \emptyset$ ▷ Active paths
2: $x \leftarrow \textsc{ApplyChatTemplate}(x)$ ▷ Format for instruct models
3: $\text{cache}_0, \text{logits}_0 \leftarrow \textsc{Encode}(x)$ ▷ Single prompt encoding
4: $\mathcal{P} \leftarrow \{(\emptyset, 0, \text{cache}_0)\}$ ▷ Initialize: (tokens, log-prob, cache)
5: **for** $t = 1$ to T **do**
6: $\quad \mathcal{P}' \leftarrow \emptyset$
7: $\quad$ **for** each path $(y, \ell, c) \in \mathcal{P}$ **do**
8: $\quad\quad \text{logits}, c' \leftarrow \textsc{Forward}(y_{-1}, c)$ ▷ Single token forward pass
9: $\quad\quad p \leftarrow \textsc{Softmax}(\text{logits}/\text{temperature})$
10: $\quad\quad H_{\text{norm}} \leftarrow -\sum_v p_v \log p_v \,/\, \log |V|$ ▷ Normalized entropy
11: $\quad\quad$ **if** $H_{\text{norm}} > \tau$ **then** ▷ High uncertainty: branch
12: $\quad\quad\quad k' \leftarrow \textsc{AdaptiveFactor}(k, M, |\mathcal{P}|)$ ▷ Adjust for budget
13: $\quad\quad\quad c_{\text{cow}} \leftarrow \textsc{WrapCOW}(c')$ ▷ Prepare for efficient branching
14: $\quad\quad\quad \{v_1, \ldots, v_{k'}\} \leftarrow \textsc{SampleDistinct}(p, k')$
15: $\quad\quad\quad$ **for** $i = 1$ to k' **do**
16: $\quad\quad\quad\quad \mathcal{P}' \leftarrow \mathcal{P}' \cup \{(y \circ v_i,\ \ell + \log p_{v_i},\ c_{\text{cow}}.\textsc{Branch}())\}$
17: $\quad\quad\quad$ **end for**
18: $\quad\quad$ **else** ▷ Low uncertainty: continue single path
19: $\quad\quad\quad v \leftarrow \textsc{Sample}(p)$
20: $\quad\quad\quad \mathcal{P}' \leftarrow \mathcal{P}' \cup \{(y \circ v,\ \ell + \log p_v,\ c')\}$
21: $\quad\quad$ **end if**
22: $\quad$ **end for**
23: $\quad$ Move paths ending with EOS to completed set $\mathcal{Y}$
24: $\quad$ **if** $|\mathcal{P}'| > M$ **then** ▷ Prune if over budget
25: $\quad\quad \mathcal{P} \leftarrow \textsc{TopK}(\mathcal{P}', M, \text{key} = \ell)$ ▷ Keep highest probability
26: $\quad$ **else**
27: $\quad\quad \mathcal{P} \leftarrow \mathcal{P}'$
28: $\quad$ **end if**
29: **end for**
30: **return** $\mathcal{Y} \cup \mathcal{P}$

3.3 Layer 2: Semantic Entropy (SE)

While token-level entropy captures local uncertainty during generation, it cannot detect *semantic* disagreement across complete responses. Layer 2 addresses this by clustering generated responses by meaning and computing entropy over the resulting distribution.

3.3.1 Core Idea

Token-level entropy triggers branching whenever the model faces lexical ambiguity, but not all ambiguity reflects genuine uncertainty.

Table 1: Illustrative example: Token-level entropy vs. semantic entropy for the question "What is the capital of France?". Scenario (a) shows no semantic equivalence; Scenario (b) shows partial equivalence. Semantic entropy correctly captures lower uncertainty in (b).

(a) No Semantic Equivalence

Answer	$p(s \mid x)$	$\sum_{c \in C} p(c \mid x)$
Paris	0.5	0.5
Rome	0.4	0.4
London	0.1	0.1
Entropy	0.94	0.94

(b) Partial Semantic Equivalence

Answer	$p(s \mid x)$	$\sum_{c \in C} p(c \mid x)$
Paris	0.5	0.9
It's Paris	0.4	
London	0.1	0.1
Entropy	0.94	0.33

In both scenarios, token-level entropy (computed over individual answer strings) is identical (0.94), triggering similar branching behavior in EAB. However, in Scenario (b), the responses "Paris" and "It's Paris" are semantically equivalent; they convey the same factual meaning. Clustering them into a single group reduces the effective uncertainty from 0.94 to 0.33. This demonstrates why the second semantic entropy layer is essential.

3.3.2 Embedding-Based Semantic Similarity

We measure semantic similarity by encoding responses into a shared embedding space using a pretrained sentence encoder. Two responses r_i and r_j are considered semantically similar if their cosine similarity exceeds a threshold:

$$\text{sim}(r_i, r_j) = \frac{\mathbf{e}_i \cdot \mathbf{e}_j}{\|\mathbf{e}_i\| \|\mathbf{e}_j\|} > \theta \tag{7}$$

where $\mathbf{e}_i = \text{encode}(r_i)$ is the embedding of response r_i. This captures semantic equivalence even when responses differ lexically (e.g., "The capital is Paris" $\approx$ "Paris is France's capital").

3.3.3 Agglomerative Clustering

Given N generated responses $\{r_1, \ldots, r_N\}$, we partition them into semantic clusters using agglomerative clustering with average linkage:

1. Encode all responses: $\mathbf{e}_i = \text{encode}(r_i)$
2. Compute pairwise cosine distances: $d_{ij} = 1 - \text{sim}(r_i, r_j)$
3. Merge clusters hierarchically until distance exceeds threshold δ

The number of clusters K emerges naturally, ranging from $K = 1$ (full agreement) to $K = N$ (complete disagreement).

3.3.4 Semantic Entropy Computation

Let $p_k = |C_k|/N$ be the fraction of responses in semantic cluster k. We compute normalized semantic entropy as:

$$H_{\text{sem}}^{\text{norm}} = -\frac{\sum_{k=1}^{K} p_k \log p_k}{\log K} \tag{8}$$

This yields a value in $[0, 1]$, where 0 indicates full agreement (all responses in one cluster) and 1 indicates uniform disagreement across K clusters.

3.3.5 Combined Uncertainty Score

Normalized entropy alone has a limitation: both 2 uniform clusters and 10 uniform clusters yield $H_{\text{sem}}^{\text{norm}} = 1$, yet 10 clusters indicates higher uncertainty. We address this with a combined score:

$$U = \alpha \cdot H_{\text{sem}}^{\text{norm}} + (1 - \alpha) \cdot \frac{K - 1}{N - 1} \tag{9}$$

where $\alpha = 0.6$ weights entropy and $(K-1)/(N-1)$ measures cluster diversity relative to the maximum possible.

