AutoJudge: Judge Decoding Without Manual Annotation

Roman Garipov* [1,2,3] Fedor Velikonivtsev* [1,2] Ivan Ermakov [1,2]
Ruslan Svirschevski [3] Vage Egiazarian [4] Max Ryabinin [5]
[1]HSE University [2]Yandex Research [3]ITMO University [4]IST Austria [5]Together AI

Our Approach

Setup: solving problems with LLM inference, e.g. Chain-of-Thought math reasoning or programming. LLM generation is sequential, predicting one token at a time.

Speculative Decoding [1]: generate tokens faster with an auxiliary "draft" model — a faster but less accurate LLM:

- **Drafting phase:** use the fast θ_{draft} model to generate multiple next tokens (draft tokens).
- **Verification phase:** run the main θ_{target} model in parallel on draft tokens to compare next token predictions. Return tokens up to the first mismatch. Repeat.

Not all mismatches affect the final answer (Figure 1 →)
Important: errors in formula, program logic & syntax.
Unimportant: word choice, formatting, minor notation.

Judge Decoding [2]: ask humans to judge if a mismatch is important (affects the final answer) or not. Train a linear classifier to detect "unimportant" tokens from LLM hidden state at inference time. During verification, accept mismatching tokens if they are unimportant for the task.

AutoJudge (Ours): Fire humans. Find important tokens automatically with a semi-greedy search (Algorithm 1 →) For each mismatch between target and draft models, start with the draft token and see if this affects response quality. If it does, then the token was important. If response quality did not change, the mismatch is unimportant: accept it.

Response quality is task-specific. For math reasoning, two responses are equivalent if they have the same final answer (up to notation). For programming, the two programs must pass the same tests. Can be extended: for general QA, let LLM "judge" decide if responses are the same.

Train classifier: collect training data via Algorithm 1, fit a 2-class Logistic Regression to predict whether the current token is important based on LLM hidden states.

Inference: run speculative decoding normally until the first mismatch. If the classifier deems that the mismatch is unimportant - accept that draft token and any future matching tokens. Easy to integrate: see reference vLLM [3] implementation. Compatible with any speculative decoding: we test vanilla and EAGLE-2 [4].

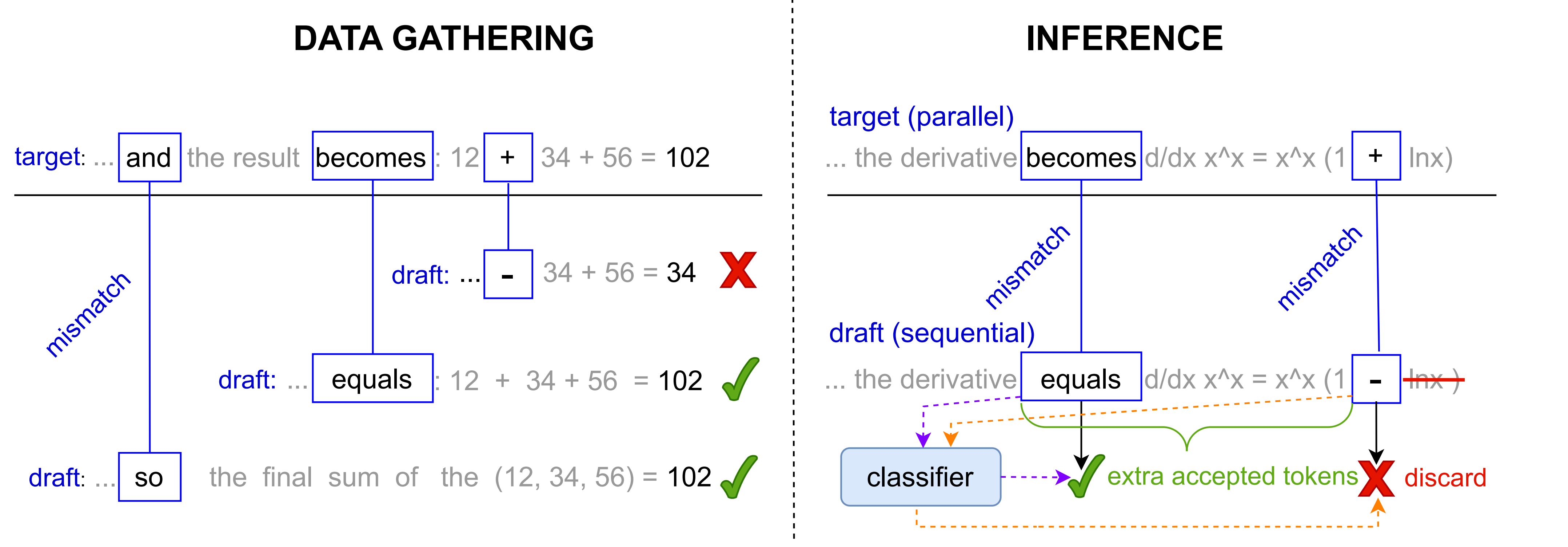
DATA GATHERING
INFERENCE
target: ... and the result becomes: 12 + 34 + 56 = 102
draft: ... - 34 + 56 = 34
draft: ... equals : 12 + 34 + 56 = 102
draft: ... so the final sum of the (12, 34, 56) = 102
mismatch
target (parallel)
... the derivative becomes d/dx x^x = x^x (1 + lnx)
draft (sequential)
... the derivative equals d/dx x^x = x^x (1 - lnx)
mismatch
mismatch
classifier
extra accepted tokens
discard

