

# **Long-Term Compressive Memory Transformer for Encoding and Verbalizing Robot Experiences**

Master's Thesis  
of

**Leonard Bärmann**

KIT Department of Informatics  
Institute for Anthropomatics and Robotics (IAR)  
Interactive Systems Lab (ISL)

**Reviewers:** Prof. Dr. Waibel  
Prof. Dr. Asfour

**Advisors:** M. Sc. Stefan Constantin  
M. Sc. Fabian Peller-Konrad

**Duration:** November 12, 2020 – May 10, 2021



**Erklärung:**

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 10. Mai 2021

Leonard Bärmann



**Abstract:**

Verbalization of past events and experiences is a key skill for autonomous robots as it increases user trust and system transparency. Since this requires a robot to store, filter and recall information about its former observations and actions, it needs to be equipped with an Episodic Memory (EM). Previous work introduced the task of Episodic Memory Verbalization (EMV) and presented a neural network architecture for approaching this problem. Based on an analysis of the current limitations, the present thesis proposes a new model for EM, called the Long-Term Compressive Memory Transformer, providing both unlimited context length as well as an EM constrained to a constant size. Furthermore, a notably larger and more diverse EMV dataset is collected, recording a multimodal data stream from thousands of simulated robot executions. Using the dataset of Bärmann et al. (2021) in addition to the newly collected one, the introduced model is trained and evaluated, thereby validating the feasibility of the proposed architecture with the results surpassing both a text-only baseline as well as the previous work. Moreover, a variant of the model featuring multiple compression levels is developed and empirically compared to the other approaches. Eventually, ablation studies on certain aspects of the chosen approach in combination with detailed analyses of all experimental results reveal that, although significant improvements were made, there are still unsolved problems in the EMV task. To facilitate future work in this field, the presented dataset, models and code will be released at <https://gitlab.com/lbaermann>.

### **Kurzzusammenfassung:**

Verbalisierung vergangener Ereignisse und Erfahrungen ist eine Schlüsselfähigkeit für autonome Roboter und fördert das Vertrauen des Benutzers in die Technologie. Um zu ermöglichen, dass ein Roboter Informationen über seine früheren Beobachtungen und Handlungen speichert, filtert und abrufen kann, muss er mit einem episodischen Gedächtnis (Episodic Memory, EM) ausgestattet werden. In vorherigen Arbeiten wurde die Aufgabe des Verbalisierens von Robotererfahrungen (Episodic Memory Verbalization, EMV) eingeführt und eine neuronale Netzwerkarchitektur zur Lösung dieses Problems vorgestellt. Basierend auf einer Analyse der Grenzen des bestehenden Systems schlägt die vorliegende Arbeit ein neues Modell namens Long-Term Compressive Memory Transformer zur Erstellung von EM vor. Dieses bietet sowohl eine unbegrenzte Kontextlänge als auch ein auf eine konstante Größe beschränktes EM. Zusätzlich wird ein deutlich größerer und vielfältigerer EMV-Datensatz, der einen multimodalen Datenstrom aus Tausenden von simulierten Roboteraktionen beinhaltet, präsentiert. Unter Verwendung des Datensatzes von Bärman et al. (2021) sowie des neu gesammelten Datensatzes wird das vorgestellte Modell trainiert und evaluiert, wobei die Ergebnisse sowohl eine text-only-Baseline als auch die vorherige Arbeit übertreffen. Darüber hinaus wird eine Variante des Modells mit mehreren Kompressionsstufen entwickelt und empirisch mit den anderen Ansätzen verglichen. Schließlich zeigen Ablationsstudien zu bestimmten Aspekten des gewählten Ansatzes in Kombination mit detaillierten Analysen aller experimentellen Ergebnisse, dass es mit Blick auf die EMV-Aufgabe trotz signifikanter Verbesserungen noch ungelöste Probleme gibt. Um zukünftige Arbeiten auf diesem Gebiet zu erleichtern, werden der vorgestellte Datensatz, die Modelle und der Code unter <https://gitlab.com/lbaermann> veröffentlicht werden.

