Validator-Guided Decoding for Multi-Hop Knowledge Editing

Anonymous ACL submission

Abstract

Knowledge editing aims to update specific factual knowledge in large language models without full retraining. Although existing editors can often modify local factual associations, edited models still struggle to apply updated knowledge in multi-hop reasoning. In this work, we study this problem through the lens of the generate-verification gap: a model may fail to directly generate the correct updated answer, while a verifier conditioned on the edit facts can still judge whether a proposed answer is compatible with the updated knowledge. Motivated by this observation, we propose **V**alidator-**G**uided **D**ecoding (**VGD**), a test-time method that uses edit facts as verification memory to improve answer generation after knowledge editing. VGD employs a trained validator as an external correctness signal over candidate answers, allowing the model to retain reliable outputs while selectively revising uncertain or inconsistent predictions. It is editor-agnostic and can be applied to base models or on top of existing editors without modifying the editor or further updating the edited model. Experiments on multi-hop knowledge editing benchmarks show that VGD improves both hop-level factual consistency and final multi-hop answer accuracy, outperforming direct ICL and CoT-style prompting baselines and enabling more reliable use of updated knowledge in complex reasoning. Our code will be available.

1 Introduction

Large language models (LLMs) have shown strong capabilities in language understanding, generation, and reasoning (Touvron et al., 2023; OpenAI, 2023; DeepSeek-AI et al., 2025; Yang et al., 2024; Wei et al., 2022), but their factual knowledge is inevitably incomplete, outdated, or incorrect, making full retraining an impractical solution for targeted updates. Knowledge editing (KE) addresses this problem by modifying specific factual associations while preserving unrelated behaviors (De Cao et al., 2021; Mitchell et al., 2022; Yao et al., 2023; Qiao et al., 2024b; Gupta et al., 2024). Existing methods range from parameter-preserving approaches, which rely on in-context evidence or external memory to steer predictions (Zheng et al., 2023; Zhong et al., 2023), to parameter-updating approaches, which directly alter model weights through learned updates or locate-then-edit mechanisms (Meng et al., 2022, 2023; Li et al., 2024; Fang et al., 2025; Jiang et al., 2025). The latter are especially appealing because the edited knowledge is stored in the model itself, avoiding persistent retrieval or prompting overhead while aiming to maintain locality and general ability (Gu et al., 2024; Gupta et al., 2024; Qiao et al., 2025; Ma et al., 2025). However, success on local factual recall does not guarantee reliable use of edited knowledge in compositional settings. Recent studies show that edited models often fail to propagate updates through multi-hop queries, exhibit shortcut reasoning, or access the relevant knowledge too late during generation (Hoelscher-Obermaier et al., 2023; Cohen et al., 2024; Qin et al., 2024; Yao et al., 2025; Biran et al., 2024; Wang et al., 2025).

These limitations suggest that multi-hop editing failures are not solely caused by unsuccessful fact insertion. A model may store or access an updated association, yet still fail to use it reliably when composing downstream answers. Recent work on test-time computation provides a useful perspective: generation and verification need not be equally difficult, and models may fail to produce a correct solution while still being able to recognize whether a candidate is valid (Setlur et al., 2026; Snell et al., 2025). In multi-hop knowledge editing, this asymmetry appears as a generate–verification gap: an edited model may generate an answer from the pre-edit world, while a verifier can still detect its incompatibility with the updated facts.

One natural response is to expose the edit facts directly in the input, using direct in-context learn-

ing or chain-of-thought prompting (Wei et al., 2022; Wang et al., 2025; Zhao et al., 2024). However, these approaches still rely on a single generation process to identify relevant updates, combine them with other knowledge, and avoid reverting to pre-edit associations. Prior work on self-correction further shows that simply prompting a model to revise its own answer is often unreliable without a strong verification signal, especially for smaller or mid-sized models (Kumar et al., 2025; Zhang et al., 2024; Stechly et al., 2025). This motivates a decoding procedure that separates candidate generation from explicit correctness checking.

We propose **V**alidator-**G**uided **D**ecoding (**VGD**), an editor-agnostic test-time framework for multi-hop knowledge editing. VGD first generates a candidate answer, verifies it against the edited facts with a trained validator, and selectively revises low-confidence answers. In this way, VGD follows the broader direction of verifier-guided test-time control (Chang et al., 2025; Lee et al., 2025; Nguyen and Le, 2026), but targets the specific challenge of using edited knowledge in multi-hop reasoning. This distinction calls for an evaluation that separates verification-guided correction from ordinary fact-conditioned prompting. Since VGD uses edit facts during verification and revision, we compare it with direct ICL and CoT-style baselines that expose the same facts during generation, and further evaluate it as a plug-in module for parameter-updating editors. Our contributions are threefold:

- We characterize multi-hop knowledge editing through a generate–verification gap, where updated answers may be hard to generate directly but easier to verify against edit facts.
- We propose VGD, an editor-agnostic test-time decoding framework that uses a trained validator to accept reliable answers and revise inconsistent ones under updated factual knowledge.
- We show that VGD outperforms direct ICL and CoT-style prompting baselines, improves multiple parameter-updating editors as a plug-in module, and reduces the need for revision when applied to edited models.

2 Related Work

Multi-hop and reasoning-based knowledge editing. Multi-hop knowledge editing evaluates whether edited facts can be propagated through downstream reasoning chains rather than merely recalled in isolation (Zhong et al., 2023; Cohen et al., 2024; Qin et al., 2024). Recent work addresses this challenge by analyzing reasoning failures, modifying reasoning-relevant components, or improving edit propagation beyond direct factual recall (Hoelscher-Obermaier et al., 2023; Yao et al., 2025; Biran et al., 2024). Another line of work injects edit facts through prompts, demonstrations, or reasoning traces instead of relying solely on parameter updates (Zheng et al., 2023; Qiao et al., 2024a; Zhong et al., 2023; Wei et al., 2022; Wang et al., 2025). VGD is related to these parameter-preserving and reasoning-based approaches, but differs by using edit facts to verify and selectively revise candidate answers rather than treating them only as context for a single generation pass.

Verification and test-time revision. Recent work on test-time computation shows that additional inference-time search, verification, or refinement can improve model reasoning (Snell et al., 2025; Setlur et al., 2026; Ji et al., 2025). This motivates the generate–verification gap: a model may fail to generate a correct answer directly while still recognizing valid candidates. Verifier-guided inference and refinement have been studied in broader reasoning settings (Chang et al., 2025; Lee et al., 2025), while self-correction studies show that revision often requires strong feedback or trained verifiers (Kumar et al., 2025; Zhang et al., 2024; Stechly et al., 2025). VGD adapts these ideas to multi-hop knowledge editing by using a trained validator to guide answer acceptance and revision under updated facts.

3 Editing–Verification Gap

We study whether multi-hop editing failures arise only from missing updated knowledge, or also from a gap between generating and verifying updated answers. We formulate this phenomenon as an *editing–verification gap* (EVG), where an edited model fails to generate the correct answer but can still verify whether a proposed answer is compatible with the edited facts.