Figure 1. AutoJudge pipeline: **(left)** data gathering: detect mismatching tokens that affect final response quality; these tokens are then used to train a classifier **(right)** use the trained classifier to generate more tokens per cycle with speculative decoding.

Algorithm 1. Search for Important Tokens. The algorithm is used during the data gathering stage (Figure 1 **left**) to label which of the mismatching tokens affect the final answer.

```
1:  Input: $x$: prompt, $\theta_{\text{draft}}, \theta_{\text{target}}$: draft & target models
2:  Output: labeled mismatching tokens $\mathcal{M}$
3:  $\mathcal{M} \leftarrow \emptyset$   ▷ Tuples (position, tokens, is important?)
4:  $y \leftarrow \text{generate}(x, \theta_{\text{target}})$
5:  $\alpha \leftarrow \text{extractAnswer}(y)$
6:  $\widetilde{y} \leftarrow \text{forward}(x \oplus y, \theta_{\text{draft}})\texttt{.argmax(-1)}_{\texttt{[len(x)-1:-1]}}$
7:  $\mathcal{I} \leftarrow \{i \mid y_i \neq \widetilde{y}_i\}$   ▷ Indices where models mismatch
8:  while $\mathcal{I} \neq \emptyset$ do
9:      $t \leftarrow \min(\mathcal{I})$   ▷ Find the first unlabeled mismatch
10:     $\hat{y} = y_{1:t} \oplus \widetilde{y}_t \oplus \text{generate}(x \oplus y_{1:t} \oplus \widetilde{y}_t, \theta_{target})$
11:     $\hat{\alpha} \leftarrow \text{extractAnswer}(\hat{y})$ ▷ Alternative answer with $\widetilde{y}_t$
12:     if $\alpha \equiv \hat{\alpha}$ then
13:         ▷ Equivalent answer, token $y_t$ is not important
14:         $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t, y_t, \widetilde{y}_t, \texttt{False})\}$
15:         $y \leftarrow \hat{y}$ ▷ Continue search from the new response
16:         $\widetilde{y} \leftarrow \text{forward}(x \oplus y, \theta_{\text{draft}})\texttt{.argmax(-1)[len(x)-1:-1]}$
17:     else
18:         ▷ Different answer, token $y_t$ is important, ...
19:         $\mathcal{M} \leftarrow \mathcal{M} \cup \{(t, y_t, \widetilde{y}_t, \texttt{True})\}$   ▷ ...so we keep it
20:     end if
21:     $\mathcal{I} \leftarrow \{i | y_i \neq \widetilde{y}_i \cap i > t\}$ ▷ Keep mismatches past $t$
22: end while
23: return $\mathcal{M}$
```