3.3.6 Combining Both Layers

The two layers provide complementary uncertainty signals, enabling fine-grained diagnosis of model behavior:

Scenario	**Token-level**	**Response-level**
Confident & consistent	Low	Low
Confident & contradictory	Low	High
Hesitant & consistent	High	Low
Hesitant & contradictory	High	High

These categories support actionable interpretation:

- **Confident & consistent** → High reliability; suitable for automated use.
- **Confident & contradictory** → Potential hallucination or factual error: the model is certain but generates conflicting answers (e.g., "Paris" vs. "Lyon" for France's capital).
- **Hesitant & consistent** → Stylistic uncertainty: the model explores phrasing variations but agrees on meaning (e.g., "Paris is the capital" vs. "The capital is Paris").
- **Hesitant & contradictory** → Genuine ambiguity: the model is uncertain both locally and globally, suggesting the prompt is ambiguous, out-of-distribution, or requires external knowledge.

EPFL

4 Implementation

We implement our two-layer uncertainty quantification system in Python using Hugging Face Transformers [9]. The full codebase is publicly available at: DoniaGasmii/cost-aware-semantic-uncertainty-llm

This section describes key implementation decisions, data structures, and algorithms.

4.1 Layer 1: Entropy-Adaptive Branching (EAB)

4.1.1 Framework and Model Integration

We use `AutoModelForCausalLM` from the Transformers library with manual control over the generation loop. This provides direct access to:

- Raw logits for entropy computation
- KV-cache states (`past_key_values`) for prefix reuse
- Per-token log-probabilities for path scoring

For instruction-tuned models (e.g., Qwen2.5-Instruct), we apply the model's chat template to format prompts before generation. This ensures proper handling of system prompts and conversation structure, which is critical for eliciting well-calibrated responses from instruct-tuned models.
While this approach forgoes the throughput optimizations of engines like vLLM [3], it offers the fine-grained introspection required for dynamic branching and uncertainty analysis.

4.1.2 Path Representation

Each active generation path is represented by a lightweight data structure:

```
class GenerationPath:
    tokens: List[int]              # Generated token IDs
    log_prob: float                # Cumulative log P(path)
    cache: Tuple[Tensor, ...]      # KV-cache up to current step
    branch_points: List[int]       # Positions where branching occurred
    parent_id: Optional[int]       # Parent path ID (tree tracking)
    path_id: Optional[int]         # Unique path identifier
```

The initial path contains an empty token list, zero log-probability, and the KV-cache from prompt encoding. When branching occurs, child paths copy the parent's state and record the branch position. A `PathManager` maintains separate lists for active and completed paths, moving paths to the completed set upon EOS token generation.

4.1.3 KV-Cache Management

To minimize memory overhead during branching, we implement a **copy-on-write (COW)** strategy:

- When a path branches, child paths store a *reference* to the parent's cache plus their own divergent tokens.
- The full cache for a child is reconstructed on-demand by concatenating the shared prefix (from parent) with its private suffix.
- Actual tensor copying occurs only when a path modifies its cache (lazy allocation).

This reduces peak memory usage by $\sim$50% compared to eager deep copying, critical for maintaining a reasonable path budget ($M = 20$) on a single GPU.

4.1.4 Branching and Sampling Logic

At each generation step, we process all active paths sequentially:

1. For each path, run a forward pass with its cached states to obtain next-token logits.
2. Compute normalized entropy H_{norm} from the softmax distribution.
3. If $H_{\text{norm}} > \tau$ and path budget allows:
   - Sample k *distinct* tokens via multinomial sampling without replacement
   - Create k child paths, each initialized with:
     - Parent's token sequence + sampled token
     - Updated cumulative log-probability
     - COW reference to parent cache
4. Otherwise, extend the path with the highest-probability token.
5. After processing all paths, prune to top-M by log-probability if needed.

Sequential processing of paths simplifies bookkeeping for paths of varying lengths, though it sacrifices batch-level parallelism.

4.1.5 System Configuration

Default hyperparameters:

- Entropy threshold: $\tau = 0.055$
- Branch factor: $k = 3$
- Max paths: $M = 20$
- Temperature: $T = 0.8$
- Max new tokens: 50

All experiments use `Qwen/Qwen2.5-3B-Instruct`[6] unless otherwise noted, running on an NVIDIA A100 with 26GB VRAM.

4.2 Layer 2: Semantic Entropy (SE)

After EAB generates N candidate responses, we compute semantic entropy to quantify meaning-level disagreement.

4.2.1 Sentence Encoding

We use Sentence Transformers [5] to encode responses into dense vectors. Three encoder options are supported:

- `sentence-t5-xxl`: 768-dim embeddings, highest quality (default)
- `all-mpnet-base-v2`: 768-dim, good quality/speed tradeoff
- `all-MiniLM-L6-v2`: 384-dim, fastest, lower quality

Embeddings are L2-normalized to enable cosine similarity via dot product.

4.2.2 Clustering Implementation

We use scikit-learn's `AgglomerativeClustering` with the following configuration:

```
clustering = AgglomerativeClustering(
    n_clusters=None,              # Determined by threshold
    distance_threshold=0.05,      # Merge if similarity > 0.95
    metric="cosine",
    linkage="average"
)
labels = clustering.fit_predict(embeddings)
```

The distance threshold $\delta = 0.05$ corresponds to a similarity threshold of $\theta = 0.95$, requiring near-identical semantic content for cluster membership.

4.2.3 Semantic Entropy Computation

By default, we compute semantic entropy using uniform cluster weights based on response counts. Let C_k be the set of responses in cluster k, and N the total number of responses. The probability of cluster k is:

$$p_k = \frac{|C_k|}{N} \tag{10}$$

and normalized semantic entropy is:

$$H_{\text{sem}}^{\text{norm}} = -\frac{\sum_{k=1}^{K} p_k \log p_k}{\log K} \tag{11}$$

This is implemented as:

```
def compute_semantic_entropy(labels):
    cluster_counts = np.bincount(labels)
    probs = cluster_counts / len(labels)
    probs = probs[probs > 0]  # Avoid log(0)
    raw_entropy = -np.sum(probs * np.log(probs))
    return raw_entropy / np.log(len(probs))  # Normalize
```

Optional: Probability-Weighted Clustering When EAB provides path log-probabilities $\log P_i$, we can weight clusters by generation likelihood:

$$p_k = \frac{\sum_{i \in C_k} \exp(\log P_i)}{\sum_{j=1}^{N} \exp(\log P_j)} \tag{12}$$

This down-weights clusters containing low-probability outliers, focusing uncertainty estimation on likely responses, following the standard formulation of semantic entropy [2]. While this variant is implemented, our experiments use the default count-based weighting (each response contributes equally).