3.1 Setup of the EVG Diagnostic

Let $\mathcal{X}$ denote a set of multi-hop editing instances. Each instance $x \in \mathcal{X}$ contains a set of edit requests, a multi-hop question, and its updated answer. Each edit request changes the object of a factual relation from an old value to a new value:

$$(s_j, r_j, o_j^{\text{old}} \rightarrow o_j^*). \quad (1)$$

Method	Single-hop EVG			Multi-hop EVG		
	H. Acc.	V. Acc.	Δ_{hop}	M. Acc.	AllPass	Δ_{mh}
ROME	86.80	89.22	+2.42	18.10	48.17	+30.07
MEMIT	70.90	78.99	+8.09	9.23	28.93	+19.70
AlphaEdit	68.53	77.93	+9.40	9.93	28.00	+18.07
CaKE	89.70	90.39	+0.69	54.40	56.13	+1.73

Table 1: Editing–verification gap on MQuAKE-CF-3k with LLaMA-3-8B-Instruct under the self-verification setting. H. Acc. and M. Acc. are standard generation-based editing metrics. V. Acc. and AllPass are computed by prompting the same edited model to verify proposed answers under the edited facts.

We verbalize the edit requests in instance x as a set of edited facts

$$\mathcal{E}_x = \{f_j\}_{j=1}^{m_x}, \tag{2}$$

where m_x is the number of edits in x, and each f_j states one updated factual association. Given a question q, the model is expected to answer according to the edited world defined by $\mathcal{E}_x$.

For the multi-hop question q_x^{mh}, the dataset provides the updated answer y_x^{mh} and a chain of updated single-hop reasoning steps:

$$\mathcal{C}_x^* = \{(q_{x,i}^*, y_{x,i}^*)\}_{i=1}^{L_x}, \tag{3}$$

where $L_x = |\mathcal{C}_x^*|$ is the number of single-hop steps in instance x. Each pair $(q_{x,i}^*, y_{x,i}^*)$ corresponds to one intermediate reasoning step under the edited world. We also consider incompatible candidates $\bar{y}_{x,i}$, which are no longer correct answers to the same question after editing.

The EVG diagnostic compares two abilities of the same edited model $M_\theta^{\mathcal{E}}$: direct answer generation and edit-conditioned answer verification. In the verification setting, given edited facts $\mathcal{E}_x$, a question q, and a proposed answer a, the model predicts

$$v_M(\mathcal{E}_x, q, a) \in \{\text{YES}, \text{NO}\}. \tag{4}$$

The gold label is denoted by $\ell \in \{\text{YES}, \text{NO}\}$ and is defined as

$$\ell^*(\mathcal{E}_x, q, a) = \begin{cases} \text{YES}, & a \in \mathcal{A}(y_{\mathcal{E}_x}(q)), \\ \text{NO}, & \text{otherwise}, \end{cases} \tag{5}$$

where $y_{\mathcal{E}_x}(q)$ is the correct answer to q under the edited world, and $\mathcal{A}(y)$ contains the canonical answer and its aliases. The prompt used for edit-conditioned answer verification is shown in Appendix A.

For each instance x, we construct support and conflict verification examples:

$$\mathcal{D}_{\text{sup}}(x) = \{(\mathcal{E}_x, q_{x,i}^*, y_{x,i}^*, \text{YES})\}_{i=1}^{L_x}, \tag{6}$$
$$\mathcal{D}_{\text{conf}}(x) = \{(\mathcal{E}_x, q_{x,i}^*, \bar{y}_{x,i}, \text{NO}) \mid \bar{y}_{x,i} \notin \mathcal{A}(y_{x,i}^*)\}.$$

The EVG verification set for instance x is

$$\mathcal{D}_{\text{evg}}(x) = \mathcal{D}_{\text{sup}}(x) \cup \mathcal{D}_{\text{conf}}(x), \tag{7}$$

and the full verification set is

$$\mathcal{D}_{\text{evg}} = \bigcup_{x \in \mathcal{X}} \mathcal{D}_{\text{evg}}(x). \tag{8}$$

We formulate EVG as answer verification rather than generic fact contradiction, because an old-world statement may be unrelated to a given edit unless it is evaluated as a proposed answer to the same question.

3.2 Measuring the Editing–Verification Gap

EVG measures whether an edited model can verify updated answers more reliably than it can generate them. We use the standard hop-wise accuracy (H. Acc.) and multi-hop accuracy (M. Acc.) from the benchmark as generation-side metrics. H. Acc. evaluates direct generation on the updated single-hop questions in $\mathcal{C}_x^*$, while M. Acc. evaluates direct generation on the final multi-hop question q_x^{mh}.

On the verification side, we instantiate v_M using the same edited model being evaluated. That is, the edited model is prompted with the edited facts, a question, and a proposed answer, and must output YES or NO. For brevity, let $z = (\mathcal{E}_x, q, a, \ell)$ denote a verification example, and define

$$c(z) = \mathbb{I}\left[v_M(\mathcal{E}_x, q, a) = \ell\right]. \tag{9}$$

Then the overall verification accuracy is

$$\text{V.Acc.} = \frac{1}{|\mathcal{D}_{\text{evg}}|} \sum_{z \in \mathcal{D}_{\text{evg}}} c(z). \tag{10}$$

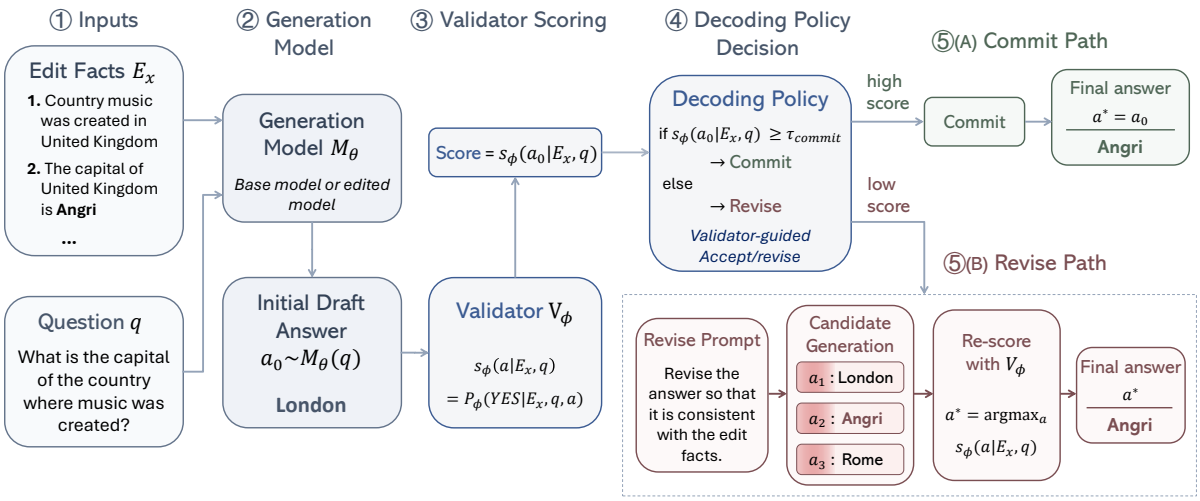

Figure 1: Overview of Validator-Guided Decoding (VGD). Given edited facts and a question, a base or edited model first generates an initial candidate answer. A trained validator then scores whether the candidate correctly answers the question under the edited facts. High-confidence candidates are directly committed, while low-confidence candidates trigger revision and candidate exploration. The final answer is selected according to the validator signal.

