CASR: GENERATING COMPLEX SEQUENCES WITH AUTOREGRESSIVE SELF-BOOST REFINEMENT

Hongwei Han[1]* **Mengyu Zhou**[2]† **Shi Han**[2] **Xiu Li**[1]† **Dongmei Zhang**[2]
[1]Tsinghua Shenzhen International Graduate School, Tsinghua University
[2]Microsoft Research
hhw20@mails.tsinghua.edu.cn, li.xiu@sz.tsinghua.edu.cn
{mezho, shihan, dongmeiz}@microsoft.com

ABSTRACT

There are sequence generation tasks where the best order to generate the target sequence is not left-to-right. For example, an answer to the Sudoku game, a structured code like s-expression, and even a logical natural language answer where the analysis may be generated after the decision. We define the target sequences of those tasks as complex sequences. Obviously, a complex sequence should be constructed with multiple logical steps, and has dependencies among each part of itself (*e.g.* decisions depend on analyses). It's a great challenge for the classic left-to-right autoregressive generation system to generate complex sequences. Current approaches improve one-pass left-to-right generation on NLG tasks by generating different heuristic intermediate sequences in multiple stages. However, for complex sequences, the heuristic rules to break down them may hurt performance, and increase additional exposure bias. To tackle these challenges, we propose a PLM-friendly autoregressive self-boost refinement framework, CASR. When training, CASR inputs the predictions generated by the model itself at the previous refinement step (instead of those produced by heuristic rules). To find an optimal design, we also discuss model architecture, parameter efficiency and initialization strategy. By evaluating CASR on Sudoku, WebQSP, MTOP and KVRET through controlled experiments and empirical studies, we find that CASR produces high-quality outputs. CASR also improves Accuracy on Sudoku (70.93% $\rightarrow$ 97.28%) and achieves state-of-the-art performance on KVRET with Micro F1 score (67.88% $\rightarrow$ 70.00%).

1 INTRODUCTION

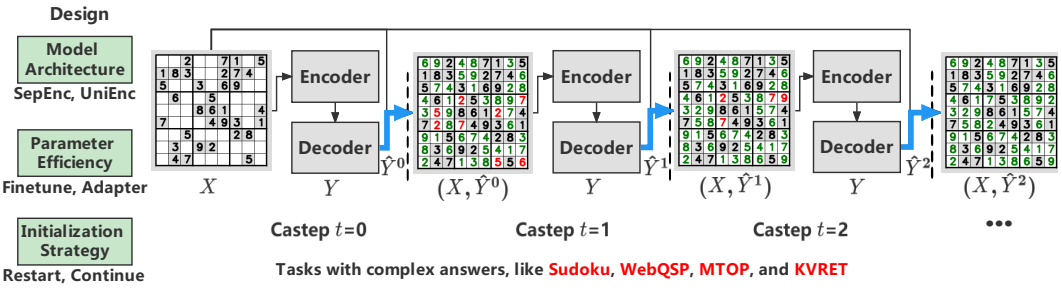


Figure 1: The Overview of CASR Framework. X, Y and $\hat{Y}$ denote the input, ground truth and prediction, respectively. The blue arrows show how we iteratively added back the previous-step prediction $\hat{Y}^{t-1}$ to the input for generating refined output $\hat{Y}^t$.

* The contributions by Hongwei Han have been conducted and completed during their internships at Microsoft Research Asia, Beijing, China.

† Corresponding authors.

Sequence generation models are widely used in tasks related to natural, domain-specific and programming languages – *E.g.*, question answering (Pandya & Bhatt, 2021), neural machine translation (Yang et al., 2020), speech recognition (Malik et al., 2021), automatic data analysis (Zhou et al., 2020), drug discovery (Kim et al., 2021), document summarization (Ma et al., 2020), code search and generation (Lee et al., 2021), *etc.*

To achieve better performance on these tasks, recent works often adopt autoregressive (AR) models (Wu et al., 2016), especially the ones with one-pass L2R (left-to-right) token-by-token generation / decoding order. Many SOTA-performance generative PLMs (pre-trained language models) are one-pass L2R models, such as GPT (Radford et al., 2018), T5 (Raffel et al., 2020), Bart (Lewis et al., 2020), *etc.* Different from non-autoregressive (NAR) models (Gu et al., 2017) which assume independence among tokens, L2R models assume the conditional probability in the form of $P(Y|X) = \prod_i P(y_i|X, y_{<i})$ which better captures the left-side dependencies that exist in most generation tasks. More variations of generation models are discussed in §2.1.

However, for many sequence generation tasks, beyond the left-side dependencies, there are right-side dependencies in the answer sequence, which together lead to multi-hop dependency chains, making left-to-right not the best order for generation. We call these tasks as complex tasks, and the answer sequences of these tasks as complex sequences. Complex tasks (see more details in §2.2, including Sudoku (PARK), WebQSP (Yih et al., 2016), MTOP (Li et al., 2021), and KVRET (Eric et al., 2017), *etc.*.) require better generation mechanism beyond one-pass L2R generation, since complex sequences are usually long, difficult, structured, or logical, which should be constructed with multiple logical steps.

6	9	2	4	8	7	1	3	5
1	8	3	5	9	2	7	4	6
5	7	4	3	1	6	9	2	8
4	6	1	7	5	3	8	9	2
3	2	9	8	6	1	5	7	4
7	5	8	2	4	9	3	6	1
9	1	5	6	7	4	2	8	3
8	3	6	9	2	5	4	1	7
2	4	7	1	3	8	6	5	9

(a) A 9x9 Sudoku Example. White cells denote blanks, and the green numbers in them denote the ground truth.

3d	1	4b	2
4	2c	3	1a
2g	4	1	3
1	3f	2e	4

(b) A 4x4 Sudoku Example. The game solving order of a human is from "a" to "g", rather than row-by-row.

Figure 2: Examples of Sudoku.

Human beings solve a complex problem with respect to its intrinsic order. For example, the order to write hierarchical answers (such as s-Expression or SQL code) is usually bottom-up or top-down following the dependencies between components as discussed by Sun et al. (2020). The order to give an NL response is first analyses then decisions as discussed by CoT (Wei et al., 2022). The order to solve a puzzle (such as the example 4x4 Sudoku game in Figure 2) is usually from easy parts to hard parts, because the hard parts become easier when the easy parts are correctly solved. (That is also verified in §5.1 where our CASR model learns to solve easy parts before hard ones.) Obviously, people give answers to different tasks in various orders with respect to all kinds of dependencies.

Mimicking human behavior, some existing works design specific intermediate sequences to solve the dependency order challenge. *E.g.*, templates (Hua & Wang, 2020) or heuristic rules (Zhang et al., 2018; Tan et al., 2021) are applied in autoregressive NL generation, allowing models to generate some parts (intermediate sequences) before the other parts in an answer via iterative refinement (rather than one-pass decoding). However, 1) it's really hard to design the best heuristic order and easy to miss intrinsic dependencies for some tasks, and we need expert knowledge or manual efforts to design specific heuristic orders for all different tasks. Besides, 2) when using teacher forcing strategy to parallel train all refinement iterations, additional exposure bias occurs.