# Contents

<b>1. Introduction</b>	<b>2</b>
<b>2. Basics</b>	<b>4</b>
2.1. Gated RNN Units . . . . .	4
2.2. Encoder-Decoder Network . . . . .	5
2.3. Transformer . . . . .	6
<b>3. Related Work</b>	<b>9</b>
3.1. Verbalization of Robot Experience . . . . .	9
3.2. Episodic Memory . . . . .	9
3.2.1. Learning from Provided Structure . . . . .	10
3.2.2. Self-Supervised Structure Learning . . . . .	11
3.2.3. Long-Range Transformer . . . . .	12
<b>4. Models and Methods</b>	<b>15</b>
4.1. EMV Architecture . . . . .	15
4.2. Episode Input Data . . . . .	16
4.3. Episodic Memory Encoder . . . . .	17
4.3.1. Baseline Models . . . . .	18
4.3.2. Compressive Transformer Models . . . . .	18
4.4. Speech Encoder and Decoder . . . . .	23
4.5. Supplementary Decoders & Loss Functions . . . . .	23
<b>5. Evaluation</b>	<b>26</b>
5.1. Datasets . . . . .	26
5.1.1. Simulated Episode Recordings . . . . .	26
5.1.2. Grammar-Generated Q&A . . . . .	27
5.1.3. Generalization Test Sets . . . . .	29
5.2. Evaluation Metrics . . . . .	30
5.3. Training Procedure . . . . .	31
5.4. Results . . . . .	33
<b>6. Conclusion and Future Work</b>	<b>44</b>
<b>Appendices</b>	<b>47</b>
<b>A. Additional Analyses</b>	<b>48</b>
<b>B. Learning Curves</b>	<b>50</b>
<b>C. Examples</b>	<b>51</b>
<b>D. Data Collection Site</b>	<b>53</b>

# 1. Introduction

For humans, narrating past events and experiences to other people is a key part of social interaction. Thus, to become accepted by potential users, a humanoid robot must be able to verbalize its observations and actions in a similarly natural way. For instance, consider a person coming home and asking her/his autonomous kitchen robot “What did you do when I was at work?”. By responding with “I cleaned up the table and made the dishes. You’re welcome!”, the robot makes its actions transparent and thereby increases the user’s trust in it. Furthermore, robot experience verbalization is crucial in case of failures. If the robot fails to execute some action (or worse, causes damage, e. g. by dropping a plate), it must be able to tell the human that something went wrong, and ideally also provide details about how and why it happened.

To realize such functionality, a robot must be equipped with a component resembling the human episodic memory, storing and processing information about past events and their spatio-temporal relations (Tulving, 1972). Particularly, the robotic system needs to process, filter and store data from a highly multi-modal, continuous stream of experiences, including camera images, kinematic data, symbolic execution information, and more. Verbalization then is the process of understanding a natural language query provided by a user, retrieving the relevant information from the robot’s episodic memory, and synthesizing a natural language answer to fulfill the users request.

While previous work (Rosenthal et al., 2016; Zhu et al., 2017) on episodic memory verbalization (EMV) used rule-based procedures for recording episodic memories as well as generating natural language representations thereof, recently, Bärmann et al. (2021) proposed to use a data-driven method, tackling the EMV task as an end-to-end learning problem. For this purpose, a basic architecture for approaching the EMV task is introduced: Episodic memory (EM) is created by feeding high-dimensional experience vectors provided by the robotic system into an LSTM, and storing only the lower-dimensional sequence of output hidden states. Additionally, a natural language query encoder, which is a Transformer encoder (Vaswani et al., 2017), receives the users question and produces a latent representation. Eventually, the natural language decoder attends to both the query encoding as well as the sequence of latent episodic memory entries to produce the robot’s response, using an auto-regressive Transformer decoder.

While this model already shows some promising results, it has several drawbacks and limitations:

1. As the evaluation and ablation study of Bärmann et al. (2021) indicates, the model has severe difficulties to generalize well to long histories, i. e. the episodic memory is not good at “memorizing” longer sequences of events.
2. Although each key-frame in the stream of robot experiences is compressed to a latent, comparatively low-dimensional EM vector, the content of EM still grows linearly with the number  $N$  of experiences the robot makes. This is infeasible for actually deploying this system to a robot, since the storage consumption of the EM grows with  $O(N)$  and, even more importantly, the runtime of Transformer decoder self-attention also scales with  $O(N)$  at *query* time.
3. Evaluation on a small, hand-annotated dataset additionally revealed that the model is mostly unable to generalize to natural language variations in the user’s query.