Since $\mathcal{D}_{\text{evg}}$ is constructed from the single-hop reasoning steps, V. Acc. serves as the verification-side counterpart to H. Acc.

To obtain a case-level approximation to multi-hop verification, we compute an all-pass score:

$$\text{AllPass} = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \prod_{z \in \mathcal{D}_{\text{evg}}(x)} c(z). \tag{11}$$

AllPass requires all verification decisions associated with instance x to be correct. Although it is not identical to answering the multi-hop question, it approximates the verification-side counterpart to M. Acc.: if a model can correctly verify the relevant single-hop answers and reject incompatible alternatives across the reasoning chain, then it has substantially more evidence for solving the corresponding multi-hop query than direct generation accuracy alone suggests. We therefore quantify EVG at two levels:

$$\begin{aligned} \Delta_{\text{hop}} &= \text{V.Acc.} - \text{H.Acc.}, \\ \Delta_{\text{mh}} &= \text{AllPass} - \text{M.Acc.}. \end{aligned} \tag{12}$$

A positive gap indicates that the edited model is better at verifying updated-world compatibility than directly generating the updated answer.

Table 1 shows that edited models exhibit a clear editing–verification gap. At the single-hop level, V. Acc. is higher than H. Acc., suggesting that models can often recognize whether proposed intermediate answers are compatible with the edited facts even when they fail to generate them directly. At the multi-hop level, AllPass is a stricter case-wise metric, yet it remains consistently higher than M. Acc. These results indicate that edited models may retain usable verification signals that are not reliably converted into correct multi-hop generations.

3.3 Implications

The EVG diagnostic suggests that verification should be used not merely as an evaluation tool, but also as a test-time control signal. Since edited models can often judge candidate answers more reliably than they can generate them, a decoding procedure can first obtain candidate answers and then use verification to decide whether to accept the current answer or explore revised alternatives.

4 Validator-Guided Decoding

The EVG analysis suggests that edited models may fail to directly generate updated answers even when they retain useful signals for verification. We therefore propose **Validator-Guided Decoding** (**VGD**), a test-time framework that separates answer generation from edit-conditioned answer verification. Instead of relying on a single generation pass, VGD trains a validator on the same backbone model to provide a more stable verification signal, which is then used to decide whether a candidate answer should be accepted or revised.

4.1 Overview

Figure 1 illustrates the VGD framework. Given edited facts $\mathcal{E}_x$ and a question q, a generation model

M_θ first produces an initial answer $a_0 \sim M_\theta(q)$. A trained validator V_ϕ then scores the answer by

$$s_\phi(a_0 \mid \mathcal{E}_x, q) = P_\phi(\text{YES} \mid \mathcal{E}_x, q, a_0). \quad (13)$$

If the score is high, VGD commits to the current answer; otherwise, it revises the answer, evaluates the revised candidates, and selects the one best supported by the validator. The generation model can be either an unedited base model or an edited model, making VGD applicable as a standalone edit-memory-assisted decoder or as a plug-in module for existing editors. This separation between generation and verification allows VGD to retain reliable initial outputs while correcting uncertain or inconsistent ones. We next describe validator training and the decoding policy.

4.2 Validator Training

The validator V_ϕ is trained to judge whether a proposed answer is correct under the edited facts. Following the EVG observation, this turns the model's verification ability into a stable test-time signal for answer acceptance and revision. Each training example is represented as

$$z = (\mathcal{E}_x, q, a, \ell), \quad (14)$$

where $\mathcal{E}_x$ is the edited facts for instance x, q is a question, a is a proposed answer, and $\ell \in \{\text{YES}, \text{NO}\}$ is the verification label. The validator learns to output YES when a correctly answers q under $\mathcal{E}_x$, and NO otherwise.

To make this signal applicable throughout decoding, we construct a unified validator dataset from both single-hop and multi-hop questions. For single-hop verification, positive examples use updated answers from the edited reasoning chain, while negative examples use incompatible answers:

$$z_{\text{sup}} = (\mathcal{E}_x, q^*_{x,i}, y^*_{x,i}, \text{YES}), \quad (15)$$

$$z_{\text{conf}} = (\mathcal{E}_x, q^*_{x,i}, \bar{y}_{x,i}, \text{NO}). \quad (16)$$

For multi-hop verification, we use the multi-hop question q^{mh}_x and construct analogous examples:

$$z^{\text{mh}}_{\text{sup}} = (\mathcal{E}_x, q^{\text{mh}}_x, y^{\text{mh}}_x, \text{YES}), \quad (17)$$

$$z^{\text{mh}}_{\text{conf}} = (\mathcal{E}_x, q^{\text{mh}}_x, \bar{y}^{\text{mh}}_x, \text{NO}). \quad (18)$$

The resulting training set is

$$\mathcal{D}_{\text{val}} = \mathcal{D}^{\text{sh}}_{\text{val}} \cup \mathcal{D}^{\text{mh}}_{\text{val}}, \quad (19)$$

where $\mathcal{D}^{\text{sh}}_{\text{val}}$ and $\mathcal{D}^{\text{mh}}_{\text{val}}$ denote the single-hop and multi-hop verification subsets. A joint validator allows VGD to use the same verification interface for both intermediate single-hop queries and final multi-hop queries, without introducing question-type-specific controllers.

Backbone	Single-hop	Multi-hop	Overall
LLaMA-3-8B-Ins.	99.85	99.79	99.83
Qwen2.5-7B-Ins.	99.72	99.54	99.64

Table 2: Held-out verification accuracy of the trained validator under case-level evaluation.

Given the verification prompt $p(\mathcal{E}_x, q, a)$, the validator is optimized by minimizing the label prediction loss:

$$\mathcal{L}_{\text{val}} = -\sum_{z \in \mathcal{D}_{\text{val}}} \log P_\phi(\ell \mid p(\mathcal{E}_x, q, a)). \quad (20)$$

At inference time, we use the probability assigned to YES as the scalar validation score:

$$s_\phi(a \mid \mathcal{E}_x, q) = P_\phi(\text{YES} \mid p(\mathcal{E}_x, q, a)). \quad (21)$$

This score is then used by the decoding policy to determine whether the current candidate should be accepted or revised.

To avoid leakage between validator training and downstream evaluation, we use case-level k-fold training. For each fold, the validator is trained on the training cases and used only on the held-out cases assigned to that fold. Thus, no test case is used to train the validator that controls decoding for that case. We report the held-out accuracy of the trained joint validator in Table 2. Its strong performance on verification examples shows that the validator provides a reliable signal for the decoding policy introduced next.