In this paper, **CASR** (Generating **C**omplex Sequences with **A**uto-regressive **S**elf-Boost **R**efinement) framework is proposed by us to: 1) *decide intermediate sequences of complex answers for different tasks in a data-driven way*, 2) *avoid additional exposure bias.* As shown in Figure 1 and will be discussed in §3, in CASR we design a model architecture (§3.2) to not only take in the original input X, but also the previous prediction $\hat{Y}^{t-1}$ in both *training* and inference. A special process (§3.1) is designed to train refinement models M^t for each step $t = 0, 1, ..., T-1$. To enhance the performance on downstream tasks, CASR models could be initialized with pre-trained language models (§3.3) such as T5 (Raffel et al., 2020), and even trained in a "Continue" way (§3.4) by initializing M^t

with M^{t-1}. For CASR models to be more efficient, we also explore the parameter-efficient model designs (SepEnc *vs.* UniEnc in §3.2), and tuning with parallel adapters (§3.3).

We evaluate CASR (and several baselines) on WebQSP, MTOP, KVRET, and Sudoku in §4, achieving SOTA performance on KVRET. In detail, CASR improves F1 on WebQSP from 70.81 to 74.81, EM on MTOP from 78.64 to 81.92, Micro F1 on KVRET from 67.40 to 70.00[1], and accuracy on Sudoku from 70.93 to 97.28. We find the optimal CASR design (§4.2) is "Fine-tuning" "SepEnc" with "Continue" strategy. We also do empirical studies on complexity (§5.1), attention map (§5.2), and visualize cases (§5.3). Then we find that, CASR benefits hard sequences more than easy ones, and CASR can indeed correct the wrong part of the previous prediction according to other parts it depends on (as shown in Figure 8).

In summary, our major contributions are: First, we point out the challenge to generate complex sequences due to the existence of multi-hop dependency chains and conduct a comprehensive review on existing iterative refinement methods. Second, we propose an autoregressive self-boost refinement framework, CASR, to decide intermediate sequences of complex answers in a data-driven way. The code of CASR framework is open sourced in the repository at `https://github.com/RalphHan/CASR`. Third, we conduct experiments and empirical studies on four complex tasks to show CASR works and interpret how it works.

2 Related Work

Table 1: The Objectives of Sequence Generation Methods.

Method	Objective
NAT (Gu et al., 2017)	$\prod_i P(y_i\|X)$
INAT (Lee et al., 2018)	$\prod_i P(y_i^t\|X, \hat{Y}^{t-1})$
Levenshtein (Gu et al., 2019)	Imitate an expert policy to delete and insert
L2R (Wu et al., 2016)	$\prod_i P(y_i\|X, y_{i-1}, ..., y_1)$
XLNet (Yang et al., 2019)	$\prod_i P(y_{z_i}\|X, y_{z_{i-1}}, ..., y_{z_1})$
Bidirectional (Zhang et al., 2018)	$\prod_i P(y_i\|X, y_{i+1}, ..., y_n) \cdot \prod_i P(y_i\|X, y_{i-1}, ..., y_1, C^{R2L})$
Progressive (Tan et al., 2021)	$\prod_i P(y_i^t\|X, Y^{t-1}, y_{i-1}^t, ..., y_1^t)$
Ours	$\prod_i P(y_i^t\|X, \hat{Y}^{t-1}, y_{i-1}^t, ..., y_1^t)$

Table 2: The Number of Samples in Train, Dev, and Test Splits of WebQSP, MTOP, KVRET, and Sudoku.

Task	Train	Dev	Test
WebQSP	2673	309	1639
MTOP	15667	2235	4386
KVRET	6291	777	808
Sudoku	800K	100K	100K

2.1 Sequence Generation Methods

In recent years, many studies have been made on seq2seq generation. For input X and target Y sequences, the ultimate objective is to maximize $P(Y|X)$. As shown in Table 1, different method formulates the objective differently. These methods can be divided into non-autoregressive (row 1-3) and autoregressive (row 4-7) ones. Highly related ones are run as baselines in Table 5.

NAT (Gu et al., 2017) assumes that each token in sequence Y is mutually independent from each other, thus formulating $P(Y|X)$ as $\prod_i P(y_i|X)$. To reduce the modeling bias in NAT, iterative refinement is applied on answers in INAT (Lee et al., 2018). Levenshtein Transformer (Gu et al., 2019) further breaks down each refinement step into deletion, insertion, and classification so as to allow the model to edit the generation in a non-autoregressive way. Despite their efficiency, non-autoregressive models perform relatively poor comparing to autoregressive ones (see §4.4).

L2R (Wu et al., 2016) generation tries to decode from left to right, which is also adopted by GPT (Radford et al., 2018), basic transformer decoder (Vaswani et al., 2017), VAE decoder (Yu et al., 2020), and RNN decoder (Xu et al., 2015; Xia et al., 2017). XLNet (Yang et al., 2019) allows the model to generate with any given order Z, but cannot decide by itself the best order. Bidirectional decoder (Zhang et al., 2018) generates backward (R2L) then forward (L2R), which handles at most 2-step logic. It trains an additional R2L decoder from scratch, without leveraging generation capabilities of PLMs as in CASR (see §3.3).

PAIR (Hua & Wang, 2020) is used in controlled text generation with a template (constructed from a set of provided key phrases and placeholders) as input. It fills and refines the placeholders in an auto-regressive way. The template, the sequence length and position of placeholders remain

[1]70.00 achieves the SOTA for micro f1 on KVRET, and the previous SOTA is 67.88 (Xie et al., 2022).

unchanged during refinement. Similarly, chain of thought prompting (Wei et al., 2022) constructs the intermediate process through manually provided templates. These two methods are not chosen as baselines in §4 because they both require expert designs for each specific task.

Progressive generation (Tan et al., 2021) is proposed to remove the requirement of templates and the limitation of fixed length during refinement. It breaks down the vocabulary into multiple stages according to word importance (average of tf-idf). Important words are generated first during early refinement steps, while other words (both important and unimportant) are generated later. Note that in Table 1, Y^{t-1} (row 7) denotes the intermediate sequence produced by the heuristics of progressive generation, but $\hat{Y}^{t-1}$ (row 8) is the previous prediction generated by our CASR model (see §3).

2.2 Tasks with Complex Answers

The four complex-answer tasks referred in §1 will be further introduced in this section.

WebQSP (Yih et al., 2016) is a classic dataset for KBQA(Knowledge Base Question Answering). The input consists of a knowledge graph and an NL query, and the output is an s-Expression which can be executed on the knowledge graph. The SOTA method of WebQSP (F1=83.6%) is ranking with bootstrapping negative samples (Ye et al., 2022).

MTOP (Li et al., 2021) is a benchmark for comprehensive multilingual task-oriented semantic parsing. The input consists of a list of API calls and an NL query, and the output is a tree-based TOP Representation that can be executed.

KVRET (Eric et al., 2017) is a benchmark for table conversation. The input consists of a table and an NL query, and the output is an NL response corresponding to the dialog. The SOTA method of MTOP (EM=86.78%) and KVRET (Micro F1=67.88%) is multi-task (20+ tasks) prefix tuning with T5-3B as the backbone (Xie et al., 2022).

Sudoku (PARK) is an open dataset on Kaggle. Its game target is to fill the blanks correctly with the constraint that any two numbers in the same row, column, and house shouldn't have the same value. We choose Sudoku as an intuitive toy task for better demonstration of ideas in this paper.

Examples of WebQSP (Figure 5), MTOP (Figure 6) and KVRET (Figure 7) are shown in Appendix §A. The number of samples in each split of the four tasks is shown in Table 2.