Thus, the present thesis aims at enhancing the architecture of Bärmann et al. (2021), tackling some of its problems and thereby improving results on the EMV task. Specifically, addressing the above mentioned issues, the following goals are set:

1. To improve generalization to longer episodic histories, the EM encoder needs to use a more sophisticated architecture than a plain LSTM. However, the requirement to keep only a constant amount of raw experiences needs to be kept, in order to be able to handle a potentially infinite number of

---

inputs<sup>1</sup>. Therefore, neither a plain Temporal Convolutional Network (Bai et al., 2018) nor a vanilla Transformer encoder can be used, and some kind of recurrence needs to be integrated.

2. When constructing a new EM encoder model as mentioned above, it needs to be constrained to produce an episodic memory representation with a constant space requirement. Thereby, also the runtime cost at query time automatically reduces to  $O(1)$ , which is highly desirable.

While addressing the third of the above limitations, namely the poor generalization to variations of natural language queries, is equally desirable, it stays out of the scope of this work. Particularly, this would require a more diverse dataset of human-annotated natural language queries and answers to be collected, which is a cost- and work-intensive task. Thus, the focus in this thesis is on improving the construction of episodic memory.

With these goals set, two major contributions are made: First, a far larger and more realistic dataset of simulated robot experiences is collected (compared to the one in Bärmann et al., 2021), featuring more diversity in involved actions and objects, as well as new grammar-generated dialogs specifically facilitating the training of long-term memory. Second, the base model of the previous work is extended with a new EM encoder, honoring the  $O(1)$  space constraint as well as allowing for a theoretically unbounded information flow path in time. For this purpose, the Long-Term Compressive Memory Transformer is developed. Evaluation on both the existing dataset of Bärmann et al. (2021) and the new dataset reveals the feasibility of this architecture, with the results surpassing both a text-only baseline as well as the previous work. Furthermore, a variant of the model featuring multiple compression levels is proposed, and empirically compared to the other approaches.

The remainder of this thesis is structured as follows: Chapter 2 begins with presenting some basics on recurrent networks and the Transformer architecture. Then, related work with respect to both the EMV task as well as the construction of long-term EM is reviewed in chapter 3. After that, chapter 4 presents the EMV architecture in general and the compressive-transformer-based implementation of EM in particular. Chapter 5 first introduces datasets, evaluation methodology and training procedure, and then discusses the experimental results in section 5.4. Finally, chapter 6 draws a conclusion and sheds some light on possible directions of future work.

---

<sup>1</sup>When deploying an EMV system to a real robot, the stream of episodic inputs is unlimited. Thus, the system must not depend on all inputs to be present before doing any processing (because there is no end of EM inputs) and, in order to avoid flooding the disk, it must not store all past inputs.

## 2. Basics

The following chapter describes some of the basic techniques concerning the use of artificial neural networks in this work. It assumes the reader is already familiar with the concept of an Artificial Neural Network (ANN) or Multi-Layer Perceptron (MLP), as well as its training strategy, namely the backpropagation algorithm. Furthermore, some basic understanding of Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) should be present. More details on these topics can e.g. be found at Bishop (2006) and Olah (2015).

### 2.1. Gated RNN Units

While basic RNNs allow to work with sequential data, they often suffer from the vanishing gradient problem when working with long input sequences (Hochreiter et al., 1997). The main issue is that, due to the hidden state being overwritten at each time step, the gradient vanishes (multiplies near to zero) when applying too many time steps, and therefore, the net is unable to be trained to “remembering” relevant facts over a longer period of time. For solving this problem, a common approach is to introduce so called “gates”, controlling the insertion and deletion of memory state independently of the output value, so that relevant information can be remembered over long time distances. Most major successes with RNNs were actually achieved using variants of gated units (Chung, Gulcehre, et al., 2014), and two of them will be introduced in the following.

**LSTM** The most popular approach to avoid the vanishing gradient problem is the Long Short-Term Memory (LSTM) cell introduced by Hochreiter et al. (1997). It adds gates, i.e. multiplicative weights, to the hidden state  $h_t$  and introduces an additional cell state  $c_t$  (also referred to as “memory”). At time step  $t$ , first, a new cell state candidate  $\tilde{c}_t$  is calculated using

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1}) \quad (2.1)$$

Updating the cell state is done by “forgetting” the old cell state data with factors  $f_t$  and “inserting” the proposed data with factors  $i_t$  as

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \quad (2.2)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \quad (2.3)$$

$$c_t = f_t \odot c_t + i_t \odot \tilde{c}_t \quad (2.4)$$

with  $\odot$  representing pointwise multiplication. Eventually, the hidden state to output is determined by the cell state and an “output gate” multiplier  $o_t$  specifying the amount of memory information to let through:

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \quad (2.5)$$

$$h_t = o_t * \tanh(c_t) \quad (2.6)$$

All matrices  $W_*$  and  $U_*$  are trainable parameters. Figure 2.1 summarizes the calculations done inside an LSTM cell.