4.3 Validator-Guided Decoding Policy

Given a question q and edited facts $\mathcal{E}_x$, VGD uses the validator score to control whether the current answer should be accepted or revised. Let M_θ denote the generation model, which can be either a base model or an edited model. We first obtain an initial candidate answer $a_0 \sim M_\theta(q)$. The validator then assigns a correctness score

$$s_0 = s_\phi(a_0 \mid \mathcal{E}_x, q) = P_\phi(\text{YES} \mid p(\mathcal{E}_x, q, a_0)). \quad (22)$$

If s_0 is sufficiently high, VGD directly commits to a_0. This conservative acceptance step is important because edited models often already produce correct answers, and unnecessary revision may overwrite reliable generations. Formally, VGD uses a

commit threshold τ_{com}:

$$\text{Commit}(a_0) \quad \text{if} \quad s_\phi(a_0 \mid \mathcal{E}_x, q) \geq \tau_{\text{com}}. \tag{23}$$

If the candidate is not accepted, VGD enters a revision stage. At revision step t, given the current draft a_{t-1}, we construct a revision prompt

$$r_t = r(\mathcal{E}_x, q, a_{t-1}), \tag{24}$$

which provides the edited facts, the question, and the current draft answer. The full template is shown in Appendix A. To encourage exploration during revision, revised candidates are sampled with a higher-temperature decoding configuration:

$$\mathcal{A}_t = \{a_t^{(1)}, \ldots, a_t^{(K)}\}, a_t^{(k)} \sim M_\theta(r_t; T_{\text{rev}}), \tag{25}$$

where T_{rev} denotes the revision temperature. This turns revision into a controlled local exploration process: the generator proposes diverse alternatives, while the validator determines which candidates are compatible with the edited facts.

Each revised candidate is scored by the validator:

$$s_t^{(k)} = s_\phi(a_t^{(k)} \mid \mathcal{E}_x, q). \tag{26}$$

VGD then selects the candidate with the strongest validator support:

$$a_t^* = \arg\max_{a_t^{(k)} \in \mathcal{A}_t} s_\phi(a_t^{(k)} \mid \mathcal{E}_x, q), s_t^* = \max_k s_t^{(k)}. \tag{27}$$

If $s_t^* \geq \tau_{\text{com}}$, VGD immediately commits to a_t^*. This early-stop strategy prevents unnecessary revisions once a sufficiently verified answer is found, reducing the risk of overwriting a correct candidate through further exploration. Otherwise, the best candidate becomes the next draft, $a_t = a_t^*$, and revision continues until either early stopping is triggered or the maximum number of revisions T is reached. When no candidate reaches the commit threshold within T revision steps, VGD falls back to the candidate with the highest validator score among all explored candidates:

$$a_{\text{final}} = \arg\max_{a \in \mathcal{B}} s_\phi(a \mid \mathcal{E}_x, q), \tag{28}$$

where

$$\mathcal{B} = \{a_0\} \cup \bigcup_{t=1}^{T} \mathcal{A}_t \tag{29}$$

is the candidate pool accumulated during decoding. This fallback rule makes VGD robust to cases where no candidate exceeds the commit threshold, while still returning the answer most supported by the validator. The overall policy can be summarized as

$$a_{\text{final}} = \begin{cases} a_0, & s_\phi(a_0 \mid \mathcal{E}_x, q) \geq \tau_{\text{com}}, \\ a_t^*, & \exists t \leq T,\ s_t^* \geq \tau_{\text{com}}, \\ \arg\max_{a \in \mathcal{B}} s_\phi(a \mid \mathcal{E}_x, q), & \text{otherwise.} \end{cases} \tag{30}$$

For clarity, we provide the complete pseudo-code of VGD in Appendix B.

VGD can be viewed as conservative verifier-guided exploration over candidate answers. Unlike direct ICL or CoT prompting, which relies on a single fact-conditioned generation trajectory, VGD separates proposal, verification, and revision. The generator explores possible answers under controlled stochastic decoding, while the validator provides a correctness signal for early commitment, candidate selection, and fallback. Since the policy only requires candidate answers and validator scores, it is independent of the underlying editing algorithm and can be applied to both base and edited models.

5 Experiments

We evaluate whether VGD improves multi-hop use of edited knowledge, and whether its gains go beyond direct fact-conditioned prompting.

Datasets and Models. We evaluate on three MQuAKE variants (Zhong et al., 2023): **MQuAKE-CF-3k**, **MQuAKE-CF-3k-v2**, and the temporal subset **MQuAKE-T**. We use **LLaMA3-8B-Instruct** (Touvron et al., 2023) and **Qwen2.5-7B-Instruct** (Yang et al., 2024). Validator training and VGD evaluation follow case-level k-fold splits to prevent test-case leakage. Hyperparameters are reported in Appendix C.

Baselines. We compare with representative parameter-updating editors, including **LoRA** fine-tune, **ROME** (Meng et al., 2022), **MEMIT** (Meng et al., 2023), **AlphaEdit** (Fang et al., 2025), and **CaKE** (Yao et al., 2025). Since VGD accesses edited facts during verification and revision, we also include **Direct ICL** and **CoT ICL** baselines that expose the same facts during generation (Wei et al., 2022; Wang et al., 2025). VGD is evaluated both with the unedited base model and as a plug-in decoding module on top of edited models.

Method	MQuAKE-CF-3k		MQuAKE-CF-3k-v2		MQuAKE-T	
	H-Acc.	M-Acc.	H-Acc.	M-Acc.	H-Acc.	M-Acc.
LLaMA3-8B-Instruct						
Base-ICL	85.68	42.43	85.82	42.08	88.31	52.17
Base-CoT	69.50	45.33	69.83	46.15	73.60	47.78
LoRA	66.00	27.60	64.70	24.60	92.30	66.00
ROME	86.80	18.10	86.40	15.50	90.70	4.60
MEMIT	70.90	9.23	74.00	10.00	88.40	2.10
AlphaEdit	68.53	9.93	63.70	8.50	73.40	1.00
CaKE	89.70	54.40	90.10	57.10	88.20	78.30
Base+VGD	**92.60**	**71.70**	88.35	64.79	**93.13**	64.70
CaKE+VGD	90.55	68.20	**90.63**	**69.47**	88.75	**83.02**

Table 3: Main results on three MQuAKE variants with LLaMA3-8B-Instruct. H-Acc. measures hop-wise accuracy on intermediate questions, and M-Acc. measures accuracy on final multi-hop questions.

Evaluation Metrics. We report **Multi-hop Accuracy (M-Acc.)**, which measures correctness on final multi-hop questions, and **Hop-wise Accuracy (H-Acc.)**, which measures correctness on intermediate single-hop questions. For VGD, we additionally report revision diagnostics, including revision rate and average number of revisions, to distinguish final accuracy from test-time correction cost.

6 Results

We present the experimental results from four perspectives: overall performance, plug-in gains on edited models, robustness across reasoning depths, and revision cost.

6.1 Main Results

Table 3 reports the main results on LLaMA3-8B-Instruct. We compare VGD with parameter-updating editors and fact-accessible Direct ICL and CoT ICL baselines. We report VGD with the unedited generator to isolate verifier-guided revision, and CaKE+VGD as an editor-backed setting built on the strongest baseline editor. Results on Qwen2.5-7B-Instruct are provided in Appendix D.