4.3 Two-Layer Uncertainty Quantification (EAB-SE)

The full uncertainty quantification pipeline:

1. **Generate:** EAB produces N diverse responses with token-level entropy tracked
2. **Encode:** Sentence transformer maps responses to embeddings
3. **Cluster:** Agglomerative clustering groups semantically similar responses into K clusters
4. **Quantify:** Compute normalized semantic entropy $H_{\text{sem}}^{\text{norm}}$ and combined uncertainty score U
5. **Report:** Return token-level statistics and response-level metrics $(H_{\text{sem}}^{\text{norm}}, K, U)$

5 Evaluation

We evaluate our uncertainty quantification system in three stages: (1) Layer 1 (EAB) efficiency and branching behavior, (2) Layer 2 (SE) uncertainty estimation quality, and (3) the full pipeline combining both layers. This modular evaluation isolates each component's contribution before measuring end-to-end performance.

5.1 Layer 1: Entropy-Adaptive Branching (EAB)

We first validate that entropy-adaptive branching efficiently generates diverse samples.

5.1.1 RQ1: Does EAB reduce computational cost?

Hypothesis. Entropy-adaptive branching (EAB) reduces computational cost compared to naive sampling by adaptively limiting branching in low-uncertainty regions, leading to fewer token-steps and lower wall-clock time, even if it incurs higher peak GPU memory usage due to parallel path management.

Experimental Setup. We compare EAB against naive sequential sampling at temperature $T = 0.7$ (which reflects a realistic deployment baseline where memory constraints often preclude parallel generation) on 750 synthetically generated prompts covering factual QA, creative writing, reasoning, opinion, and open-ended tasks. We use Qwen2.5-3B-Instruct as the primary model, with additional runs on 0.5B and 1.5B variants to assess scalability. For EAB, we fix the entropy threshold $\tau = 0.055$, branch factor $k = 3$, and maximum concurrent paths $m = 20$. Prompt lengths range from 43 to 293 tokens and output lengths from 10 to 100 tokens. We measure: (1) *token-step speedup* (ratio of total forward passes), (2) *wall-clock speedup* (real runtime ratio), and (3) *memory ratio* (peak GPU memory usage).

Impact of Prompt Length. Figure 5 shows EAB's performance across varying prompt lengths.

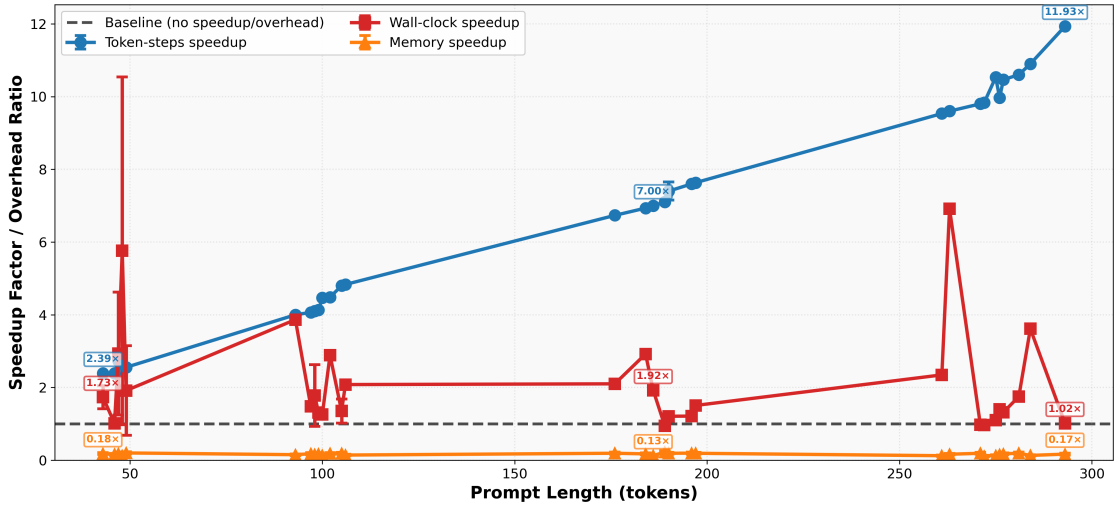


Figure 5: EAB efficiency vs prompt length. Token-step speedup (blue) scales from 2.39× to 11.93× with prompt size, wall-clock speedup (red) follows with higher variance due to GPU overhead and variable branching behavior, and memory overhead (orange) remains stable at 5–7× higher usage than naive sequential sampling.

Observation. Token-step speedup increases from 2.39× (43 tokens) to 11.93× (293 tokens). In absolute terms, naive sampling requires 890 token-steps for a 43-token prompt vs 7,160 for 293 tokens, while EAB requires only 374 and 600 respectively. Wall-clock speedup ranges from 1.73× to 6.91× with high variance.

Interpretation. EAB's prompt encoding is shared once across all samples, yielding cost $L_{\text{prompt}} + L_{\text{gen}} \times N$ versus naive's $(L_{\text{prompt}} + L_{\text{gen}}) \times N$. As L_{prompt} grows, relative savings increase. The token-step to wall-clock gap reflects real-world overhead: memory allocation, GPU kernel launches, and non-parallelizable operations.

Higher memory usage is expected as EAB maintains concurrent generation paths while naive generates sequentially, trading memory for computational efficiency.

Impact of Model Size. Figure 6 shows EAB's robustness across model scales.

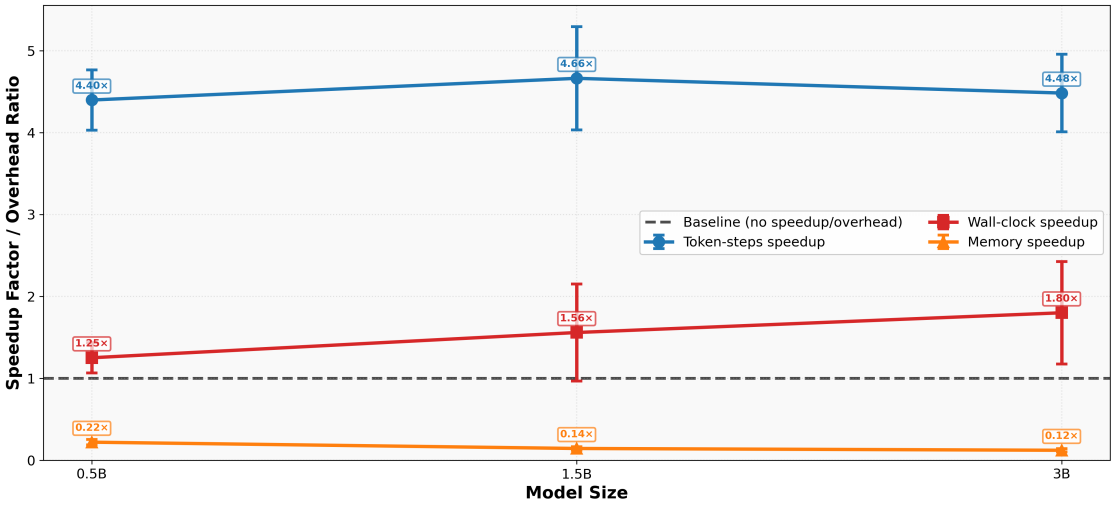


Figure 6: EAB efficiency vs model size. Speedup remains consistent across scales, with slightly improved wall-clock performance and moderately worse memory overhead for larger models.