3 CASR Framework

As shown in Figure 1 and introduced in §1 and §2.1, the key idea of CASR framework is to take the previous-step prediction $\hat{Y}^{t-1}$ as part of the current-step (t) input for generating a refined output $\hat{Y}^{t}$. The iterative inference process in CASR follows the idea: As shown in Algorithm 1, the prediction at castep t is $\hat{Y}^{t}$ ($0 \leq t < T$) generated by the corresponding CASR model M^t, where the input to M^0 is the original input X, and the input to M^t ($t > 0$) is $(X, \hat{Y}^{t-1})$. Following the notions in §2.1, X, Y, and $\hat{Y}$ denote the input, the ground truth and the prediction, respectively. The iterative refinement process in CASR will take at most T **caste**s (CASR steps).

The undetermined M^t models in the CASR inference process lead to more questions: 1) How to design the training process and objectives that match the inference process? 2) How to design CASR model architecture that leverages existing PLMs and takes the extra $\hat{Y}^{t-1}$ as input? 3) When leveraging a large PLM, can we train and save CASR models M^t with less parameters? 4) Can the training and inference process be more effective by exploiting the relationships among CASR models M^t of each castep? In the following, we will discuss these problems one by one.

3.1 Training Process

As formulated in Algorithm 2, the training process is also iterative.

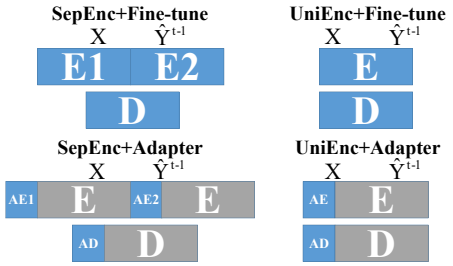
At the 0-th castep, we train the model M^0 normally on train set with the objective to maximize $P(Y|X)$ (as discussed in §2.1). When the training is done, we use M^0 to generate prediction $\hat{Y}^0$ for each X in train and dev sets with beam searching (Ney et al., 1987).

At the t-th castep ($0 < t < T$), we train M^t to refine the last prediction $\hat{Y}^{t-1}$. The objective becomes maximizing $P(Y|X, \hat{Y}^{t-1})$ on train set. After training, we generate $\hat{Y}^t$ on all train and dev sets to compose the samples $(X, Y, \hat{Y}^t)$ for the next castep $t+1$.

After the training process, T versions of CASR model M^t ($0 \leq t < T$) are available for the inference process. Because the distribution of $\hat{Y}^{t-1}$ could change with t, and different casteps may learn different refinement patterns, by default all the T versions are saved.

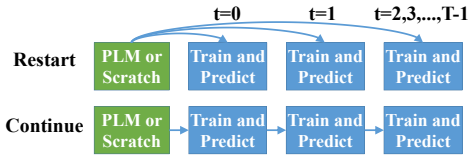
In the following, we will discuss more details of the algorithms on M^t model architecture (§3.2), tuning (§3.3), and initialization (§3.4). More discussions on result selections are in Appendix §B.

3.2 Model Architecture



AE and AD denote adapters of encoder and decoder. Blue and gray segments are the parameters to be trained and frozen, respectively.

Figure 3: Model Architectures and Parameter Efficiency Choices.



The arrows begin at the source of initial weights of M^t.

Figure 4: Initialization Strategies.

The next problem after the training process is how to design a model architecture which could take $(X, \hat{Y}^{t-1})$ as input and generate $\hat{Y}^t$. Starting from any Transformer encoder-decoder (Vaswani et al., 2017) based model (such as T5, or an untrained Transformer), in CASR we provide two modification approaches to take in $\hat{Y}^{t-1}$.

As shown in Figure 3, **SepEnc** has two encoders to encode X and $\hat{Y}^{t-1}$ separately (encode before concat), and **UniEnc** adopts only one encoder for both inputs (concat before encode). The output sequence length of SepEnc and UniEnc are both len(X) + len($\hat{Y}^{t-1}$). Let H denote the encoder output. For SepEnc, $H = Concat(Encoder_1(X), Encoder_2(\hat{Y}^{t-1}))$, and for UniEnc, $H = Encoder(Concat(X, \hat{Y}^{t-1}))$.

Both approaches have their own advantages: SepEnc naturally handles the distribution differences between X and $\hat{Y}^{t-1}$, while UniEnc requires fewer model parameters. UniEnc forces one encoder to handle two kinds of sequences with potentially different types of contents and lengths. As a trade-off, SepEnc brings two times larger encoding part. And the decoder generates answers from H, just like the classic transformer decoder (Vaswani et al., 2017).

3.3 Parameter Efficiency

Both SepEnc and UniEnc approaches in §3.2 could take a PLM (pre-trained language model, such as T5) as starting point for model parameter initialization (more details in §3.4). In this way, CASR could leverage the existing knowledge from the PLM to enhance performances on downstream tasks (*e.g.*, the ones in §2.2).

However, it's costly to fine-tune all the parameters from a large PLM and save all the parameters of all the T CASR models M^t. Thus, besides the fine-tuning approach, in CASR we also try parameter-efficient tuning. As shown in Figure 3, parallel adapters (He et al., 2021) could be added to every encoder and decoder. By adopting adapters, the parameters from the PLM are frozen, and only the parameters of the adapters are trained. In other words, one only needs to save the adapter (rather than the whole model) parameters for T times.

3.4 INITIALIZATION STRATEGY

In Algorithm 2, at the beginning of each castep, we initialize the parameter weights of M^t with a chosen strategy from {Restart, Continue}. Illustrations of each strategy are shown in Figure 4.

"**Restart**" means initializing each M^t with the same PLM weights (or from scratch). For SepEnc architecture, Both $Encoder_1$ and $Encoder_2$ can be initialized with the parameter weights of the PLM encoder. "**Continue**" means initializing with the best checkpoint from the previous castep. In this way, M^t inherits the knowledge from M^{t-1} to avoid the potential cold start issue at castep t.

In the "Restart" strategy, when comparing the late castep with the early castep, the improvement is only brought by refinement, rather than the extra continuous training steps (which exist in the "Continue" strategy). Note that, we design "Restart" mainly to decouple the effects of refinement from training steps, which is necessary for controlled experiments in §4.

4 EXPERIMENTS

In this section, we run controlled experiments to evaluate CASR designs (§3) on WebQSP (Yih et al., 2016), MTOP (Li et al., 2021), KVRET (Eric et al., 2017) and Sudoku (PARK) (§2.2), and compare CASR with high-related baselines (§2.1).

4.1 EXPERIMENT SETUP

In this section, we explain how we design the controlled experiments and reproduce the high-related baselines. For more details about hyperparameter and resource consumption (number of parameters, training/inference time, *etc.*) please see Appendix §C.

4.1.1 CONTROLLED EXPERIMENTS

In Table 3, we apply CASR (with max castep $T = 3$) on the 24-layer generative PLM T5-base (Raffel et al., 2020). Controlled experiments Ⓐ∼Ⓔ are run to compare the variations of **Param**eter **E**fficiency (§3.3), **Init**ialization Strategy (§3.4) and Model **Arch**itecture (§3.2). The evaluation metrics (on test sets) are F1 score for WebQSP, template accuracy and exact match for MTOP, and BLEU and Micro F1 scores for KVRET. For each row, the average (**AVG**) of all metrics (on test set) is calculated to provide an overall merged score.