Across datasets, VGD substantially improves both hop-wise and multi-hop accuracy over Direct ICL and CoT ICL. When applied on top of CaKE, VGD further improves the strongest edited baseline and achieves the best overall performance in most settings. This shows that separating answer proposal, verification, and revision is more effective than relying on a single generation pass over the edited facts. The result supports the central design of VGD: edited facts are used not only as generation context, but as a verification signal that guides selective test-time revision. Appendix G provides a concrete example of this process.

6.2 VGD as a Plug-in Module for Editors

We further evaluate whether VGD can serve as a plug-in decoding module for different parameter-updating editors. The full results on both LLaMA3-8B-Instruct and Qwen2.5-7B-Instruct are provided in Appendix E. Across editors, datasets, and backbones, applying VGD consistently improves hop-wise and multi-hop accuracy, confirming that its benefit is not tied to a specific editing algorithm or model family. The gains are most pronounced on final multi-hop accuracy, suggesting that edited models often retain useful signals for the updated facts but fail to compose them reliably during direct generation. VGD complements parametric editing by correcting these residual generation failures at test time. This also motivates the following analyses, where we examine whether the gains persist across reasoning depths and how much revision cost is required.

6.3 Performance by Reasoning Depth

We next examine whether the gains of VGD persist as the reasoning chain becomes deeper. MQuAKE-CF-3k separates cases into 2-hop, 3-hop, and 4-hop groups, allowing us to analyze depth-wise performance under the same evaluation protocol. Table 5 reports M-Acc. across reasoning depths for representative methods on LLaMA3-8B-Instruct. We provide a complete overview covering all editors,

Method	Rev. Rate	Avg. Rev.	Gen./Edit Time	VGD Time	Time Overhead	Rel. M Gain
Base	75.06%	0.77	7.50 s	11.11 s	+148.13%	+71.70%
LoRA	47.44%	0.50	13.86 s	5.80 s	+41.85%	+38.67%
ROME	34.00%	0.35	25.55 s	4.46 s	+17.45%	+48.83%
MEMIT	36.58%	0.38	64.75 s	5.47 s	+8.45%	+59.34%
AlphaEdit	36.48%	0.38	28.63 s	5.31 s	+18.55%	+56.57%
CaKE	7.4%	0.08	64.64 s	2.20 s	+3.40%	+13.80%

Table 4: Revision frequency and runtime cost of VGD on MQuAKE-CF-3k with LLaMA3-8B-Instruct. Rev. Rate denotes the percentage of queries that trigger at least one revision, and Avg. Rev. denotes the average number of revision attempts per query. Gen./Edit Time is the average cost before applying VGD. VGD Time is the additional cost of validation and revision. Time Overhead is computed as VGD Time divided by Gen./Edit Time. Rel. M Gain denotes the relative M-Acc. improvement over the corresponding editor without VGD.

Method	2-hop	3-hop	4-hop
Base-ICL	68.50	35.00	23.80
Base-CoT	53.75	37.50	44.75
Base+VGD	80.80	59.60	74.70
CaKE	64.14	49.00	50.00
CaKE+VGD	78.90	58.30	67.40

Table 5: Depth-wise M-Acc. on MQuAKE-CF-3k with LLaMA3-8B-Instruct.

VGD variants, H-Acc., and M-Acc. in Appendix F.

Across reasoning depths, VGD maintains clear gains over Direct ICL and CoT ICL, indicating that validator-guided revision is not limited to shallow fact substitution. Instead, it remains effective when the model must combine multiple edited or edit-dependent facts to reach the final answer. This supports our view that VGD improves the test-time use of edited knowledge, rather than only increasing access to the edited facts. The same pattern appears for edited generators. CaKE+VGD improves over CaKE at every reasoning depth, showing that VGD can further correct residual multi-hop failures even when the underlying editor already improves knowledge propagation. Overall, the depth-wise results indicate that VGD provides robust gains under increasingly complex reasoning structures.

6.4 Revision Frequency and Runtime Cost

We finally analyze the computational cost of VGD. Table 4 reports revision frequency, average revision attempts, and runtime overhead on MQuAKE-CF-3k with Qwen2.5-7B-Instruct. For each method, Gen./Edit Time measures the cost of the underlying generator before VGD: initial decoding time for Base, and editing plus initial decoding time for edited generators. VGD Time measures the additional cost introduced by validation and revision.

The results show that VGD provides large relative gains in M-Acc. with modest runtime overhead on editors. This efficiency comes from the selective nature of the decoding policy: candidates with high validator confidence are committed directly, while additional revision is invoked only when the current answer is unreliable. Thus, VGD does not uniformly increase generation cost, but allocates extra computation to cases that are more likely to benefit from correction. The revision statistics also clarify the role of parameter editing. Unedited base model relies more heavily on test-time revision, while edited generators generally require fewer revision attempts to achieve strong final performance. This suggests that parametric editing and VGD improve different parts of the pipeline: editing provides a better initial generator state, whereas VGD improves the reliability of final answers through validation and selective correction.

7 Conclusion

We examined multi-hop knowledge editing through the generate–verification gap, where edited models may fail to directly generate updated answers despite retaining useful verification signals. We proposed Validator-Guided Decoding (VGD), a test-time framework that separates answer proposal, edit-conditioned verification, and selective revision. VGD uses a trained validator to decide whether to accept an answer or explore revised candidates, without modifying the editor or further updating the edited model. Experiments on multiple MQuAKE variants show that VGD improves multi-hop use of edited knowledge across reasoning depths and consistently outperforms other baseline editors.

Limitations

This work focuses on multi-hop factual knowledge editing and is evaluated mainly on MQuAKE-style benchmarks. Although these benchmarks provide controlled edit facts and reasoning chains, they may not fully reflect open-domain, long-context, or interactive knowledge-update scenarios. Extending VGD to broader editing settings remains an important direction.

VGD also uses edited facts during verification and revision. We therefore compare against Direct ICL and CoT ICL baselines that expose the same facts, but the boundary between verification-guided revision and fact-conditioned prompting remains an important consideration. Future work may explore settings where edited facts are compressed, retrieved, or implicitly injected into the validator, reducing the need to present all edit facts explicitly during test-time correction.

References

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. 2024. Hopping too late: Exploring the limitations of large language models on multi-hop queries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14113–14130, Miami, Florida, USA. Association for Computational Linguistics.

Kaiyan Chang, Yonghao Shi, Chenglong Wang, Hang Zhou, Chi Hu, Xiaoqian Liu, Yingfeng Luo, Yuan Ge, Tong Xiao, and JingBo Zhu. 2025. Step-level verifier-guided hybrid test-time scaling for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18462–18477, Suzhou, China. Association for Computational Linguistics.

Roi Cohen, Eden Biran, Ori Yoran, Amir Globerson, and Mor Geva. 2024. Evaluating the ripple effects of knowledge editing in language models. *Transactions of the Association for Computational Linguistics*, 12:283–298.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2025. Alphaedit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, volume 2025, pages 16366–16396.

Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. 2024. Model editing harms general abilities of large language models: Regularization to the rescue. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16801–16819, Miami, Florida, USA. Association for Computational Linguistics.

Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. 2024. Model editing at scale leads to gradual and catastrophic forgetting. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15202–15232, Bangkok, Thailand. Association for Computational Linguistics.

Jason Hoelscher-Obermaier, Julia Persson, Esben Kran, Ioannis Konstas, and Fazl Barez. 2023. Detecting edit failures in large language models: An improved specificity benchmark. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11548–11559, Toronto, Canada. Association for Computational Linguistics.

Yixin Ji, Juntao Li, Yang Xiang, Hai Ye, Kaixin Wu, Kai Yao, Jia Xu, Linjian Mo, and Min Zhang. 2025. A survey of test-time compute: From intuitive inference to deliberate reasoning. *arXiv preprint arXiv:2501.02497*.

Houcheng Jiang, Junfeng Fang, Ningyu Zhang, Mingyang Wan, Guojun Ma, Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2025. Anyedit: Edit any knowledge encoded in language models. In *Forty-second International Conference on Machine Learning*.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2025. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Hyunseok Lee, Seunghyuk Oh, Jaehyung Kim, Jinwoo Shin, and Jihoon Tack. 2025. Revise: Learning to refine at test-time via intrinsic self-verification. *arXiv preprint arXiv:2502.14565*.

Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. 2024. Pmet: Precise model editing

in a transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18564–18572.

Jun-Yu Ma, Hong Wang, Hao-Xiang Xu, Zhen-Hua Ling, and Jia-Chen Gu. 2025. Perturbation-restrained sequential model editing. In *International Conference on Learning Representations*, volume 2025, pages 17302–17329.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 35.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. 2023. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2022. Fast model editing at scale. In *International Conference on Learning Representations*.

Tuc Nguyen and Thai Le. 2026. Atlas: Adaptive test-time latent steering with external verifiers for enhancing llms reasoning. *arXiv preprint arXiv:2601.03093*.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Shanbao Qiao, Xuebing Liu, and Seung-Hoon Na. 2024a. COMEM: In-context retrieval-augmented mass-editing memory in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2333–2347, Mexico City, Mexico. Association for Computational Linguistics.

Shanbao Qiao, Xuebing Liu, and Seung-Hoon Na. 2024b. DistillMIKE: Editing distillation of massive in-context knowledge editing in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7639–7654, Bangkok, Thailand. Association for Computational Linguistics.

Shanbao Qiao, Xuebing Liu, and Seung-Hoon Na. 2025. Wasserstein distance constraint and parameter sparsification for batched and iterative knowledge editing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25019–25028.

Jiaxin Qin, Zixuan Zhang, Chi Han, Pengfei Yu, Manling Li, and Heng Ji. 2024. Why does new knowledge create messy ripple effects in LLMs? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12602–12609, Miami, Florida, USA. Association for Computational Linguistics.

Amrith Setlur, Matthew Y. R. Yang, Charlie Victor Snell, Jeremiah Greer, Ian Wu, Virginia Smith, Max Simchowitz, and Aviral Kumar. 2026. e3: Learning to explore enables extrapolation of test-time compute for LLMs. In *The Fourteenth International Conference on Learning Representations*.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. 2025. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, volume 2025, pages 98190–98243.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Changyue Wang, Weihang Su, Qingyao Ai, Yichen Tang, and Yiqun Liu. 2025. Knowledge editing through chain-of-thought. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 10673–10693, Suzhou, China. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Yunzhi Yao, Jizhan Fang, Jia-Chen Gu, Ningyu Zhang, Shumin Deng, Huajun Chen, and Nanyun Peng. 2025. CaKE: Circuit-aware editing enables generalizable knowledge learners. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 11366–11382, Suzhou, China. Association for Computational Linguistics.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10222–10240, Singapore. Association for Computational Linguistics.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024. Small language models need strong verifiers to self-correct reasoning. In

Findings of the Association for Computational Linguistics: ACL 2024, pages 15637–15653, Bangkok, Thailand. Association for Computational Linguistics.

Zihao Zhao, Yuchen Yang, Yijiang Li, and Yinzhi Cao. 2024. RippleCOT: Amplifying ripple effect of knowledge editing in language models via chain-of-thought in-context learning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6337–6347, Miami, Florida, USA. Association for Computational Linguistics.

Ce Zheng, Lei Li, Qingxiu Dong, Yuxuan Fan, Zhiyong Wu, Jingjing Xu, and Baobao Chang. 2023. Can we edit factual knowledge by in-context learning? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4862–4876, Singapore. Association for Computational Linguistics.

Zexuan Zhong, Zhengxuan Wu, Christopher Manning, Christopher Potts, and Danqi Chen. 2023. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15686–15702, Singapore. Association for Computational Linguistics.

A Prompt Templates

We provide the prompt templates used by VGD for edit-conditioned answer verification and revision. The same validator template is used during validator training and test-time validation. The revision template is used when the validator rejects the current draft answer and VGD samples revised candidates.

Validator prompt. Given edited facts $\mathcal{E}_x$, a question q, and a proposed answer a, the validator predicts whether a correctly answers q under the edited facts.

Validator Prompt

```
You are a careful answer verifier.

Assume the following facts are true.

Facts:
{fact_lines}

Question:
{question}

Proposed answer:
{proposed_answer}

Does the proposed answer correctly answer the
question under the facts above?
Answer with only YES or NO.
```

Revision prompt. When the validator assigns a low confidence score to the current draft answer, VGD constructs the following prompt to sample revised candidates from the generator.

Revision Prompt

```
You are answering a factual question under
updated facts.

Updated facts:
{fact_lines}

Question:
{question}

Current draft answer:
{current_answer}

Revise the answer so that it is fully
consistent with the updated facts.
Output only the final concise answer, with no
explanation.

Final concise answer:
```

B Pseudo-code of Validator-Guided Decoding

Algorithm 1 provides the pseudo-code of the VGD procedure described in Section 4.3.

C Hyperparameter Settings

Table 6 summarizes the main hyperparameters used for validator training and VGD decoding.

We use the same decoding and validation hyperparameters across datasets. For validator training, we fine-tune the corresponding backbone model with LoRA adapters. The validator is trained under case-level k-fold splits, where each fold uses the training cases for supervision and the held-out cases for VGD evaluation. For VGD, we use a conservative commit threshold to preserve reliable initial answers and trigger revision only when the validator score is low. During revision, we sample multiple candidates with a higher temperature to encourage local exploration, then select or commit candidates according to the validator score. For direct ICL, the edited facts are prepended to the question. For CoT ICL, we use the same edited facts and additionally prompt the model to reason step by step before producing the final answer. All prompting baselines use the same answer extraction and answer-matching protocol as VGD.