**GRU** The Gated Recurrent Unit (GRU) introduced by Cho et al. (2014) is an RNN variant requiring less parameters than LSTMs. Nevertheless, it is shown to have a competitive performance on a variety of tasks (Chung, Gulcehre, et al., 2014), and thus is commonly used in practice. The basic idea of gating

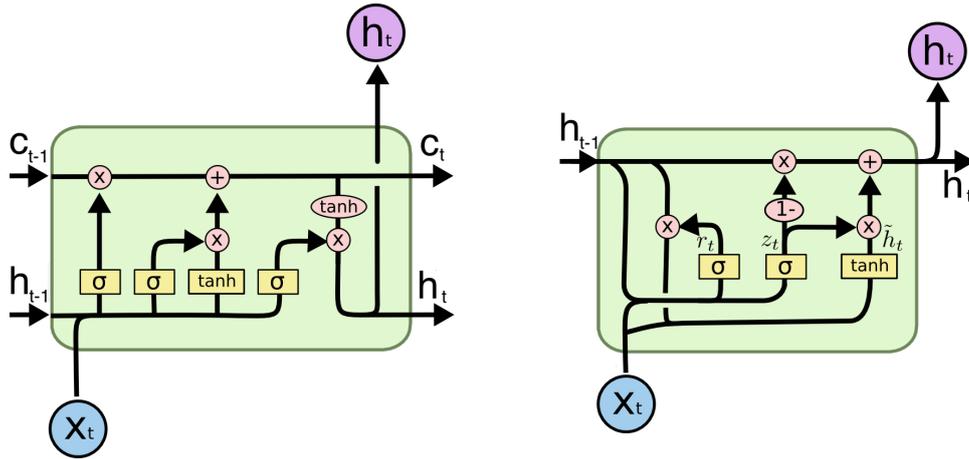


Figure 2.1.: Illustration of the calculations done inside a single LSTM (left) and GRU (right) cell.  $\sigma$  and  $\tanh$  boxes represent a neural network layer with the given activation function, red circles pointwise operations. Merging vector paths are concatenations. Graphics modified from Olah (2015)

is similar to LSTM, but the GRU does not require an additional memory cell and uses only two instead of four gates. In particular, at time step  $t$ , first, the reset gate  $r_t$  and update gate  $z_t$  are calculated as

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \quad (2.7)$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \quad (2.8)$$

The reset gate is now used to modulate a cell update vector  $\tilde{h}_t$ , which is

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1})) \quad (2.9)$$

Finally, the influence of  $\tilde{h}_t$  on the new hidden state is controlled by the update gate:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \quad (2.10)$$

## 2.2. Encoder-Decoder Network

Sutskever et al. (2014) introduced encoder-decoder networks to handle general sequence-to-sequence tasks. In the original formulation, an encoder network first transforms the input sequence  $\mathbf{x} = (x_1, \dots, x_N)$  to a fixed-length representation (context) vector  $c$ . Then, the decoder uses this context vector to generate outputs  $\mathbf{y} = (y_1, \dots, y_{M-1}, y_M)$ , with the final output  $y_M$  being a special token to represent the sequence end.

**Function** More formally, with  $rnn_{\text{enc}}$  being the recurrent function of the encoder and  $h_0 = s_0 = 0$ , the context vector  $c$  is defined as

$$\begin{aligned} h_t &= rnn_{\text{enc}}(x_t, h_{t-1}) \\ c &= h_N \end{aligned} \quad (2.11)$$

with  $t$  running from 1 to  $N$ . Decoding with  $rnn_{\text{dec}}$  uses the context vector to produce its outputs  $s_t$ :

$$s_t = rnn_{\text{dec}}(c, s_{t-1}) \quad (2.12)$$

Eventually, an MLP layer  $g$  called “generator” maps the decoder hidden states to the actual output token probabilities:

$$\mathbb{P}(y_i|t) = g(s_t)[i] \quad (2.13)$$

$$y_t = \operatorname{argmax}_{y_i \in V_{\text{out}}} (\mathbb{P}(y_i|t)) \quad (2.14)$$

where  $V_{\text{out}} = \{y_1, \dots, y_O\}$  is the output vocabulary set and  $g$  usually uses the *softmax* function to output a probability distribution vector of dimension  $O$ .