Table 3: Controlled Experiments of CASR Framework on WebQSP, MTOP, and KVRET.

	t	CASR Variations			WebQSP	MTOP		KVRET		
		Param.E.	Init.	Arch.	F1	Acc.	EM	BLEU	Micro F1	AVG
Ⓐ0	0	Fine-tune	Restart	SepEnc	70.81	82.49	78.64	18.33	67.40	63.53
Ⓑ0		Fine-tune	Continue	SepEnc						
Ⓒ0		Fine-tune	Restart	UniEnc						
Ⓓ0		Fine-tune	Continue	UniEnc						
Ⓔ0		Adapter	Restart	SepEnc	70.20	77.34	73.10	16.38	65.59	60.52
Ⓐ1	1	Fine-tune	Restart	SepEnc	73.03	82.92	78.93	18.17	67.65	64.14
Ⓑ1		Fine-tune	Continue	SepEnc	**74.61**	85.07	81.19	18.55	**70.00***	**65.88**
Ⓒ1		Fine-tune	Restart	UniEnc	73.14	82.83	78.80	**18.80**	67.88	64.29
Ⓓ1		Fine-tune	Continue	UniEnc	74.16	84.86	81.24	18.06	69.12	65.49
Ⓔ1		Adapter	Restart	SepEnc	67.92	77.43	73.21	16.60	65.86	60.20
Ⓐ2	2	Fine-tune	Restart	SepEnc	73.09	82.92	78.93	18.26	67.70	64.18
Ⓑ2		Fine-tune	Continue	SepEnc	**74.81**	**85.61**	**81.69**	**19.12**	**69.85**	**66.22**
Ⓒ2		Fine-tune	Restart	UniEnc	73.14	82.83	78.77	18.78	68.06	64.32
Ⓓ2		Fine-tune	Continue	UniEnc	74.32	**85.66**	**81.92**	18.34	68.80	65.81
Ⓔ2		Adapter	Restart	SepEnc	68.13	77.45	73.23	16.63	65.66	60.22

The three row blocks Ⓐ0∼Ⓔ0, Ⓐ1∼Ⓔ1 and Ⓐ2∼Ⓔ2 correspond to M^0, M^1 and M^2 models at **Castep** $t = 0, 1, 2$, respectively. By definitions in §3, row Ⓐ0∼Ⓓ0 share the same result because initialization and architecture choices do not influence M^0 models.

Since CASR converges on Sudoku with larger T, we evaluate it separately. In Table 4, We train 12-layer encoder-decoder transformers from scratch with max castep $T = 5$. The metric is accuracy (% correctly filled blanks). For a 9×9 board, the input and output sequences of a Sudoku game are both 81-number serialized sequences. During training, we apply supervision only on blanks and apply constrained generation to

Table 4: Sudoku Testing Results.

t	Restart	Continue	Continue w/o $\hat{Y}^{t-1}$
0	70.93		
1	77.00	85.71	84.28
2	78.69	92.48	89.16
3	79.11	95.66	91.13
4	79.21	**97.28**	92.09

make sure the model only predicts on blanks. And we fill the predictions back to the blanks of the input as the combination of X and $\hat{Y}$, at the beginning of each castep $t > 0$. In other words, we use a special UniEnc to encode the combination, with $H = Encoder(Combine(X, \hat{Y}^{t-1}))$. "Restart" and "Continue" correspond to the same choices as Ⓒ and Ⓓ in Table 3. "Continue w/o $\hat{Y}^{t-1}$" denotes that the previous predictions are not put back to the blanks of the input at each castep, which is equivalent to training the origin model T times steps as usual.

4.1.2 BASELINE COMPARISONS

In Table 5, we run some high-related baselines as introduced in §2.1. We run INAT and Levenshtein with the official code provided by fairseq[2], where self-attention layers of the encoder and the decoder are initialized with bert, leaving cross-attention layers of the decoder trained from scratch. To decouple the effect from PLMs and make a fairer comparison between NAR and AR methods, we add row Ⓧc, where the decoder of CASR is not initialized with T5 but from scratch (random weights), marked as "/Dec". In row Ⓛ2, "CASR-L" denotes changing the backbone of row Ⓑ2 from T5-base to T5-large. For fair comparisons, the original backbones in Bidirectional Decoder (GRU) and Progressive Generation (Bart) implementations are changed to T5-base (same as CASR), and set $T = 3$ (same as CASR) for Progressive Generation.

Table 5: Comparisons between CASR and Baselines.
"NAR" and "AR" denote non-autoregressive and autoregressive refinement.
"–.01–" denotes the metrics are lower than 0.01.

		Methods	WebQSP	MTOP		KVRET		AVG
			F1	Acc.	EM	BLEU	Micro F1	
Ⓧa	NAR	INAT (Lee et al., 2018)	—.01—					0.00
Ⓧb	NAR	Levenshtein (Gu et al., 2019)	22.54	—.01—				4.51
Ⓧc		CASR Ⓑ / Dec	4.08	63.02	12.93	7.18	10.17	19.48
Ⓧd		Bidirectional (Zhang et al., 2018)	68.43	61.33	57.84	8.98	54.77	50.27
Ⓧe	AR	Progressive (Tan et al., 2021)	72.05	80.80	77.04	15.49	64.54	61.98
Ⓑ0		Finetune	70.81	82.49	78.64	**18.33**	67.40	63.53
Ⓑ2		CASR	**74.81**	**85.61**	**81.69**	**19.12**	**69.85**	**66.22**
Ⓛ2		CASR-L	**77.99**	**87.73**	**84.54**	18.14	**68.80**	**67.44**

4.2 BEST CASR DESIGNS

We achieve the SOTA performance on Micro F1 of KVRET, which is **70.00**. The previous SOTA is 67.88 (Xie et al., 2022) based on T5-3B, and we beat it with a smaller T5-base.

For the parameter initialization choices, "Continue" strategy brings better performance than "Restart". This is observed by comparing Ⓐ with Ⓑ rows, Ⓒ with Ⓓ rows in Table 3, and comparing "Restart" and "Continue" columns in Table 4.

For the model architectures, by comparing Ⓐ with Ⓒ rows, Ⓑ with Ⓓ rows, we find that there is no great performance gap between SepEnc and UniEnc, and SepEnc is slightly better than UniEnc when combined with "Continue".

For the parameter efficiency, by comparing Ⓔ with Ⓐ rows, we can find that adapter-tuning perform worse than fine-tuning. As the cost of freezing PLM parameters for efficiency, performance drops as expected when bringing adapter-tuning.

4.3 ANALYSIS ON SELF-BOOST REFINEMENT

As introduced in §3.4, we design "Restart" to decouple the effects of refinement from fitting the training set longer than t=0. The variations with "Restart" strategy (Ⓐ, Ⓒ and Ⓔ in Table 3, "Restart" column in Table 4) demonstrates how self-boost refinement could improve the quality of generated sequences. Here the improvement is only brought by refinement since CASR models at all casteps are initialized with the same PLM/Scratch version and take the same training steps.

In Table 4, comparing "Continue" with "Continue w/o $\hat{Y}^{t-1}$", we find that self-boostly feeding $\hat{Y}^{t-1}$ to the model is indeed helpful, which improves the accuracy from 92.09% to 97.28%.

Interestingly, the performance gap between castep 0 and 1 is much larger than the gap between t to $t + 1$ ($t > 0$). One possible explanation is that later refinements have reached the upper bound.

[2] https://github.com/facebookresearch/fairseq

As we can see in Table 4, Sudoku requires more refinement steps (5 in our case) for the gap to converge. This shows there are more logical steps required and more dependencies exist in the generated Sudoku answers.