Table 1. Examples of mismatching tokens with [alternatives] by Algorithm 1: labeled as important or unimportant. The first example shows alternative responses (✓ if $\alpha \equiv \hat{\alpha}$, ✗ if not).

```
[GSM8K] Arnel had ten boxes of pencils ... how many
pencils are in each box? A: Arnel kept ten pencils
and shared the remaining pencils with his 5 friends.
 [.] He shared the ... ✓        [equally] with ... ✓
(continued) This means that the total number of
pencils he shared is 10 * x - 10. ...
      [Arnel] ... ✓     [-] x - 10 ... ✗
```

```
[GSM8K] Adlai has 2 dogs and 1 chicken. How many
animal legs are there in all? A: To find the total
number of animal legs, we need to calculate the
legs [total] of each animal and then add them up.
- 2 dogs have 4 [2] legs each, so 2 dogs have 2
* [times] 4 = 8 legs. - 1 chicken has 2 legs. ...
```

```
[LCB]Given a string S of lowercase...If there are ad-
jacent occurrences of a and b in S, print Yes; ...
def[#] solve[check](s):
  for i in range(len(s) -[)] 1):
    if s[i] == 'a' and s[i+1] == 'b':
      return "Yes"
    if s[i] == 'b[a]' and s[i+1] == 'a':
      return "Yes"
  return "No"
```

Experiments

We evaluate AutoJudge on math reasoning with GSM8K [5] and programming on LiveCodeBench v5 [6], focusing on two model families: Llama 3.x [7] and Qwen2.5 [8]. In Figures 3 and 2, we compare the average number of accepted tokens (per verification phase) for different classifier settings, showing that AutoJudge produces significantly more tokens per speculation with negligible loss in accuracy and can further speed up decoding at the cost of some accuracy drawdown. Tables 2 and 3 show real-world speedups on A100 GPUs with both on-device inference and offloading. Both setups show significant speed-up over vanilla speculative decoding. Additional setups and evaluations, including larger models and EAGLE-2 acceleration can be found in the paper.

Figure 2. LiveCodeBenchsv5 Pass@1 and accepted tokens on for Llama 3.1 8B draft / Llama 3.1 70B target