Observation. Token-step speedup is consistent: 4.40× (0.5B), 4.66× (1.5B), and 4.48× (3B). Wall-clock speedup increases from 1.25× to 1.80× for larger models. Memory overhead is 4.5–10× higher, worsening for larger models due to increased KV cache size.

Interpretation. Consistent token-step speedup confirms model-agnostic efficiency, EAB adapts to entropy signals, not architecture. The slight improvement in wall-clock speedup for larger models suggests that GPU utilization benefits from increased compute-to-memory-bandwidth ratios in bigger transformers. Memory scaling is expected: maintaining $m = 20$ concurrent paths amplifies the base KV cache cost.

Impact of Generation Length. Figure 7 reveals when EAB provides maximum benefit.

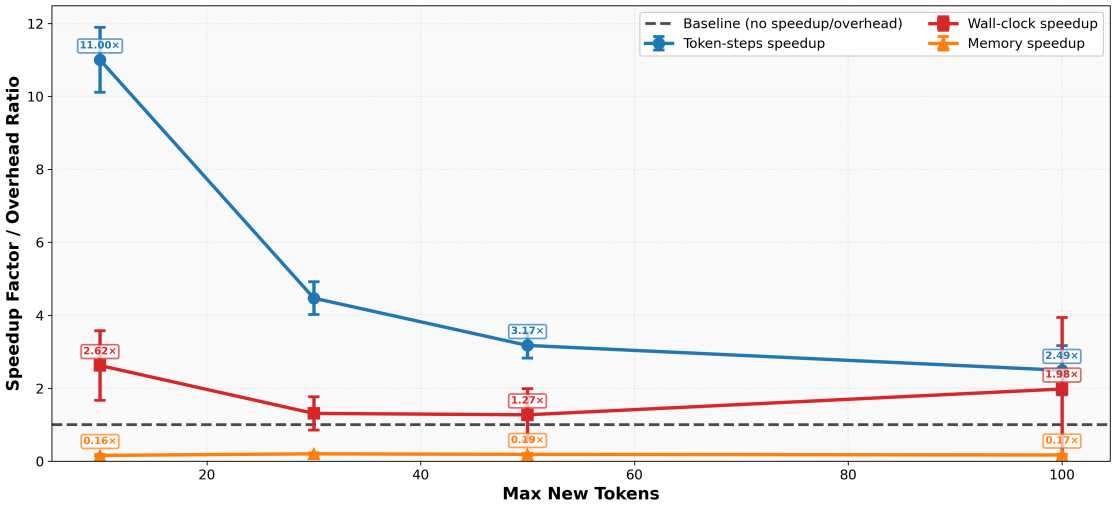


Figure 7: EAB efficiency vs generation length. Speedup decreases sharply as generation length grows longer, revealing EAB is most effective for short outputs. Memory overhead remains stable.

Observation. Token-step speedup decreases from 11.00× (10 tokens) to 4.47× (30 tokens), 3.17× (50 tokens), and 2.49× (100 tokens). Wall-clock speedup follows similarly: 2.62× (10 tokens) to 1.27× (50 tokens). Memory overhead remains constant (0.15–0.17×) across generation lengths.

Interpretation. EAB's efficiency stems from sharing prompt encoding, which becomes proportionally less significant as generation grows. For a 100-token prompt with 10 samples: generating 10-token outputs yields $(100 + 10) \times 10 = 1{,}100$ token-steps for naive vs $100 + 10 \times 10 = 200$ for EAB (5.5× speedup), but 100-token outputs yield $(100 + 100) \times 10 = 2{,}000$ token-steps for naive vs $100 + 10 \times 100 = 1{,}100$ for EAB (1.8× speedup).

RQ1 Conclusion.

EAB is most efficient for:

- Long prompts ($L_{\text{prompt}} > 50$ tokens): Speedup reaches 12× at 300 tokens.
- Short generations ($L_{\text{gen}} < 30$ tokens): Speedup exceeds 5×.
- Multiple samples needed ($N \geq 5$): Benefit compounds as prompt cost is amortized.
- Settings with sufficient GPU memory: EAB incurs 5–10× higher peak memory usage.

Model size has minimal impact on relative speedup across 0.5B–3B parameters, though we expect this may shift at larger scales. The observed gap between token-step and wall-clock gains further underscores the need for optimized KV-cache management in real-world deployment.

Our comparisons use *sequential* naive sampling as the baseline, a realistic reference for production systems where parallel generation is often infeasible due to memory constraints. Under this baseline, EAB's token-step and wall-clock speedup and memory overhead are expected trade-offs. Compared to fully parallel naive sampling, EAB would show less wall-clock advantage but lower memory consumption, positioning it as a practical middle ground between latency and memory efficiency.

5.1.2 RQ2: Does branching correlate with human ambiguity?

Hypothesis. Prompts that humans rate as ambiguous should trigger more EAB branches.

Experimental Setup. To assess alignment with human judgment, we conducted an ambiguity annotation study with 80 participants, each rating 15 diverse prompts on a 3-point scale (1 = unambiguous, 2 = moderately ambiguous, 3 = highly ambiguous). The final ambiguity score for each prompt is the average across all ratings. We ran EAB on these prompts using default parameters ($\tau = 0.055$, $k = 3$, $M = 20$, $T = 0.7$) and measured Spearman's ρ between human ambiguity scores and EAB's branching metrics (total branches and branching frequency). This human-labeled dataset enables direct correlation analysis

Results. As shown in Figures 8 and 9, EAB branching behavior strongly aligns with human-rated ambiguity.