4.4 Comparison with Baselines

In the NAR block of Table 5, we find the results of NAR methods are not on par with AR methods. With case study (Figure 10), we find the ability of NAR methods to learn syntax is poor. Comparing row (Xb) and (Xc), we find the poorness of NAR methods doesn't come from the initialization of the decoder. Comparing row (Xd), (Xe), and (B0), we find that those AR refinement methods using heuristic intermediate sequences are indeed harmful (worse than directly fine-tuning) for complex sequence generation. Compare row (B2) and (L2), we find larger backbone (T5-base → T5-large) is overall helpful (UnifiedSKG (Xie et al., 2022) observes the same phenomenon that T5-large is worse than T5-base on KVRET).

In summary, following conclusions can be drawn for CASR:

- Self-boost refinement leads to better outputs than vanilla AR (one-pass L2R) generation, NAR refinement generation, and heuristic-rule-based AR refinement generation.
- In general, the best combination is "Fine-tuning" "SepEnc" with "Continue" strategy.

5 Empirical Studies

Table 6: Result on Different Answer Length.
Produced by row (B) in Table 3 (Finetune+Continue+SepEnc). Δ_t denotes the improvement from castep $t-1$ to t.

	Task	WebQSP	MTOP		KVRET			
t	Metric	F1	Acc	Match	Bleu	Micro F1	AVG	Δ_t
0	Short	78.80	85.23	83.65	14.05	65.91	65.53	-
	Middle	83.07	84.83	80.51	21.93	71.16	68.30	-
	Long	50.51	77.31	71.62	15.08	65.69	56.04	-
1	Short	80.12	87.21	85.50	12.17	65.93	66.19	0.66
	Middle	86.41	87.19	83.01	19.81	71.51	69.59	1.29
	Long	57.52	80.71	74.95	16.33	69.48	59.80	3.76
2	Short	80.12	87.96	86.25	12.32	64.52	66.23	0.05
	Middle	86.41	87.53	83.34	19.77	71.79	69.77	0.18
	Long	58.13	81.26	75.36	17.00	69.18	60.19	0.39

Table 7: The average difficulty and ratio (sum to 1) of each correct-solving castep, produced by the setting of column "Continue" in Table 4.

t	AVG Difficulty	Ratio
0	82.80	63.54%
1	86.95	18.48%
2	88.60	8.50%
3	89.83	4.25%
4	90.86	2.51%
5 (failed)	92.21	2.72%

In this section, we conduct empirical studies to interpret how CASR works.

In §5.1, we verify that CASR brings more improvement on more complex sequence. Also, we find CASR model may implicitly learns dependency because it solves easy parts first during refinements.

In §5.2, the cross-attention map between $\hat{Y}^t$ and $\hat{Y}^{t-1}$ implies the informativeness and correctness of $\hat{Y}^{t-1}$. Meanwhile, from a microscopic perspective, each row in the cross-attention map implies the foundation of changing a component, where the hot-attended parts can be considered as the cause of changes.

In §5.3, through case studies we show the intermediate steps generated by CASR models.

5.1 Analysis on Complexity

CASR helps more on problems with more complexity. For WebQSP, MTOP, and KVRET, we assume that the complexity of a problem (X, Y) is positively correlated with the length of its answer. Thus, we divide each test dataset into "Short", "Middle" and "Long" equal splits according to len(Y), the length of ground truth answer. As we can see in Table 6, CASR brings the more improvement to the problems with "Long" answers than "Middle" and "Short" ones.

Take Sudoku as example, we find that **CASR solves easy blanks before difficult blanks during refinements.** In late casteps, the generation benefits more from refinement by gradually handling

complex dependencies. These are shown in Table 7 – As castep t increases, the average difficulty of remaining blanks increase. Here we formulate the difficulty[3] of a blank cell as: $Difficulty = (r-1)\cdot(c-1)\cdot(h-1)$ where r, c, and h denote the number of blanks in the same row, column, and house of the blank. Also, we can define the correct-solved castep for a blank: If a blank is correctly filled at castep t and remains the same for the rest casteps, then the blank is solved at castep t and t is the correct-solved castep for the blank. Otherwise, by default the correct-solved castep is set to T. The "Ratio" column in Table 7 means the percentage of blanks is correctly solved at castep t.

5.2 Analysis on Attention

To measure to what extend does $\hat{Y}^t$ attend to X or $\hat{Y}^{t-1}$ in cross-attention, we define the density. The cross attention map, A, is a tensor that displays how the tokens in $\hat{Y}^t$ attend to H. Therefore, the size of A is [num-layers, num-heads, len($\hat{Y}^t$), len(X)+len($\hat{Y}^{t-1}$)]. We define the density to X and to $\hat{Y}^{t-1}$ as: $D_X = A[..., : len(X)].mean()$ and $D_{\hat{Y}^{t-1}} = A[..., len(X) :].mean()$, which are the average attention weight among each layer, head, and token, for X and for $\hat{Y}^{t-1}$.

Similar to Table 6, We sort the samples according to their $D_{\hat{Y}^{t-1}}$, and divide a test dataset into sparse, middle, and dense splits, each holding 1/3 of the total test set. We evaluate the metrics of $\hat{Y}^{t-1}$ on the three splits. We find the performance of $\hat{Y}^{t-1}$ increases with $D_{\hat{Y}^{t-1}}$. Besides, we compute the average D_X and $D_{\hat{Y}^{t-1}}$ over all samples in the test set, and find that $D_{\hat{Y}^{t-1}}$ grows with t. Therefore, $\hat{Y}^{t-1}$ **becomes more and more informative thus be attended to by** $\hat{Y}^t$**.** Please refer to Table 10 and Table 11 in Appendix §D for more details.

Meanwhile, we draw top-5 positions in cross-attention map (filter out non-blank cells and self) for each changed cell in Figure 8. As we can see, the corrected cell usually attends to cells in the same row, column, or house.

5.3 Case Studies

As examples, the intermediate CASR prediction of each castep are shown in Figure 9 of Appendix §D. We can find that as castep t increases, there is more green (keywords) and less red (mistakes) parts, which means $\hat{Y}^t$ becomes closer to Y. In the WebQSP case, the wrong part (marked red) of $\hat{Y}^0$ is deleted by late casteps. In the MTOP case, we find that CASR correctly generates the right part ("[SL:RECIPES_INCLUDED_INGR EDIENT dairy]") of the answer before the left part ("[SL:MET HOD_RECIPES recipe]"), which one-pass L2R decoding cannot achieve. For Sudoku, an interesting phenomenon occurs. Although the wrong predictions (marked red) become fewer with the growth of castep, some previous correct predictions are modified to wrong ones, and then modified back. This can be considered a kind of exploration.

We also present the predictions from other experiments (baselines and CASR-L, as discussed in Table 5) in Figure 10, and we find it a challenge for NAR methods to generate complex sequences.

6 Conclusion

In this paper, CASR framework is proposed by us to generate **C**omplex sequences with **A**utoregressive **S**elf-Boost **R**efinement, which can decide intermediate sequences of complex answers in a data-driven way with no need for expert knowledge or manual efforts. Through controlled experiments, we find the optimal design in CASR is **Fine-tuning SepEnc** with **Continue** strategy. Also, the interpretability of CASR is enhanced via empirical studies on problem complexity and attention dependencies. CASR sheds more light on the sequence generation methods, especially on complex tasks (*e.g.* automatic data analysis and drug discovery), for future research.

[3]Difficulty: According to this definition, when a blank is the only blank in a row, column, or house, the difficulty is 0, which makes sense because the blank can be directly solved. The larger r, c, and h are, the more difficult it is to solve the blank.

