

Local Backtracking Mechanisms in Generative Modelling

Máté Norbert Molnár, Tibor Tajti

2025

Motivation

Large language models (LLMs) show impressive natural-language understanding and generation abilities, yet **standard decoding is irreversible**. Once a token x_t is chosen it permanently stays in the context, so early mistakes propagate and may lead to

- repetitions or incoherence,
- deviation from the user's intent,
- generally lower output quality.

We investigate a *conditional backtracking operator* that can retract the most recent n tokens when intermediate quality signals suggest a poor trajectory.

Definitions

Let M be a pre-trained autoregressive Transformer [3] with vocabulary \mathcal{V} . At generation step $t = 1, 2, \dots$

$$z_t = M(x_{1:t-1}), \quad p_t = \text{softmax}(z_t), \quad x_t \sim p_t,$$

producing the sequence $x_{1:t}$.

Hypothesis

Problem

- The sequence is fixed once emitted; no native mechanism exists to correct bad choices.

Idea

Introduce a **local backtracking operator**: at selected checkpoints remove the last n tokens if a decision function predicts higher expected quality after rollback.

Conditional Backtracking – formalism

Choose a checkpoint period $\kappa \in \mathbb{N}$. For every step t with $t \bmod \kappa = 0$ compute

$$\delta : \mathbb{R}^{|\mathcal{V}|} \times \Delta_{|\mathcal{V}|} \times \mathcal{V} \times \{0, \dots, |\mathcal{V}| - 1\} \rightarrow \{0, \dots, t\},$$

$$n_t = \delta(z_t, p_t, x_t, i_t; \Theta),$$

where Θ collects hyper-parameters and i_t is the index of x_t .

Conditional Backtracking – execution

If $n_t > 0$:

- delete suffix $x_{t-n_t+1:t}$;
- roll back the Transformer's key/value cache accordingly;
- resume decoding from the shortened prefix.

Implemented decision functions

We experimented with several heuristics:

- Probability threshold
- Entropy threshold
- Absolute probability floor
- Probability drop ratio
- Probability trend
- Repetition detection
- N-gram overlap
- Logit threshold

Benchmark and protocol

- Dataset: **HumanEval** [1].
- Model: **Qwen2.5 family** [2].
- Metric: percentage of correctly synthesised programs (Pass@1).
- Generation hyper-parameters (temperature, top-p, etc.) kept constant.

HumanEval example

```
def truncate_number(number: float) -> float:
    """ Given a positive floating point number, it can be decomposed into
    and integer part (largest integer smaller than given number) and decimals
    (leftover part always smaller than 1).

    Return the decimal part of the number.
    >>> truncate_number(3.5)
    0.5
    """
    ----
    return number - int(number)
```

Figure: One task from the HumanEval benchmark

Results – improvement per decision function

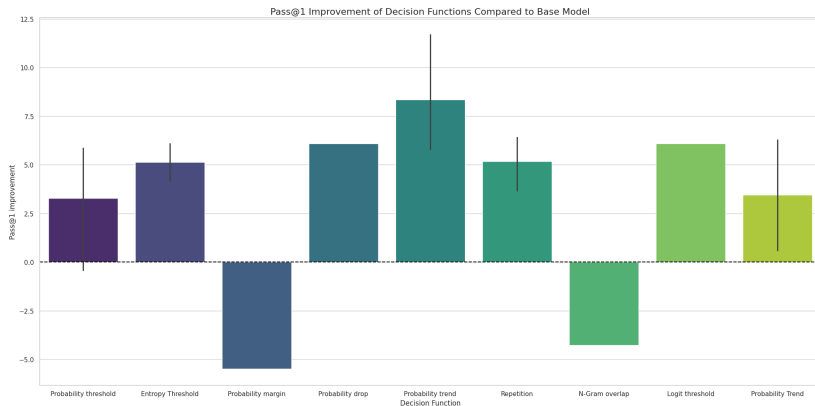


Figure: Pass@1 Improvement of Decision Functions Compared to Base Model

Results – improvement per model

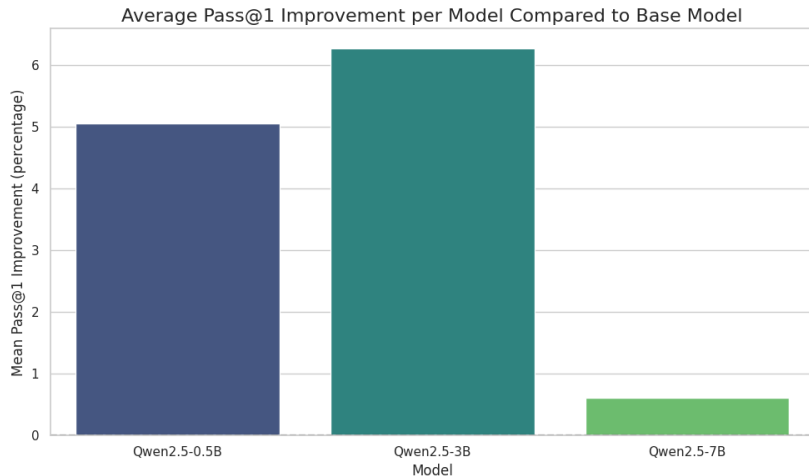


Figure: Average Pass@1 Improvement per Model

Results – generated final tokens per second

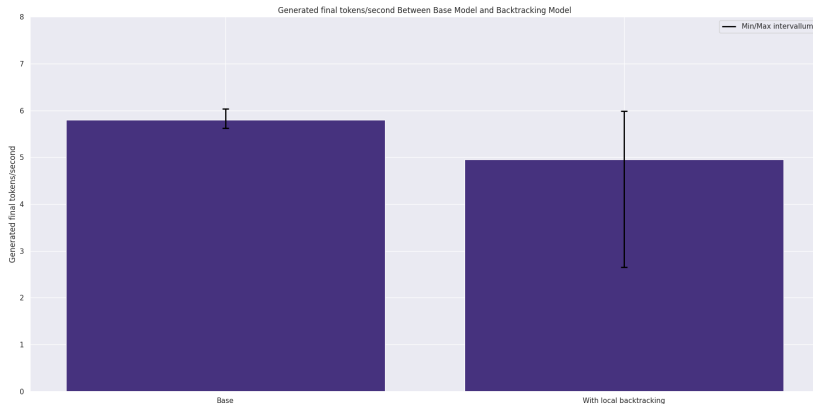


Figure: Generated final tokens/second Between Base Model and Backtracking Model

Future work

- Systematic hyper-parameter optimisation.
- Learn δ with a small neural network instead of hand-crafted heuristics.

Thank you for your attention!