D Additional Main Results on Qwen2.5-7B-Instruct

Table 7 reports the corresponding main results on Qwen2.5-7B-Instruct. Overall, the results show

Algorithm 1 Validator-Guided Decoding

Input: generation model M_θ, validator V_ϕ, edited facts $\mathcal{E}_x$, question q, commit threshold τ_{com}, candidates per revision K, maximum revisions T
Generate initial answer $a_0 \sim M_\theta(q)$ ▷ standard decoding
Compute $s_0 = s_\phi(a_0 \mid \mathcal{E}_x, q)$ ▷ validator confidence for the initial answer
Initialize candidate pool $\mathcal{B} \leftarrow \{a_0\}$ ▷ store all explored answers
if $s_0 \geq \tau_{\text{com}}$ **then**
  return a_0 ▷ early commit to preserve reliable generations
end if
Set current draft $a \leftarrow a_0$
for $t = 1$ to T **do**
  Construct revision prompt $r_t = r(\mathcal{E}_x, q, a)$ ▷ condition revision on current draft
  Sample revised candidates $\mathcal{A}_t = \{a_t^{(1)}, \ldots, a_t^{(K)}\}$ from $M_\theta(r_t)$
  Update candidate pool $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{A}_t$
  Select $a_t^* = \arg\max_{a' \in \mathcal{A}_t} s_\phi(a' \mid \mathcal{E}_x, q)$ ▷ validator-guided candidate selection
  Let $s_t^* = s_\phi(a_t^* \mid \mathcal{E}_x, q)$
  if $s_t^* \geq \tau_{\text{com}}$ **then**
    return a_t^* ▷ commit once a revised answer is sufficiently verified
  end if
  Set current draft $a \leftarrow a_t^*$ ▷ continue from the best current candidate
end for
return $\arg\max_{a' \in \mathcal{B}} s_\phi(a' \mid \mathcal{E}_x, q)$ ▷ fallback to the best verified candidate

a similar trend to the LLaMA3-8B-Instruct setting in the main text. Direct ICL and CoT ICL provide strong fact-accessible prompting baselines, but VGD achieves stronger multi-hop performance by separating answer proposal, verification, and revision. When applied on top of a strong edited generator, VGD further improves the use of edited knowledge, confirming that the benefit of validator-guided decoding is not specific to a single backbone.

E Plug-in VGD Results

We provide the full plug-in results of VGD on LLaMA3-8B-Instruct and Qwen2.5-7B-Instruct. For each parameter-updating editor, VGD is applied on top of the corresponding edited generator without changing the editing procedure. Tables 8 and 9 show that the plug-in effect of VGD is not limited to a single backbone. Across editors and datasets, VGD generally improves both hop-wise and multi-hop accuracy across editors and datasets, supporting its role as a model-agnostic and editor-agnostic decoding-time enhancement for edited models.

F Full Depth-wise Performance Overview

Figure 2 provides a complete visualization of depth-wise performance on MQuAKE-CF-3k with LLaMA3-8B-Instruct. Each subplot compares a generator with its corresponding VGD-augmented variant, and the six axes represent H-Acc. and M-Acc. under 2-hop, 3-hop, and 4-hop reasoning. This figure complements Table 5 by showing that the gains of VGD are not restricted to the representative Base and CaKE settings, but generally extend across different edited generators and evaluation depths.

G Case Study of Validator-Guided Revision

Figure 3 shows a revision trajectory produced by VGD. The initial draft gives an incorrect answer with a low validator score. VGD then constructs a revision prompt using the edited facts, samples a corrected candidate, and commits to it once the validator score reaches the commit threshold.

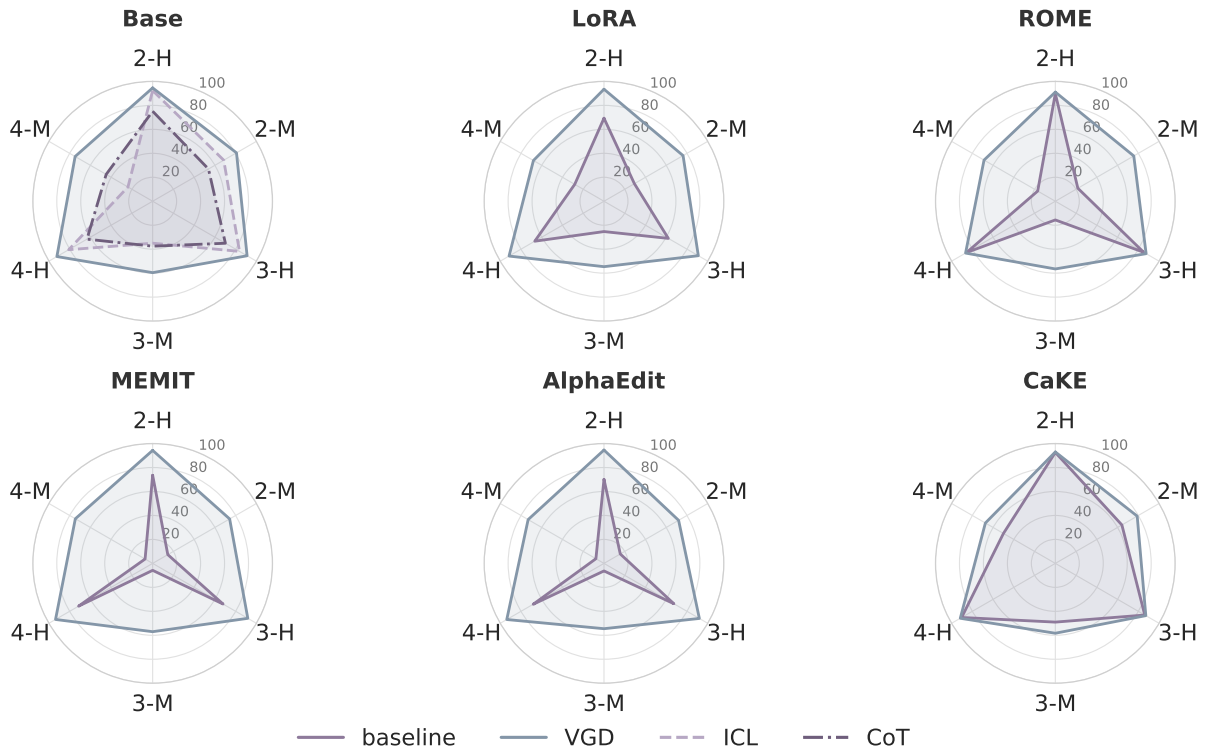

Figure 2: Full depth-wise performance overview on MQuAKE-CF-3k with LLaMA3-8B-Instruct. Each radar plot compares a generator with its VGD-augmented variant. The axes labeled k-H or k-M correspond to k-hop H-Acc. or M-Acc. for 2-hop, 3-hop, and 4-hop cases.

Hyperparameter	Value
Validator Training	
Number of folds	5
LoRA rank	8
LoRA alpha	32
Learning rate	2e-4
Warmup ration	0.03
Batch size	16
Training Epochs	3
Max sequence length	1024
VGD Decoding	
Commit threshold τ_{com}	0.85
Candidates per revision K	4
Maximum revisions T	2
Revision temperature T_{rev}	0.8
Top-p	0.95
Maximum new tokens	10

Table 6: Main hyperparameters for validator training and VGD decoding.