ETHICS STATEMENT

Datasets This work collects the public dataset for research purposes. We believe there is no privacy issue, because Sudoku, WebQSP, MTOP, and KVRET are accessible to the public.

Models This work leverages T5-base and T5-large, which are pre-trained on C4, a clean corpus. Therefore, we can make sure that CASR will not produce discriminatory answers.

Computational Resources We train on 4 Tesla V100 GPUs, and the training and inference time consumption is show in §9, which is acceptable considering the performance gain.

REPRODUCIBILITY STATEMENT

We describe experiment setup in §4.1 and provide the complete details such as hyperparameter (in §C.1) and resource consumption (in §C.2). please see §C. Meanwhile, we make our code open source in this repository `https://github.com/RalphHan/CASR`. These resources should be sufficient to reproduce results of the paper.

ACKNOWLEDGMENTS

This research was partly supported by Shenzhen Stable Supporting Program (WDZC20200820200655001).

REFERENCES

Mihail Eric, Lakshmi Krishnan, François Charette, and Christopher D. Manning. Key-value retrieval networks for task-oriented dialogue. In Kristiina Jokinen, Manfred Stede, David DeVault, and Annie Louis (eds.), *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017*, pp. 37–49. Association for Computational Linguistics, 2017. doi: 10.18653/v1/w17-5506. URL https://doi.org/10.18653/v1/w17-5506.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-autoregressive neural machine translation. *CoRR*, abs/1711.02281, 2017. URL http://arxiv.org/abs/1711.02281.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 11179–11189, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/675f9820626f5bc0afb47b57890b466e-Abstract.html.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *CoRR*, abs/2110.04366, 2021. URL https://arxiv.org/abs/2110.04366.

Xinyu Hua and Lu Wang. PAIR: planning and iterative refinement in pre-trained transformers for long text generation. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 781–793. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.57. URL https://doi.org/10.18653/v1/2020.emnlp-main.57.

Jintae Kim, Sera Park, Dongbo Min, and Wankyu Kim. Comprehensive survey of recent drug discovery using deep learning. *International Journal of Molecular Sciences*, 22(18):9983, 2021.

Celine Lee, Justin Gottschlich, and Dan Roth. Toward code generation: A survey and lessons from semantic parsing. *CoRR*, abs/2105.03317, 2021. URL https://arxiv.org/abs/2105.03317.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 1173–1182. Association for Computational Linguistics, 2018. doi: 10.18653/v1/d18-1149. URL https://doi.org/10.18653/v1/d18-1149.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 7871–7880. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.703. URL https://doi.org/10.18653/v1/2020.acl-main.703.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. MTOP: A comprehensive multilingual task-oriented semantic parsing benchmark. In Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pp. 2950–2962. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.eacl-main.257. URL https://doi.org/10.18653/v1/2021.eacl-main.257.

Congbo Ma, Wei Emma Zhang, Mingyu Guo, Hu Wang, and Quan Z. Sheng. Multi-document summarization via deep learning techniques: A survey. *CoRR*, abs/2011.04843, 2020. URL https://arxiv.org/abs/2011.04843.

Mishaim Malik, Muhammad Kamran Malik, Khawar Mehmood, and Imran Makhdoom. Automatic speech recognition: a survey. *Multim. Tools Appl.*, 80(6):9411–9457, 2021. doi: 10.1007/s11042-020-10073-7. URL https://doi.org/10.1007/s11042-020-10073-7.

Hermann Ney, Dieter Mergel, Andreas Noll, and Annedore Paeseler. A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '87, Dallas, Texas, USA, April 6-9, 1987*, pp. 833–836. IEEE, 1987. doi: 10.1109/ICASSP.1987.1169844. URL https://doi.org/10.1109/ICASSP.1987.1169844.

Hariom A. Pandya and Brijesh S. Bhatt. Question answering survey: Directions, challenges, datasets, evaluation matrices. *CoRR*, abs/2112.03572, 2021. URL https://arxiv.org/abs/2112.03572.

KYUBYONG PARK. 1 million sudoku games. https://www.kaggle.com/datasets/bryanpark/sudoku. Accessed: 2022-05-20.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, and Lu Zhang. Treegen: A tree-based transformer architecture for code generation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8984–8991. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6430.

Bowen Tan, Zichao Yang, Maruan Al-Shedivat, Eric P. Xing, and Zhiting Hu. Progressive generation of long text with pretrained language models. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy

Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 4313–4324. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.341. URL https://doi.org/10.18653/v1/2021.naacl-main.341.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022. URL https://arxiv.org/abs/2201.11903.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

Yingce Xia, Fei Tian, Lijun Wu, Jianxin Lin, Tao Qin, Nenghai Yu, and Tie-Yan Liu. Deliberation networks: Sequence generation beyond one-pass decoding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1784–1794, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/c6036a69be21cb660499b75718a3ef24-Abstract.html.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir R. Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *CoRR*, abs/2201.05966, 2022. URL https://arxiv.org/abs/2201.05966.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 2048–2057. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/xuc15.html.

Shuoheng Yang, Yuxin Wang, and Xiaowen Chu. A survey of deep learning techniques for neural machine translation. *CoRR*, abs/2002.07526, 2020. URL https://arxiv.org/abs/2002.07526.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 5754–5764, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. RNG-KBQA: generation augmented iterative ranking for knowledge base question answering. In Smaranda

Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 6032–6043. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.417. URL https://doi.org/10.18653/v1/2022.acl-long.417.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics, 2016. doi: 10.18653/v1/p16-2033. URL https://doi.org/10.18653/v1/p16-2033.

Meng-Hsuan Yu, Juntao Li, Danyang Liu, Bo Tang, Haisong Zhang, Dongyan Zhao, and Rui Yan. Draft and edit: Automatic storytelling through multi-pass hierarchical conditional variational autoencoder. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1741–1748. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5538.

Xiangwen Zhang, Jinsong Su, Yue Qin, Yang Liu, Rongrong Ji, and Hongji Wang. Asynchronous bidirectional decoding for neural machine translation. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 5698–5705. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16784.

Mengyu Zhou, Wang Tao, Pengxin Ji, Han Shi, and Dongmei Zhang. Table2analysis: Modeling and recommendation of common analysis patterns for multi-dimensional data. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 320–328. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5366.

A Examples

UnifiedSKG (Xie et al., 2022) did a great job to formulate 21 structured knowledge grounding tasks into a unified seq2seq form, and we apply their input-output form of WebQSP, MTOP, and KVRET in CASR. And for Sudoku, we apply their original serialization method, which flattens the 9x9 game table row by row into an 81-dimensional number list. Examples of WebQSP (Figure 5), MTOP (Figure 6) and KVRET (Figure 7) are shown.

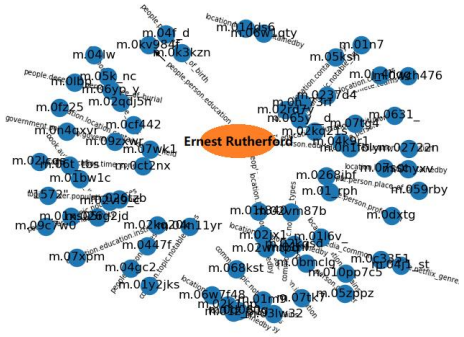
B Best Result Selection

The default strategy is to run all T casteps and take the Last output $\hat{Y}^{T-1}$ as the final prediction. However, one could argue that for a specific input X, the best prediction may occur early and more refinements may hurt. Following this we try to pick the $\hat{Y}^t$ $(0 \leq t < T)$ with the highest prediction probability (given by M^t): $\hat{Y} = \hat{Y}^{t_{pick}}$ where $t_{pick} = argmax_t P(\hat{Y}^t | X, \hat{Y}^{t-1}, M^t)$. However, that doesn't always work and we leave it as furture work.