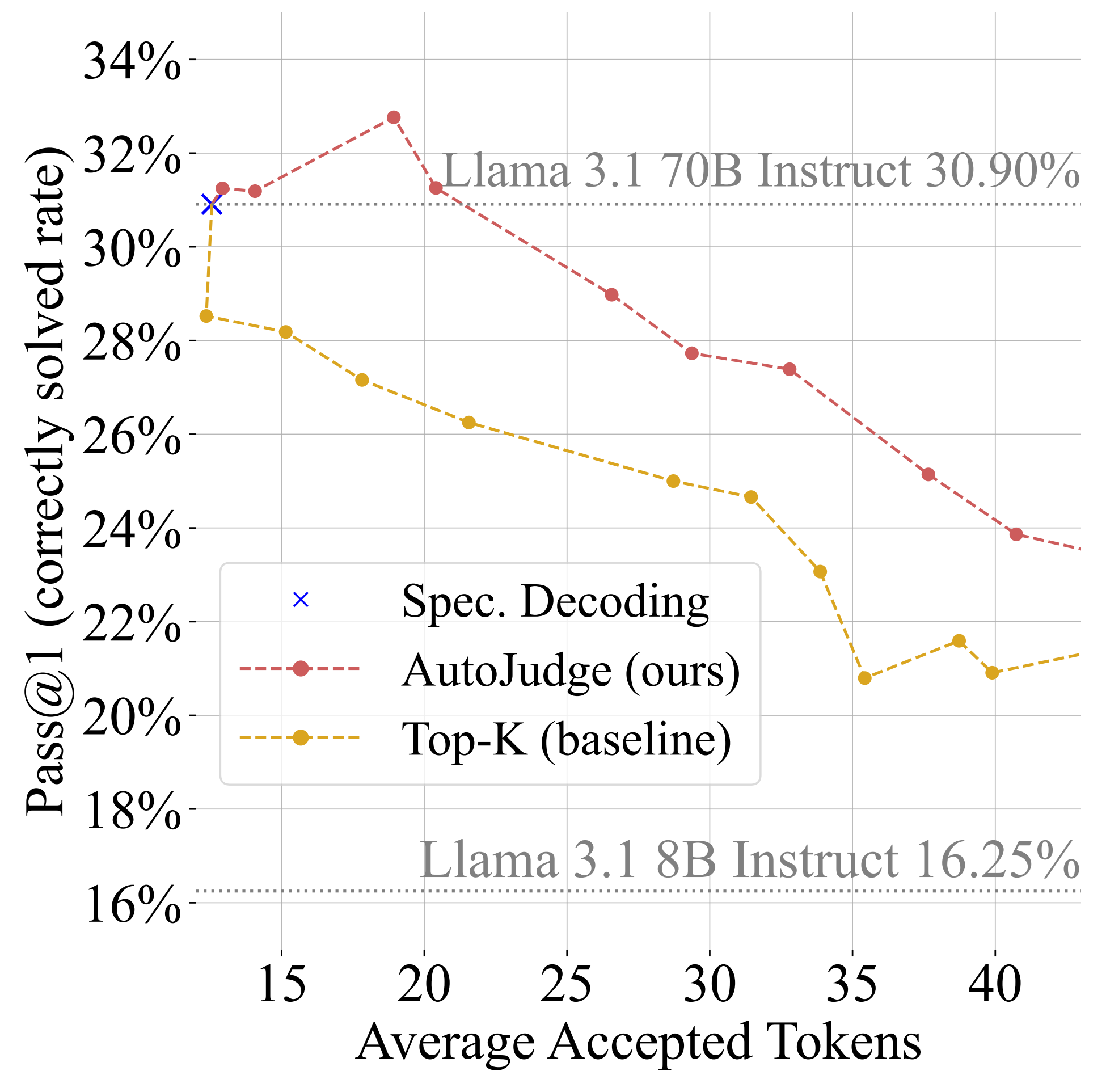

Figure 3. Accuracy and accepted tokens on GSM8K for **(top)** Llama 3.1 Instruct 8B draft / 70B target and **(bottom)** Qwen 2.5 Instruct 0.5B draft / 7B target.