Method	MQuAKE-CF-3k		MQuAKE-CF-3k-v2		MQuAKE-T	
	H-Acc.	M-Acc.	H-Acc.	M-Acc.	H-Acc.	M-Acc.
Qwen2.5-7B-Instruct						
Base-ICL	89.78	58.73	89.69	60.69	88.44	74.29
Base-CoT	86.15	54.33	87.37	56.07	83.32	65.13
LoRA	35.10	24.90	36.50	25.90	25.00	28.60
ROME	81.70	11.30	73.40	8.80	87.70	17.70
MEMIT	84.00	15.80	83.40	9.60	88.90	18.50
AlphaEdit	79.30	17.40	75.10	10.50	82.20	17.20
CaKE	90.70	59.70	89.90	59.50	94.70	70.20
Base+VGD	89.41	**75.83**	88.70	58.70	85.73	61.11
CaKE+VGD	**91.20**	71.30	**90.00**	**63.30**	**95.05**	**84.14**

Table 7: Main results on three MQuAKE variants with Qwen2.5-7B-Instruct. H-Acc. measures hop-wise accuracy on intermediate questions, and M-Acc. measures accuracy on final multi-hop questions.

Method	MQuAKE-CF-3k		MQuAKE-CF-3k-v2		MQuAKE-T	
	H-Acc.	M-Acc.	H-Acc.	M-Acc.	H-Acc.	M-Acc.
LLaMA3-8B-Instruct						
LoRA	66.00	27.60	64.70	24.60	92.30	66.00
LoRA+VGD	91.94(+25.94)	66.27(+38.67)	90.36(+25.66)	61.00(+36.40)	94.14(+1.84)	82.07(+16.07)
ROME	86.80	18.10	86.40	15.50	90.70	4.60
ROME+VGD	88.36(+1.56)	66.93(+48.83)	87.14(+0.74)	63.17(+47.67)	93.50(+2.80)	63.07(+58.47)
MEMIT	70.90	9.23	74.00	10.00	88.40	2.10
MEMIT+VGD	93.21(+22.31)	68.57(+59.34)	92.13(+18.13)	65.63(+55.63)	93.04(+6.64)	65.67(+63.57)
AlphaEdit	68.53	9.93	63.70	8.50	73.40	1.00
AlphaEdit+VGD	93.37(+24.84)	66.50(+56.57)	92.06(+28.36)	62.63(+54.13)	92.84(+19.44)	65.67(+64.67)
CaKE	89.70	54.40	90.10	57.10	88.20	78.30
CaKE+VGD	90.55(+0.85)	68.20(+13.80)	90.63(+0.53)	69.47(+12.37)	88.75(+0.55)	83.02(+4.72)

Table 8: VGD as a plug-in decoding module for edited models on LLaMA3-8B-Instruct. For each editor, +VGD applies the same validator-guided decoding policy on top of the edited generator. Values in parentheses indicate absolute gains over the corresponding editor without VGD.

Method	MQuAKE-CF-3k		MQuAKE-CF-3k-v2		MQuAKE-T	
	H-Acc.	M-Acc.	H-Acc.	M-Acc.	H-Acc.	M-Acc.
Qwen2.5-7B-Instruct						
LoRA	35.10	24.90	36.50	25.90	25.00	28.60
LoRA+VGD	38.63(+3.53)	27.27(+2.37)	38.93(+2.43)	30.15(+4.25)	27.12(+2.12)	31.17(+2.57)
ROME	81.70	11.30	73.40	8.80	87.70	17.70
ROME+VGD	88.50(+6.80)	78.20(+66.90)	78.70(+5.30)	64.30(+55.50)	91.03(+3.33)	61.44(+43.74)
MEMIT	84.00	15.80	83.40	9.60	88.90	18.50
MEMIT+VGD	89.90(+5.90)	76.70(+60.90)	85.60(+2.20)	47.70(+38.10)	91.60(+2.70)	63.30(+44.80)
AlphaEdit	79.30	17.40	75.10	10.50	82.20	17.20
AlphaEdit+VGD	87.90(+8.60)	76.70(+59.30)	82.50(+7.40)	47.00(+36.50)	88.10(+5.90)	62.80(+45.90)
CaKE	90.70	59.70	89.90	59.50	94.70	70.20
CaKE+VGD	91.20(+0.50)	71.30(+11.60)	90.00(+0.10)	63.30(+3.80)	95.05(+0.35)	84.14(+13.94)

Table 9: VGD as a plug-in decoding module for edited models on Qwen2.5-7B-Instruct. For each editor, +VGD applies the same validator-guided decoding policy on top of the edited generator. Values in parentheses indicate absolute gains over the corresponding editor without VGD.

VGD Revision Log

```
[REVISE START]
[SCOPE] main_question
[VALIDATOR QUESTION] What is the birthplace of the chairperson of the Ontario New Democratic
Party?
[FACTS] ['The chairperson of Ontario New Democratic Party is Antonis Samaras.',
'Antonis Samaras was born in the city of Harvey.']
[VALIDATOR PROMPT]:
You are a careful answer verifier.
Assume the following facts are true.
Facts:
1. The chairperson of Ontario New Democratic Party is Antonis Samaras.
2. Antonis Samaras was born in the city of Harvey.
Question:
What is the birthplace of the chairperson of the Ontario New Democratic Party?
Proposed answer:
Windsor, as that is the city where Antonis
Under this assumption, does the proposed answer correctly answer the question?
Answer with only YES or NO.
[ANCHOR] raw='Windsor, as that is the city where Antonis' | p_yes=0.0006
```

```
[REVISE ROUND 0] current_answer='Windsor, as that is the city where Antonis'
[REVISE PROMPT]
You are answering a factual question under updated facts.
Updated facts:
1. The chairperson of Ontario New Democratic Party is Antonis Samaras.
2. Antonis Samaras was born in the city of Harvey.
Question:
What is the birthplace of the chairperson of the Ontario New Democratic Party?
Current draft answer:
Windsor, as that is the city where Antonis
Revise the answer so that it is fully consistent with the updated facts. Output only the final
concise answer, with no explanation.
Final concise answer:
[VALIDATOR PROMPT]:
You are a careful answer verifier.
Assume the following facts are true.
Facts:
1. The chairperson of Ontario New Democratic Party is Antonis Samaras.
2. Antonis Samaras was born in the city of Harvey.
Question:
What is the birthplace of the chairperson of the Ontario New Democratic Party?
Proposed answer:
Harvey
Under this assumption, does the proposed answer correctly answer the question?
Answer with only YES or NO.
[REVISE CANDIDATE 0] raw='Harvey' | p_yes=1.0000
[REVISE EARLY STOP] candidate 0 reached commit threshold with p_yes=1.0000
[REVISE ROUND BEST] chosen='Harvey' | view=raw | p_yes=1.0000
[REVISE COMMIT] found a strong revised candidate
Final output: Harvey
Gold aliases: ['Harvey, IL', 'Harvey, Illinois', 'Harvey']
```

Figure 3: An example of VGD revision trajectory. The validator rejects the initial draft answer with low confidence, VGD samples a revised candidate under the edited facts, and early-stops once the candidate reaches the commit threshold.