C Experiment Details

C.1 Hyperparameters

On WebQSP, MTOP and KVRET tasks, we initialize CASR models with T5-base (Raffel et al., 2020), a 24-layer generative PLM. We set max casteps $T = 3$. Max epoch E is set to 2K, 4K, and



Request: what school did sir ernest rutherford go to?
Answer: (JOIN (R education.education.institution)
(JOIN (R people.person.education) m.02m7r))

Figure 5: An Example of WebQSP.

IN:GET: MESSAGE, WEATHER, ALARM, INFO_RECIPES, STORIES_NEWS, REMINDER, RECIPES, EVENT, CALL_TIME, LIFE_EVENT, INFO_CONTACT, CONTACT, TIMER, REMINDER_DATE_TIME, AGE, SUNRISE, EMPLOYER, EDUCATION_TIME, JOB, AVAILABILITY, CATEGORY_EVENT, CALL, EMPLOYMENT_TIME, CALL_CONTACT, LOCATION, TRACK_INFO_MUSIC, SUNSET, MUTUAL_FRIENDS, UNDERGRAD, REMINDER_LOCATION, ATTENDEE_EVENT, MESSAGE_CONTACT, REMINDER_AMOUNT, DATE_TIME_EVENT, DETAILS_NEWS, EDUCATION_DEGREE, MAJOR, CONTACT_METHOD, LIFE_EVENT_TIME, LYRICS_MUSIC, AIRQUALITY, LANGUAGE, GENDER, GROUP
IN:SEND: MESSAGE
IN:SET: UNAVAILABLE, RSVP_YES, AVAILABLE, DEFAULT_PROVIDER_MUSIC, RSVP_INTERESTED, DEFAULT_PROVIDER_CALLING, RSVP_NO
IN:DELETE: REMINDER, ALARM, TIMER, PLAYLIST_MUSIC
.....
IN:DISPREFER:
IN:HELP: REMINDER
IN:FOLLOW: MUSIC

Request: does this recipe have dairy?
Answer: [IN:IS_TRUE_RECIPES [SL:METHOD_RECIPES recipe] [SL:RECIPES_INCLUDED_INGREDIENT dairy]]

Figure 6: An Example of MTOP.

poi	poi_type	address	distance	traffic_info
pizza my heart	pizza restaurant	528 anton ct	5 miles	moderate traffic
whole foods	grocery store	819 alma st	4 miles	heavy traffic
hotel keen	rest stop	578 arbol dr	3 miles	no traffic
safeway	grocery store	452 arcadia pl	4 miles	no traffic
midtown shopping center	shopping center	338 alester ave	3 miles	no traffic
round table	pizza restaurant	113 anton ct	4 miles	heavy traffic
mandarin roots	chinese restaurant	271 springer street	3 miles	moderate traffic

Request: where is the closest grocery_store?
Response: we are 4 miles away from whole_foods and from safeway: which one do you prefer?

Figure 7: An Example of KVRET.

4K steps for fine-tuning WebQSP, MTOP and KVRET, respectively. For adapter-tuning, we double the epoch number, which is 4K, 8K, and 8K steps. For the three tasks, we set the batch-size to 128, learning-rate to 2e-5, max-input-length to 1024, max-generation-length to 128, beam-size to 4, and evaluate every 2K steps for checkpoint selection.

For Sudoku, we train a 12-layer encoder-decoder transformer from scratch, with d-model=512, ffn-dim=2048, num-heads=8. We set max castep $T = 5$ and max epoch $E = 10$K steps. We set the batch-size to 1024, learning-rate to 2e-5, beam-size to 2, and evaluate every 2K steps for checkpoint selection.

C.2 Resource Consumption

Table 8: CASR Variations and Their Number of Parameters.

A, E and D denote #parameters of adapter, encoder and decoder.

	Tuning	Init.	Arch.	#Parameters
Ⓐ	Fine-tune	Restart	SepEnc	$T*(2E+D)$
Ⓑ	Fine-tune	Continue	SepEnc	$T*(2E+D)$
Ⓒ	Fine-tune	Restart	UniEnc	$T*(E+D)$
Ⓓ	Fine-tune	Continue	UniEnc	$T*(E+D)$
Ⓔ	Adapter	Restart	SepEnc	$T*A+(E+D)$

Table 9: Training and Inference Time Consumption (take WebQSP as an example)

	Method	Training(h)	Inference(min)
Ⓧa	INAT (Lee et al., 2018)	14	2
Ⓧb	Levenshtein (Gu et al., 2019)	12	7
Ⓧc	CASR Ⓑ / Dec	8	23
Ⓧd	Bidirectional (Zhang et al., 2018)	6	16
Ⓧe	Progressive (Tan et al., 2021)	9	16
Ⓑ0	Finetune	3	8
Ⓑ2	CASR	10	25
Ⓛ2	CASR-L	28	141

Table 8 shows the total parameters of each controlled experiment to help the audience understand our methods. In row Ⓔ, the total parameters is T*A+(E+D) rather than T*A+(2E+D), because $Encoder_1$ and $Encoder_2$ are initialized with the same PLM and frozen all the time.

Table 9 shows the training and inference time consumption of CASR and baselines. Note that CASR-L runs slowly because we leverage deepspeed[4] to avoid OOM.

We train on 4 Tesla V100 GPUs. It takes 10, 22, and 19 hours to fine-tune WebQSP, MTOP and KVRET, respectively, and 12, 25, and 21 hours to adapter-tune them. Adapter-tuning is intrinsically hard so it takes more training steps to tune fewer parameters comparing to fine-tuning. For Sudoku, it takes 32 hours to train from scratch.

D Detailed Results of Empirical Studies

Table 10: Performance of last-step prediction $\hat{Y}^1$ when castep t=2, produced by the setting of row Ⓑ in Table 8, grouped by $D_{\hat{Y}^1}$, the density to $\hat{Y}^1$.

Task	WebQSP	MTOP		KVRET		
Metric	F1	Acc	Match	Bleu	Micro F1	AVG
Sparse	51.88	78.52	72.72	16.70	64.17	56.80
Middle	85.84	87.53	83.17	20.18	75.09	70.36
Dense	85.78	89.08	87.63	20.26	85.80	73.71

Table 11: The average density to input X and previous prediction $\hat{Y}^{t-1}$ of the setting of row Ⓑ in Table 8. Note that, when castep t=0, the density to the previous prediction is 0, thus not listed.

t	Task	WebQSP	MTOP	KVRET	AVG
1	Input	0.10%	0.11%	0.40%	0.20%
	Previous Prediction	0.34%	0.33%	0.84%	0.50%
2	Input	0.09%↓	0.10%↓	0.39%↓	0.19%↓
	Previous Prediction	0.61%↑	0.43%↑	1.01%↑	0.68%↑

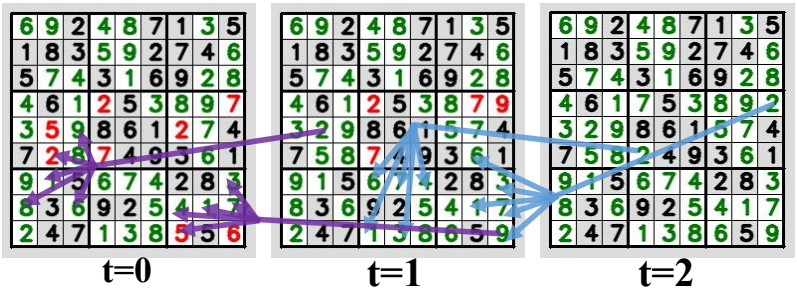


Figure 8: The Dependencies between Casteps Discovered by CASR. Shown as purple (t=1 attends to t=0) and blue (t=2 attends to t=1) arrows.