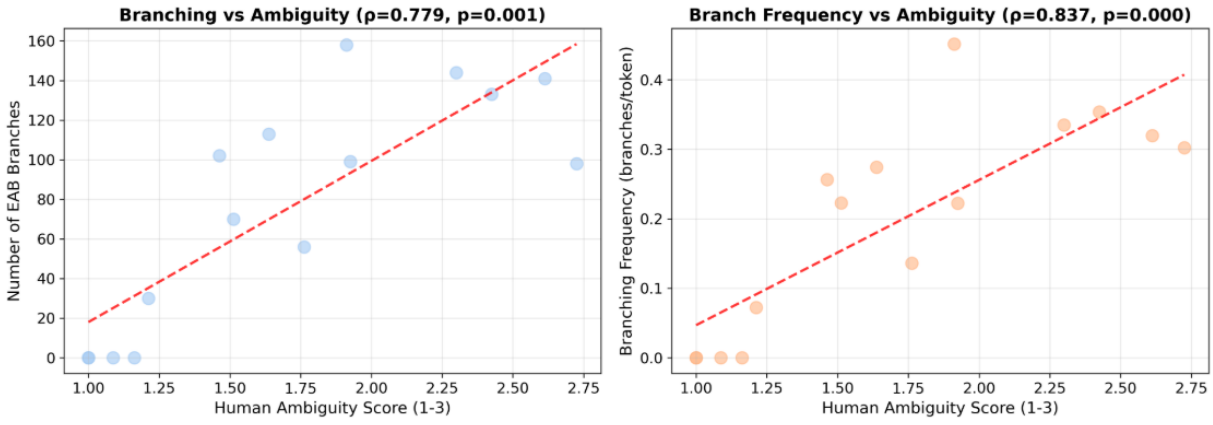


Figure 8: Scatter plots showing (left) number of EAB branches and (right) branching frequency (branches/token) vs. human ambiguity score. Spearman $\rho = 0.779$ ($p = 0.001$) and $\rho = 0.837$ ($p \geq 0.001$), respectively.

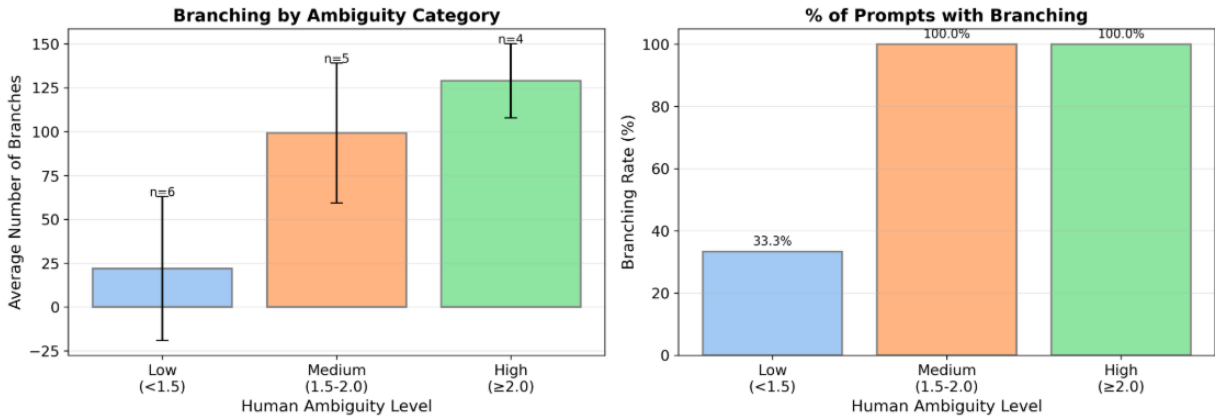


Figure 9: Group-level analysis by ambiguity category (Low: < 1.5, Medium: 1.5–2.0, High: ≥ 2.0): (left) average number of branches, (right) percentage of prompts that triggered any branching.

Observation.

- Total branches correlate strongly with human ambiguity (Spearman $\rho = 0.779$, $p = 0.001$).
- Branching frequency (branches per token) shows an even stronger correlation ($\rho = 0.837$, $p < 0.001$), indicating EAB adapts not only in quantity but also in exploration density.
- High-ambiguity prompts (≥ 2.0) averaged 130 branches vs. 25 for low-ambiguity prompts (< 1.5).
- 100% of medium/high-ambiguity prompts triggered branching, versus only 33.3% of low-ambiguity prompts.

Interpretation. Since no established benchmark exists for evaluating adaptive branching behavior, a highly niche aspect of generation dynamics, we treat human-rated ambiguity as the practical gold standard and we find that EAB's entropy-driven branching aligns closely with human intuition about ambiguity. This confirms token entropy as a perceptually meaningful uncertainty signal. The stronger correlation with branching frequency indicates EAB dynamically allocates compute precisely where uncertainty arises, at the token level.

5.1.3 RQ3: Does EAB maintain sample quality and diversity?

Hypothesis. EAB preserves or improves diversity over naive sampling by leveraging semantic entropy to avoid redundant branches, while maintaining comparable quality.

Experimental Setup. We evaluate 250 TriviaQA prompts, each generating N samples via: (1) EAB, and (2) Naive method. Diversity is measured using three metrics: *Self-BLEU* (lower = more diverse), *Distinct-n* (higher = more diverse), and *Lexical Diversity* (ratio of unique tokens to total tokens; higher = more diverse).

Results. See Figure 10 for a comparison of key diversity metrics.

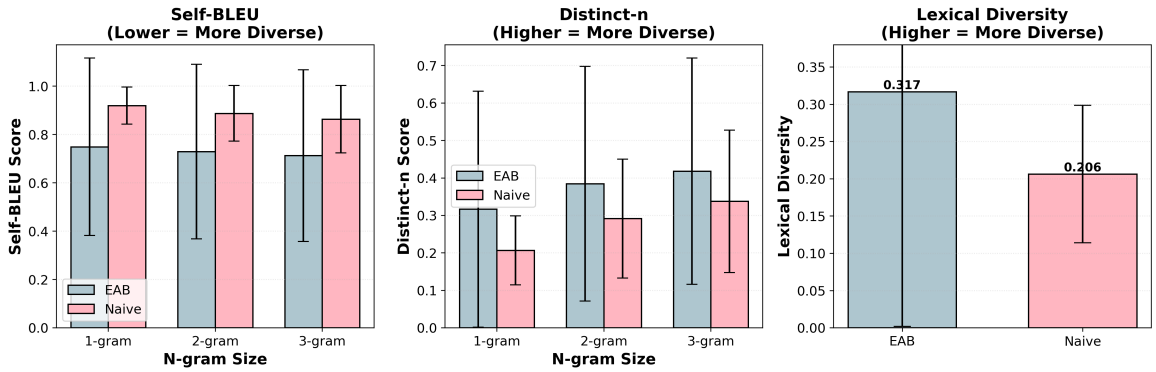


Figure 10: Diversity comparison between EAB and Naive sampling on TriviaQA ($N = 250$ prompts). EAB yields significantly lower Self-BLEU (less redundancy) and higher Distinct-2 / Lexical Diversity (richer vocabulary).

Observation. EAB achieves substantially lower Self-BLEU scores (e.g., 0.74 vs. 0.89 for 2-gram), and higher Distinct-2 (0.38 vs. 0.29) and Lexical Diversity (0.317 vs. 0.206).

Interpretation. EAB produces more diverse outputs than naive sampling by redirecting computational budget toward high-uncertainty regions, EAB enhances semantic variety.

Qualitative Example. Figure 11 shows 5 samples each from EAB and Naive for the prompt: *"Published on Feb 21, 1848, which two authors were responsible for the Communist Manifesto?"*. Though the ground truth is unambiguous, EAB produces linguistically varied yet factually consistent responses, while Naive generates near-duplicates, revealing how stochastic sampling wastes budget on surface-level repetition.