**Training and Inference** An encoder-decoder model is usually trained in “teacher-forcing” mode, i. e. instead of using the previous output of the decoder as  $s_{t-1}$ , the target output is used. In contrast, when testing the model, the probability  $\mathbb{P}(y_t|y_{t-1}, \dots, y_1, \mathbf{x})$  needs to be maximized. As a straightforward strategy, it is possible to use a greedy algorithm as shown by Eq. (2.14), picking the  $y_t$  with the highest probability at each time step. A more sophisticated implementation will use an algorithm like beam search (Lowerre, 1976) to search the path  $y_1, \dots, y_t$  with the highest probability while avoiding an exponential explosion of computation time.

**Attention** Bahdanau et al. (2015) pushed the field of machine translation forward by extending the encoder-decoder framework with a so-called “attention” mechanism. Informally, it enables the decoder to choose on which part of the input to focus while generating the next output word (Olah and Carter, 2016). For this purpose, instead of only keeping a single vector  $c$ , the whole sequence of encoder hidden states  $s_i$  is kept, and the context vector is computed dynamically at every time step, thus now writing it as  $c_t$  and computing to

$$c_t = \sum_{i=1}^N \alpha_{i,t} h_i \quad (2.15)$$

where the  $\alpha_{i,t}$  can be interpreted as a measure of how much the output at time  $t$  aligns with the input at time  $i$ . For calculating these alignment factors, a (learned) function  $a$  measuring the relevance score is required. A softmax is applied to normalize the attention weights to sum up to 1:

$$e_{i,t} = a(s_{t-1}, h_i)$$

$$\alpha_{i,t} = \frac{\exp(e_{i,t})}{\sum_{k=1}^N \exp(e_{k,t})}$$

As the attention  $a$  is differentiable, it can be trained along with the other parts of the model. This mechanism led to major improvements especially for lengthy input sequences (Bahdanau et al., 2015), and is the basis of the Transformer model, which is presented in the following section.

## 2.3. Transformer

The Transformer model introduced by Vaswani et al. (2017) is using the encoder-decoder architecture designed for handling sequence-to-sequence tasks, but works solely with attention, abandoning RNNs completely. For this purpose, self-attention is used, i. e. each position in the sequence attends to itself as well as each other position. Every layer of the Transformer encoder performs this self-attention over the previous layer’s hidden states, followed by a position-wise feedforward network for transforming the input data. The decoder works similarly, but its self-attention is constrained to respect the auto-regressive setting, and it is additionally allowed to attend to all encoder output states (so called encoder-decoder-attention).

**Attention** The following paragraphs will introduce a variant of the plain attention mechanism introduced by Bahdanau et al. (2015), as it is used in the Transformer (Vaswani et al., 2017). It involves a

query vector  $q$ , which represents the current context data of the attention mechanism, as well as a sequence of key vectors  $k_i$  and corresponding values  $v_i$ . Each key is compared to the query, yielding a relevance or compatibility score  $\alpha_i$ , which is then used to weigh each corresponding value in the linear combination which is the attention's output. Generally speaking:

$$\text{Attention}(q, K, V) = \sum_i \text{RelevanceScore}(q, k_i) \cdot v_i \quad (2.16)$$

where  $K, V$  are matrices of the  $k_i, v_i$ , respectively.

**Scaled Dot-Product Attention** In the context of self-attention, the attention mechanism as described above is used with each sequence item as the query once, and the complete sequence both as keys and values. This means, in practice, the signature  $\text{Attention}(q, K, V)$  can be replaced with  $\text{Attention}(Q, K, V)$ , where  $Q$  is also a matrix of all different query vectors (i. e. in the context of self-attention,  $Q = K = V$ ).