D.1 An Example to Compute Difficulty of Blanks

In the game table Figure 2, for the top-left blank (the ground truth is 6 in green), r=5 (the ground truths of the row blanks are 6, 9, 4, 8, and 3 respectively), c=6 (the ground truths of the column blanks are 6, 4, 3, 9, 8, and 2), and h=4 (the ground truths of the house blanks are 6, 9, 7, and 4). Then, the difficulty to fill the top-left blank is (5-1)*(6-1)*(4-1)=60.

[4] https://github.com/microsoft/DeepSpeed

Task&Input X	CASR Predictions $\hat{Y}^t$			Ground Truth Y
	t=0	t=1	t=2	
WebQSP **input:task: webqsp ; what school did sir ernest rutherford go to? ; structured knowledge: Ernest Rutherford: m.02m7r | m.068kst common.topic.notable_types m.01m9 | (omitted to save space) | m.018y0w base.act**	**(AND (JOIN common.topic.notable_types m.01nf) (JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r)))**	**(JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r))**	**(JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r))**	**(JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r))**
MTOP **input:task: mtop ; does this recipe have dairy ; structured knowledge: IN:GET: MESSAGE, WEATHER,... | (omitted to save space) | IN:FOLLOW: MUSIC**	**[IN:IS_TRUE_RECIPES [SL:RECIPES_INCLUDED_INGREDIENT dairy]]**	**[IN:IS_TRUE_RECIPES [SL:METHOD_RECIPES recipe] [SL:RECIPES_INCLUDED_INGREDIENT dairy]]**	**[IN:IS_TRUE_RECIPES [SL:METHOD_RECIPES recipe] [SL:RECIPES_INCLUDED_INGREDIENT dairy]]**	**[IN:IS_TRUE_RECIPES [SL:METHOD_RECIPES recipe] [SL:RECIPES_INCLUDED_INGREDIENT dairy]]**
KVRET **input:task: kvret ; where is the closest grocery_store ; structured knowledge: col : poi | poi_type | address | distance | traffic_info row 1 : pizza my heart | pizza restaurant | (omitted to save space) | 3 miles | moderate traffic ; context:**	**whole_foods is 4 miles away at 819_alma_st.**	**the closest grocery_store is whole_foods at 819_alma_st.**	**the closest grocery_store is whole_foods which is 4 miles away at 819_alma_st.**	**we are 4 miles away from whole_foods and from safeway: which one do you prefer?**
Sudoku	**t=0**	**t=1**	**t=2,3,4**	

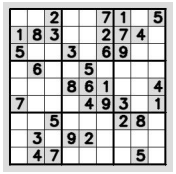
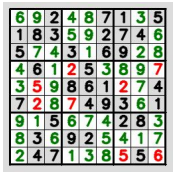
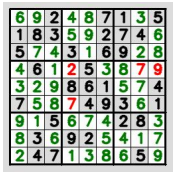
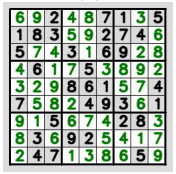
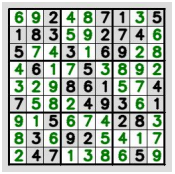






Figure 9: Case study for CASR (row Ⓑ) on WebQSP, MTOP, KVRET, and Sudoku at different casteps. The first column shows how the input is serialized from Figure 5, Figure 6, and Figure 7. For WebQSP, MTOP, and KVRET, keywords are highlighted with green color, and red color denotes wrong predictions. For Sudoku, the white fields denote the blanks to be filled, the red numbers in them denote that the predictions are wrong, and the green ones denote the correct predictions. It's a coincidence that the Sudoku prediction for this sample remains the same for t=2,3,4, so we draw them together.

Method	WebQSP	MTOP	KVRET
INAT	(JOIN (R education.education.institution) (AND (JOIN education.education.degree).02 (JOIN educationeducation.JO3.)www.wm.educationma02date)	and_IN: the_]ACT [ ] the for:]]IN for	error iserror__ theerror__ at is is_error___ is___.. would would is is_error is is
Levenshtein	(JOIN (R education.education.institution) (JOIN (R people.person.education) m.03xsv3))	_	what is for you.
CASR /Dec	(JOIN (R education.education.institution) (JOIN (R people.person.education) m.02mjmr))	[IN:GET_RECIPES [SL:RECIPES_DISH chicken]]	the nearest grocery_store is whole_foods at 819_alma_st. would you like directions there?
Bidirectional	(JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r))	[IN:GET_RECIPES [SL:RECIPES_INCLUDED_INGREDIENT dairy]]	safeway is 4 miles away.
Progressive	(AND (JOIN common.topic.notable_types m.01nf) (JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r)))	[IN:IS_TRUE_RECIPES [SL:RECIPES_INCLUDED_INGREDIENT dairy]]	the closest grocery_store is safeway.
CASR-L	(JOIN (R education.education.institution) (JOIN (R people.person.education) m.02m7r))	[IN:IS_TRUE_RECIPES [SL:RECIPES_INCLUDED_INGREDIENT dairy]]	the closest grocery_store is whole_foods at 819 alma st.

Figure 10: Case study for other experiments (baselines and CASR-L).

Algorithm 1 CASR Inference Process.

Input: max castep T; input X (from test set);
well-trained CASR models M^t $(0 \leq t < T)$
Output: the best prediction $\hat{Y}$ for X

for t in $0, 1, ..., T-1$ **do**
  /* $argmax$ by beam searching */
  if $t = 0$ **then**
    $\hat{Y}^t \leftarrow argmax_Y \ P(Y|X, M^t)$
  else
    $\hat{Y}^t \leftarrow argmax_Y \ P(Y|X, \hat{Y}^{t-1}, M^t)$
  end if
end for
return $\hat{Y}$

Algorithm 2 CASR Training Process (§3.1).

Input: max castep T; max epoch E;
all input X and ground truth Y
in train (S_t) and dev (S_d) sets
Output: CASR models M^t $(0 \leq t < T)$;

for t in $0, 1, ..., T-1$ **do**
  $\theta \leftarrow$ parameter initialization (§3.4)
  for e in $0, 1, ..., E-1$ **do**
    /* $argmax$ by updating θ using train set */
    if $t = 0$ **then**
      $\theta \leftarrow argmax_\theta \sum_{(X,Y) \in S_t} P(Y|X, \theta)$
    else
      $\theta \leftarrow argmax_\theta \sum_{(X,Y) \in S_t} P(Y|X, \hat{Y}^{t-1}, \theta)$
    end if
    $M_e^t \leftarrow \theta$
  end for
  $M^t \leftarrow$ checkpoint selection from M_e^t $(0 \leq e < E)$ on S_d
  /* predict with beam searching $\forall X \in S_t \bigcup S_d$ using M^t */
  if $t = 0$ **then**
    $\hat{Y}^t \leftarrow argmax_Y \ P(Y|X, M^t)$
  else
    $\hat{Y}^t \leftarrow argmax_Y \ P(Y|X, \hat{Y}^{t-1}, M^t)$
  end if
end for
return M^t $(0 \leq t < T)$