Sample Comparison: ***Communist Manifesto Authorship***

EAB Samples	Naive Samples
1. The Communist Manifesto was published on February 21, 1848, and was authored by Karl Marx and Friedrich Engels.	**1.** The Communist Manifesto was published on February 21, 1848, and was authored by Karl Marx and Friedrich Engels.
2. The Communist Manifesto was published on February 21, 1848, and was co-authored by Karl Marx and Friedrich Engels.	**2.** The Communist Manifesto was published on February 21, 1848, and was authored by Karl Marx and Friedrich Engels.
3. The Communist Manifesto was published on February 21, 1848, and was written by Karl Marx and Friedrich Engels.	**3.** The Communist Manifesto was published on February 21, 1848, and it was authored by Karl Marx and Friedrich Engels.
4. The Communist Manifesto was published on February 21, 1848, and was written by two authors: Karl Marx and Friedrich Engels.	**4.** The Communist Manifesto was published on February 21, 1848, and it was co-authored by Karl Marx and Friedrich Engels.
5. The Communist Manifesto was published on February 21, 1848, and was written by! Friedrich Engels and Karl Marx.	**5.** The Communist Manifesto was published on February 21, 1848, and was co-authored by Karl Marx and Friedrich Engels.

Figure 11: Real generated samples. EAB varies phrasing (authored/written/co-authored) and word order; Naive repeats core structure with minimal lexical change. This reflects EAB's branching at linguistic decision points (e.g., verb choice), while naive sampling re-samples the same high-probability path.

Observation. The EAB samples reveal exactly two linguistic branching points: (1) verb choice (*authored* vs. *written* vs. *co-authored*), and (2) lexical (*by Karl* vs. *by two* vs. *by!*). In contrast, Naive produces 3 near-identical outputs, confirming redundant sampling.

Interpretation. EAB's token-level entropy guides adaptive branching: it generates only as many samples as needed to resolve ambiguity at key decision points, avoiding wasted compute. For instance, when the model is highly confident (low entropy), EAB may produce just 1–2 nearly identical paths, sufficient for Semantic Entropy (SE) to recognize certainty as a stable signal. In contrast, Naive sampling with fixed $N = 20$ blindly re-samples the same high-probability sequence, yielding effectively one unique answer at 20× the cost. This redundancy explains both its lower measured diversity and higher computational overhead. Critically, EAB's adaptive output provides SE with varied samples when uncertainty exists and minimal, efficient ones when it doesn't, enabling more cost-effective uncertainty estimation.

5.2 Layer 2: Semantic Entropy (SE)

Before integrating SE with EAB, we validate that semantic entropy reliably predicts answer correctness. This ablation isolates Layer 2's contribution to uncertainty quantification.

5.2.1 Experimental Setup

Dataset. We evaluate on TriviaQA [1], a closed-book question answering benchmark where the model must answer factual questions without access to external documents. We use 250 questions from the validation split.

Model and sampling. We use Qwen2.5-3B-Instruct due to its strong instruction-following capabilities and manageable memory footprint. We generate 10 independent samples per question (i.e without EAB). Temperature $T = 0.7$ balances generating varied responses (needed to detect disagreement) while maintaining quality.

Correctness criterion. Following [2], we consider an answer correct if its RougeL score with any ground-truth answer exceeds 0.3. This fuzzy matching accounts for valid paraphrases (e.g., "Paris" vs "The capital is Paris").

Evaluation metrics. We measure Area Under the ROC Curve (AUROC), which quantifies how well uncertainty scores distinguish incorrect from correct answers. Concretely, AUROC is the probability that a randomly chosen *incorrect* answer has higher uncertainty than a randomly chosen *correct* answer:

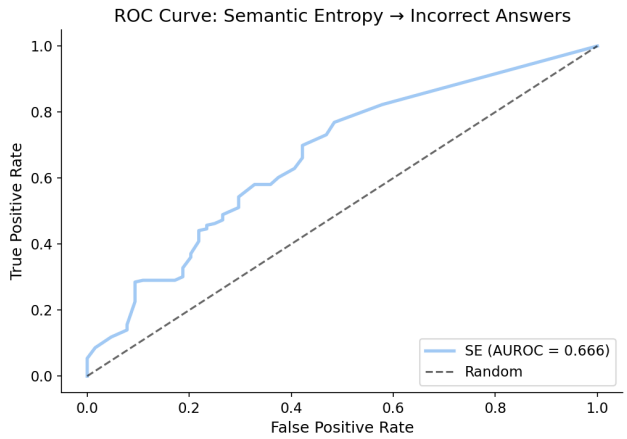
- AUROC = 0.5: Uncertainty is no better than random guessing
- AUROC > 0.5: Good discrimination, > 50% of incorrect answers rank higher
- AUROC = 1.0: Perfect separation between correct and incorrect

This metric is standard for evaluating uncertainty estimation in NLG.

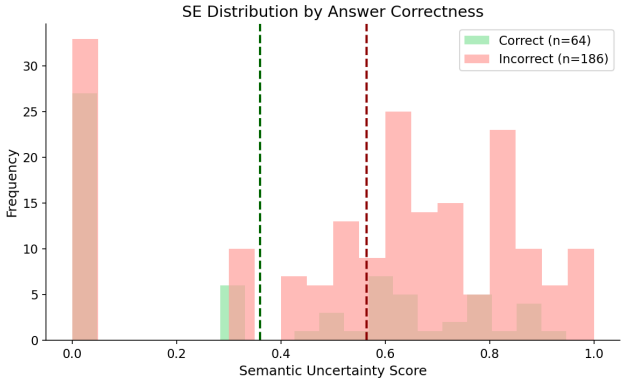
5.2.2 RQ1: Does SE predict incorrect answers?

Hypothesis. If semantic entropy captures meaningful uncertainty, questions with incorrect answers should exhibit higher SE scores (more semantic clusters, higher entropy).

Results. Figure 12 shows SE's discrimination ability.



(a) ROC curve for SE predicting incorrect answers (AUROC = 0.666).



(b) SE score distribution by correctness. Incorrect answers (n=186) have higher mean than correct answers (n=64).

Figure 12: Performance of Semantic Entropy at optimal threshold $\delta = 0.05$. SE reliably distinguishes incorrect from correct answers, with AUROC significantly above random (0.5) and clear separation in uncertainty score distributions.

Observation. At the optimal clustering threshold $\delta = 0.05$, semantic entropy achieves an AUROC of 0.666 (Figure 12a), meaning that in 66.6% of cases, a randomly selected incorrect answer has higher uncertainty than a randomly selected correct answer, significantly better than random guessing (0.5). The corresponding SE score distribution (Figure 12b) confirms this: incorrect answers ($n = 186$) are concentrated in the high-uncertainty region (SE > 0.5), while correct answers ($n = 64$) cluster near zero uncertainty.