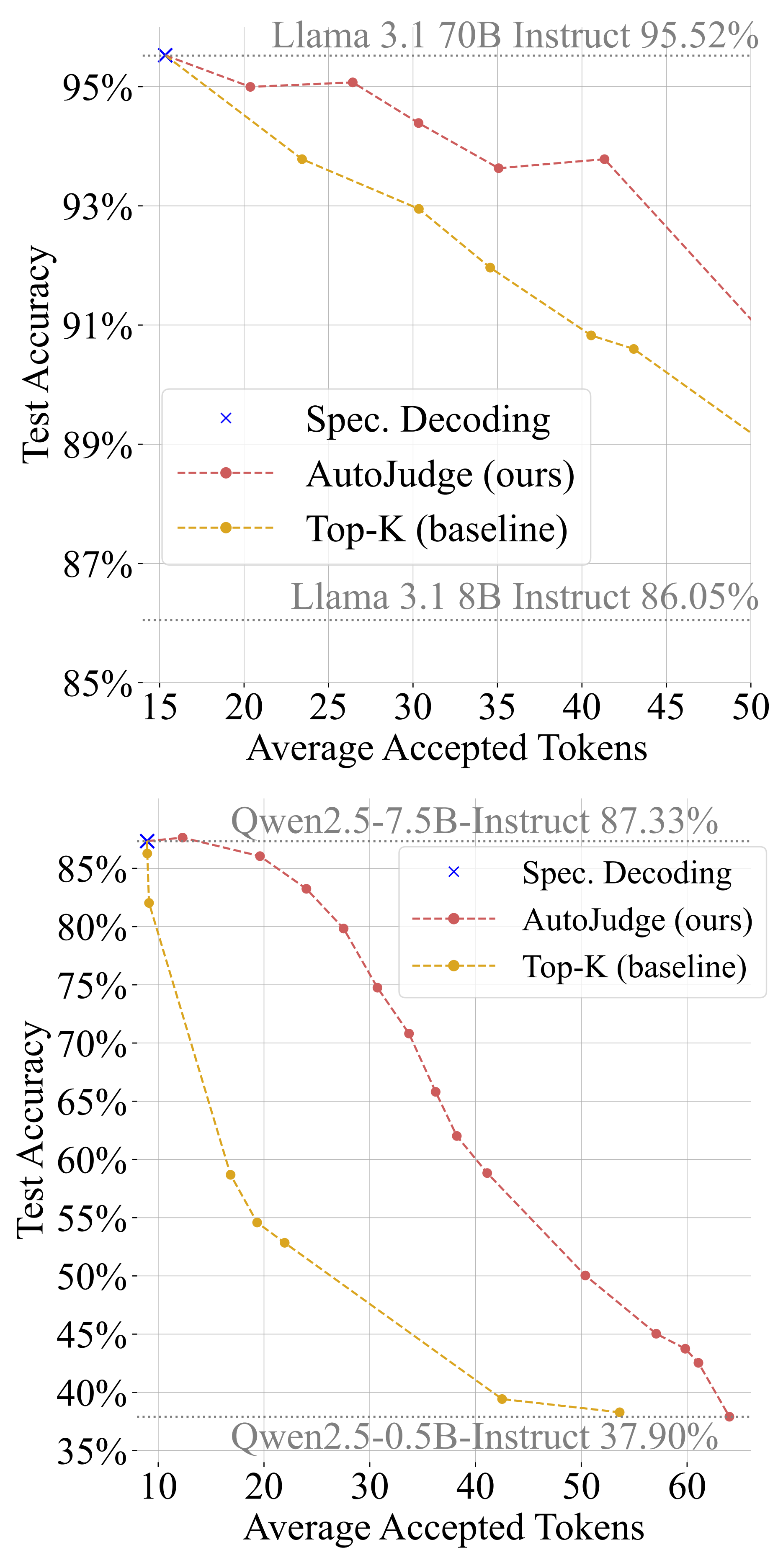

Table 2. Inference benchmarks with vLLM on the GSM8K dataset for Llama 3.1 8B Instruct draft / 3.1 70B target.

Llama 3.1 8B draft / 3.1 70B target				
Threshold	0.005	0.031	**0.145**	0.230
Accuracy, %	92.0	91.9	**89.9**	88.0
Speed, tokens/s	63.8	94.6	**108.5**	108.0
Speculative Decoding: 70.9 tokens/s				
Speedup(ours)	0.90	1.33	**1.53**	1.52

Table 3. Inference benchmarks with vLLM on the GSM8K dataset for Llama 3.1 8B Instruct draft / 3.1 70B target with offloading to CPU.

Llama 3.1 8B draft / 3.1 70B target				
Threshold	0.03	0.05	0.11	**0.18**
Accuracy, %	95.4	94.8	93.4	**92.9**
Speed, tokens/s	0.7	0.7	0.9	**1.1**
Speculative Decoding: 0.62 tokens/s				
Speedup(ours)	1.13	1.15	1.47	**1.76**

References

[1] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.

[2] Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas K Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *The Thirteenth International Conference on Learning Representations*, 2025.

[3] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[4] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024.

[5] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv:2110.14168*, 2021.

[6] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024.

[7] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[8] Qwen Team. Qwen2.5 technical report. *arXiv:2412.15115*, 2024.