To further specify the attention mechanism, a specific realization of the `RelevanceScore` function needs to be chosen. When Bahdanau et al. (2015) introduced the attention mechanism for RNN-based encoder-decoder networks, the compatibility function is a learned single-layer feed-forward network. However, simply using the dot-product between query and key vectors as a measure of relevance (in combination with a softmax layer) yields a much faster implementation, and has a comparable representational power if the queries, keys and values can be transformed by other learned components of the model. Thus, Vaswani et al. (2017) use the following attention mechanism, called ‘‘Scaled Dot-Product Attention’’:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.17)$$

where  $d_k$  is the dimension of the key vectors. The scaling factor  $1/\sqrt{d_k}$  is introduced to keep the softmax away from regions with extremely small gradients. Without the scaling, this could occur for large values of  $d_k$ , since the dot product would lead to larger numbers with increasing dimension (since more values are summed together).

The use of self-attention in the decoder additionally requires masking out illegal connections which would otherwise enable information flow in the wrong direction of time. In particular, when generating the token  $i$ , the model should not be allowed to attend to all positions  $j \geq i$  (i. e. , the model is ‘‘auto-regressive’’). This is implemented by setting the values in the softmax of Eq. (2.17) to  $-\infty$  for the illegal connections. A depiction of the scaled dot-product attention mechanism can be found on the left of Fig. 2.2.

**Multiple Attention Heads** As mentioned above, when using the plain dot product as a measure of relevance, there needs to be some other way to transform the keys, queries and values. For this purpose, Vaswani et al. (2017) use learned transformation matrices on all the inputs of the attention function. To enable the model to attend to different parts of the input data simultaneously, they additionally use multiple attention ‘‘heads’’, where each head has its own transformation parameters. This means:

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \dots, \text{head}_H]W_O \quad (2.18)$$

$$\text{with } \text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i) \quad (2.19)$$

with learned matrices  $W_Q^i, W_K^i, W_V^i, W_O$ . This is depicted in the middle of Fig. 2.2.

**Complete Transformer Layer** To complete the transformer layer, next to the multi-head self-attention module described above, a fully-connected feed-forward network with a single hidden layer is applied to each position:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \quad (2.20)$$

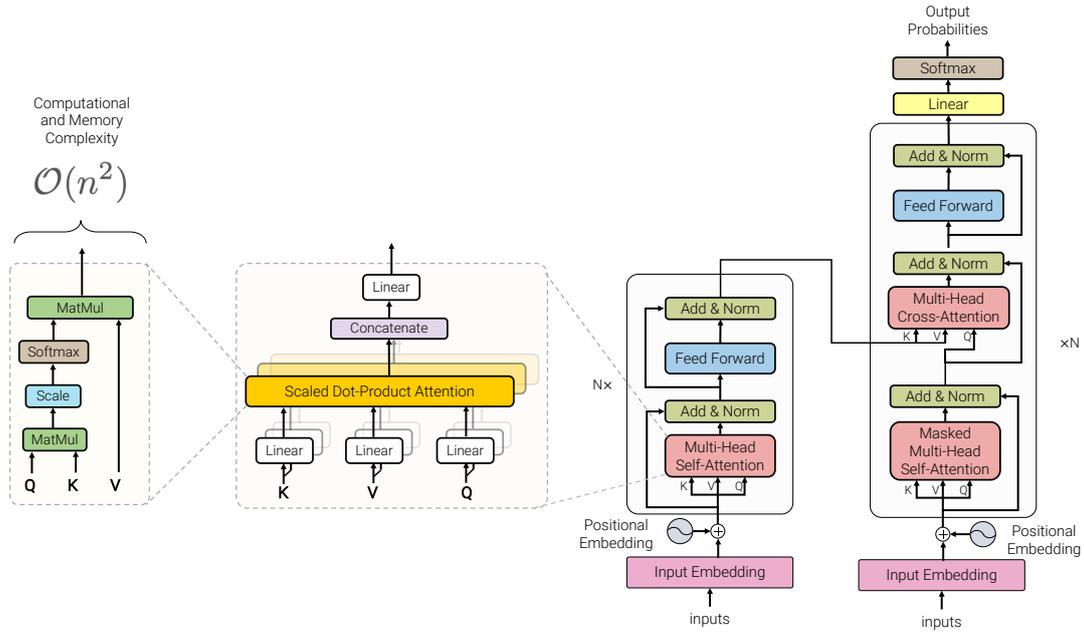


Figure 2.2.: Left: A single instance of the scaled dot-product attention mechanism. Middle: The Multi-Head Attention Module, as used in the transformer. Right: The complete transformer model. The graphic is taken from Tay et al. (2020)

where the weights  $W_1, W_2, b_1, b_2$  are shared between positions (but different in each layer).