Interpretation. SE reliably identifies uncertain predictions. The model generates more semantically diverse responses when it lacks knowledge, validating our core assumption.

5.2.3 RQ2: How does cluster count relate to correctness?

Hypothesis. Questions where all samples cluster into one semantic group (high agreement) should have higher accuracy than questions with many clusters (disagreement).

Results. Figure 13 shows accuracy stratified by the number of semantic clusters. Table 2 summarizes performance across aggregated groups.

Table 2: Accuracy by number of semantic clusters ($\delta = 0.05$, n=250).

Clusters	n	Accuracy	Interpretation
1 (full agreement)	60	45.0%	High confidence
2–3	67	25.4%	Moderate uncertainty
4+	123	16.3%	High uncertainty
Overall	250	25.6%	—

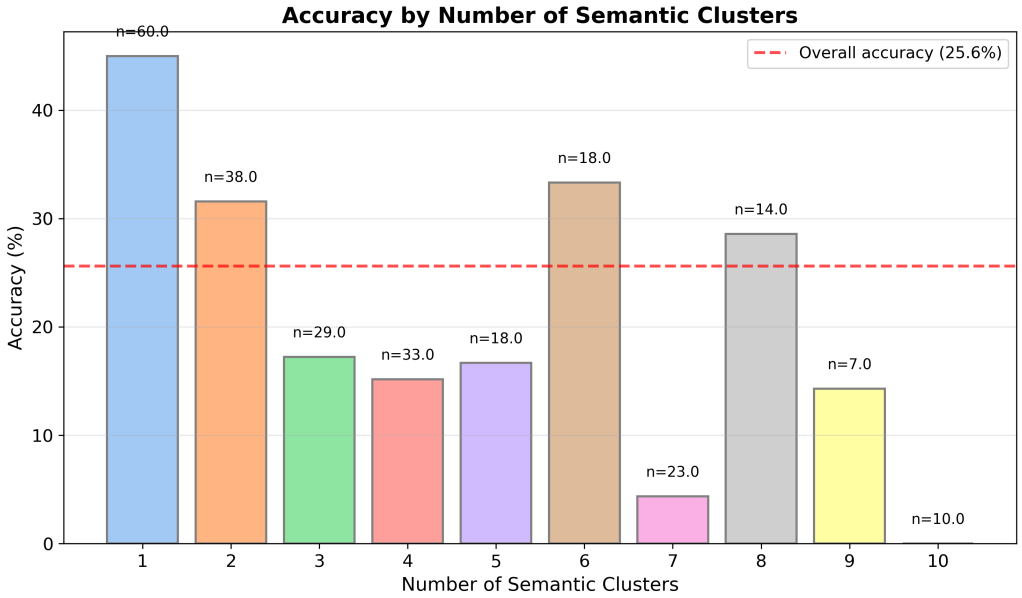


Figure 13: Accuracy by number of semantic clusters. The dashed red line indicates overall accuracy (25.6%). Accuracy decreases as cluster count increases, confirming that semantic disagreement correlates with lower correctness.

Observation. Questions with 1 cluster achieve 45.0% accuracy, compared to 16.3% for questions with 4 or more clusters, a 28.7 percentage point gap. Even within the 2–3 cluster group, accuracy (25.4%) is close to the overall mean, suggesting moderate uncertainty.

Interpretation. The number of semantic clusters provides an intuitive, interpretable signal of model confidence: unanimous agreement (1 cluster) strongly correlates with correctness, while high disagreement (4+ clusters) signals likely error. This validates our assumption that clustering-based diversity serves as a proxy for epistemic uncertainty. Crucially, it also justifies our design choice to incorporate *both* semantic entropy *and* cluster count into the final uncertainty score, since each captures complementary aspects of response diversity (distributional spread vs. structural fragmentation).

5.2.4 RQ3: Sensitivity to clustering threshold

Hypothesis. AUROC should be relatively stable across reasonable threshold values, but extreme thresholds (too strict or too loose) may degrade performance.

Results. Figure 14 shows the sensitivity of AUROC and average cluster count to the clustering distance threshold δ.

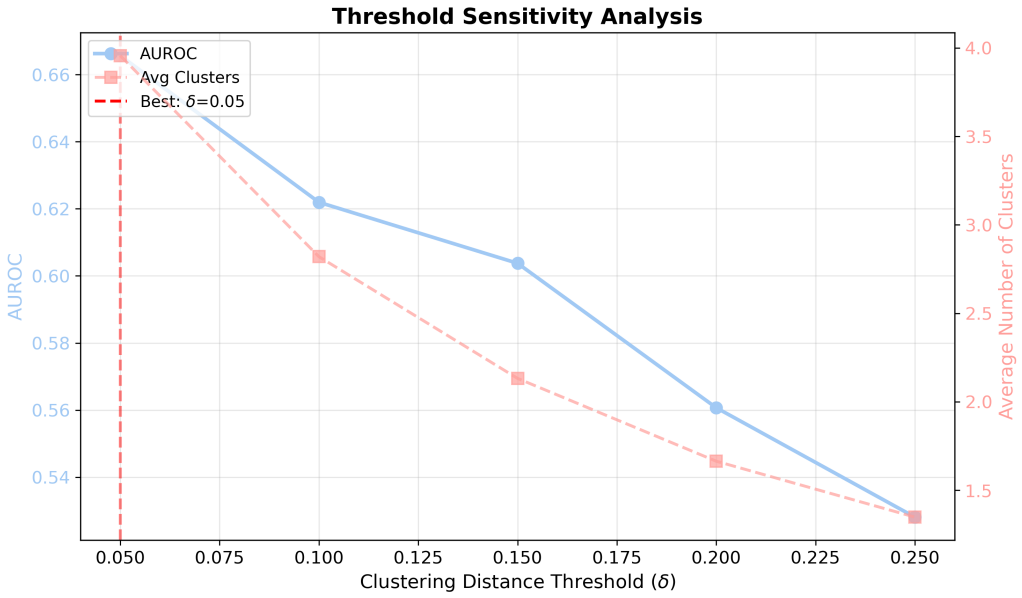


Figure 14: Sensitivity of AUROC and average number of clusters to clustering threshold δ. The optimal AUROC (0.666) occurs at $\delta = 0.05$, where semantic disagreement is most finely resolved. As δ increases, clusters merge, reducing both diversity and uncertainty signal quality.

Observation. AUROC peaks at 0.666 when $\delta = 0.05$, then steadily declines to 0.528 at $\delta = 0.25$. The average number of clusters decreases from 3.96 to 1.35 over the same range, indicating that stricter thresholds preserve semantic diversity while looser ones collapse distinct meanings into single clusters.

Notably, the optimal threshold appears to depend on the base model's reliability. In our setting, we use Qwen2.5-3B-Instruct, a capable but limited 3B-parameter model whose overall accuracy on TriviaQA remains modest (25.6%). Under such conditions, even minor inconsistencies in generations may reflect genuine epistemic uncertainty rather than stylistic variation. A stricter threshold ($\delta = 0.05$) thus better isolates these meaningful disagreements, improving the signal-to-noise ratio of SE. We hypothesize that for stronger, more accurate models (e.g., 7B+ or instruction-tuned variants), a looser threshold might suffice, as generations would be more coherent and semantically aligned when correct. This suggests that the clustering threshold should be calibrated to the model's intrinsic reliability, rather than treated as a universal hyperparameter.

Interpretation. Semantic entropy is sensitive to the clustering threshold, with performance degrading as δ increases. The peak at $\delta = 0.05$ confirms that fine-grained semantic distinctions carry meaningful uncertainty signals for less reliable models. Crucially, this implies that the threshold is not a fixed hyperparameter, but a model-dependent calibration knob, a finding that informs robust deployment of SE across diverse LLMs.

5.3 Two-Layer Uncertainty Quantification (EAB-SE)

We evaluate the full pipeline a primary final research questions

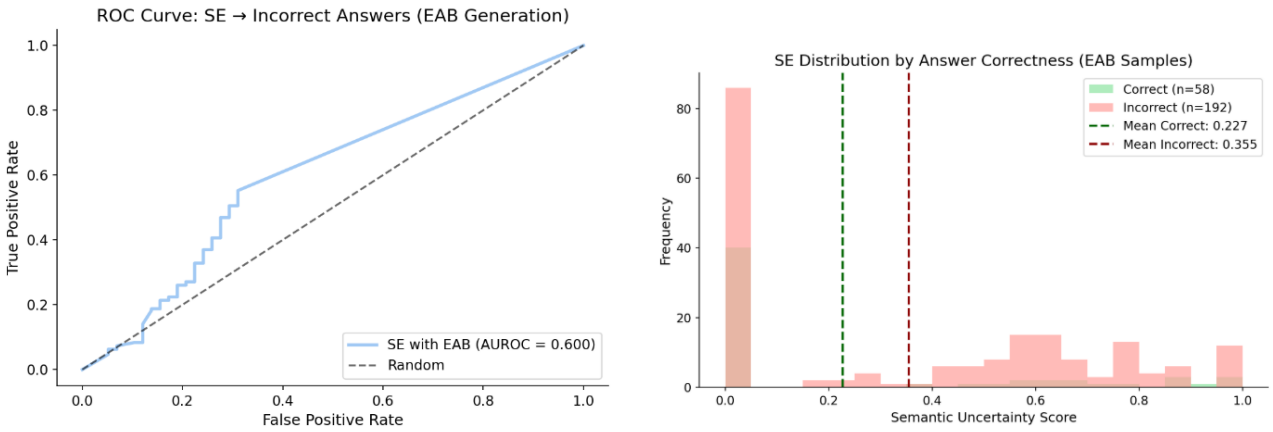
5.3.1 RQ1: Does EAB-generated Diversity Maintain SE Quality?

Hypothesis: Entropy-Adaptive Branching produces semantically diverse samples that are as effective (or more effective) for Semantic Entropy uncertainty quantification as samples from naive sampling.

Experimental Setup To evaluate whether EAB's adaptive generation strategy maintains the quality of SE-based uncertainty estimation, we compare the full EAB+SE pipeline against SE with naive sampling (Sub-

Section 5.2). Both approaches use identical semantic similarity clustering parameters to isolate the effect of the generation strategy. As for EAB we configureit with entropy threshold ($\tau = 0.05$), branch factor is set to $k = 3$ with a maximum of $M = 20$ concurrent paths. Temperature is set to $T = 0.7$ with nucleus sampling ($p = 0.9$).

Results Figure 15 presents the key results for EAB+SE with the optimal clustering threshold ($\delta = 0.05$).



(a) ROC curve (AUROC = 0.61)

(b) SE uncertainty distributions

Figure 15: EAB+SE performance on TriviaQA. The left figure shows the ROC curve for uncertainty-based error detection, while the right figure displays the distribution of SE uncertainty scores for correct versus incorrect answers. The uncertainty scores show partial separation between correct and incorrect answers, but discrimination is weaker than SE-only baseline.

Observation. The EAB+SE pipeline achieves an AUROC of 0.61 (with clustering threshold $\delta = 0.05$), compared to the SE-only baseline AUROC of 0.66 from Sub-Section 5.2. This reduced discriminative power represents a degradation in uncertainty estimation quality. The mean SE uncertainty score for incorrect answers ($\mu_{\text{wrong}} = 0.355$) is higher than for correct answers ($\mu_{\text{right}} = 0.227$), maintaining the expected directional relationship, but with reduced separation compared to naive sampling ($\mu_{\text{wrong}} = 0.564$ vs. $\mu_{\text{right}} = 0.360$). This compression reduces the dynamic range available for uncertainty discrimination, potentially explaining the degraded AUROC.
EAB generates an average of 13.3 samples per question (compared to the fixed 20 samples in naive sampling), producing 4.3 semantic clusters on average. The question-level accuracy is 23.2%, slightly lower than the SE-only baseline (25.6%), though this difference may be within normal variance.

Interpretation. The results **reject** the hypothesis that EAB maintains SE quality.
We suspect this is explained by EAB's token-by-token branching creating systematic dependencies between generated sequences: samples that share early branching points will share prefixes and are likely to cluster together regardless of semantic meaning. This introduces a *clustering bias* where semantic similarity reflects the branching tree structure rather than true semantic diversity. This path-dependency effect suggests a fundamental tension: while EAB successfully generates diverse tokens (as measured), this diversity does not translate into the kind of semantic diversity that SE relies upon for robust uncertainty quantification.

Conclusion. These findings suggest that combining EAB with SE in a pipeline may not be optimal. EAB's sampling strategy, while cost-efficient through adaptive generation, appears to corrupt the semantic signal that SE exploits. Alternative integration strategies, such as using EAB's internal entropy estimates directly or developing clustering methods robust to path-dependencies, may be necessary for effective two-layer uncertainty quantification.

6 Conclusion

EPFL

References

[1] Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611. Association for Computational Linguistics.

[2] Kuhn, L., Gal, Y., and van der Wilk, M. (2023). Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. *arXiv preprint arXiv:2302.09664*.

[3] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zhu, X., Gonzalez, J. E., Stoica, I., and Zaharia, M. (2023). vLLM: Easy, Fast, and Cheap LLM Inference with PagedAttention. Available at https://github.com/vllm-project/vllm.

[4] Lin, Z., Trivedi, S., and Sun, J. (2023). Generating with confidence: Uncertainty quantification for black-box large language models. In *Transactions on Machine Learning Research*.

[5] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

[6] Team, Q. (2024). Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[8] Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244.

[9] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.