Furthermore, each transformer sub-layer (self-attention, feed-forward, and encoder-decoder attention in the case of the decoder) uses residual connections (He et al., 2016), which were shown to ease learning of deep networks in the domain of image recognition. The result of each sub-layer is additionally normalized using layer normalization (Ba et al., 2016). The complete architecture can be seen on the right of Fig. 2.2. Formalizing the above, an encoder layer can be written as

$$\text{TransformerEncLayer}(x) = \text{norm}(x + \text{FFN}(\text{norm}(x + \text{MultiHead}(x, x, x)))) \quad (2.21)$$

and a decoder layer as

$$\text{TransformerDecLayer}(x, e) = \text{norm}(x + \text{FFN}(\text{norm}(x + \text{MultiHead}[\text{norm}(x + \text{MultiHead}(x, x, x)), e, e])))$$

where  $x$  is the input as delivered by the previous layer (or the embedded input sequence to the model, for the first layer) and  $e$  is the output of the last encoder layer.

**Positional Encoding** Since the self-attention mechanism applies the same network to each position in the input, it by itself has no notion of locality or the order of the sequence. To inject this information, a positional encoding is used. Vaswani et al. (2017) utilize sinusoidal vectors added to the input token embeddings to encode the absolute position of the token in the sequence. However, there are plenty of approaches in the literature to supply positional information: For example, instead of using sinusoidal vectors, one can treat the position encoding as learned parameters. However, this makes it more difficult to generalize to sequences longer than those seen during training. Another approach is to encode the *relative* distance between two tokens instead of the *absolute* position in the sequence. This is used – with implementations differing in the details – e. g. by Dai et al. (2019), Ainslie et al. (2020), and Q. Wu et al. (2020).

---

## 3. Related Work

This work deals with constructing an episodic memory (EM) for capturing long-term dependencies, constrained to a constant space requirement, in the context of the episodic memory verbalization (EMV) task. Hence, there are two major tracks of related work, which will be presented in the following: First, section 3.1 will deal with the EMV task and studies related to it. After that, section 3.2 will focus on the construction of EM as a particularly interesting sub-task of EMV.

### 3.1. Verbalization of Robot Experience

The term *verbalization* of an autonomous robot’s experience was introduced by Rosenthal et al. (2016), presenting a rule-based algorithm for narrating the routes taken by their robot. With the follow-up work of Perera et al. (2016), this is extended by a component for question understanding. However, the module is limited to mapping a question to a set of verbalization parameters like *abstraction* or *specificity*, thus not enabling open-ended questions about the robot’s past. Orthogonal to that, Zhu et al. (2017) also build on the initial work of Rosenthal et al. (2016) and transfer it to a humanoid robot interacting with its environment. Thus, instead of verbalizing navigation routes, they enable the ARMAR-III robot (Asfour et al., 2006) to talk about its last actions in a kitchen environment.

Inspired by these works, Bärmann et al. (2021) recently introduced a deep-learning-based system for verbalizing a robot’s experience given open-ended questions about its past. As that work is the direct predecessor of the present thesis, the model architecture as well as the mechanisms for gathering large sets of training data will be reused here. Further details about the work of Bärmann et al. (2021) will be revisited at the appropriate places of this thesis.

The preceding work also draws a connection of EMV to the field of Question Answering (QA), where the EM can be filled with content from different modalities: For instance, fact-based QA (Kumar et al., 2016), where a natural language (NL) question is given with a set of NL supporting facts, can be seen as a special case of EMV with the EM constructed from NL utterances only. Similarly, in task-oriented dialog, e. g. in a restaurant dialog setting (Wen et al., 2017), recent works like Eric et al. (2017), C.-S. Wu et al. (2019), and X. Chen et al. (2019) use neural architectures to access a symbolic knowledge base (which can be seen as a form of EM) with the purpose of answering a NL query. On the other end of the spectrum, Video QA builds an EM of the past and answers NL questions based on a sequence of images (Z. Yu et al., 2019; Zhao et al., 2020). However, as stated by Bärmann et al. (2021), all QA works differ to EMV with respect to the timing of both the episode data (an explicit time signal is missing) as well as the question. The latter is crucial, as it allows (V)QA works to use attention or similar mechanisms informed about the question directly on the episodic input data (e. g. the video or the knowledge base). In contrast, in the EMV setting, the episodic data needs to be encoded, compressed and stored independently of the question. Further, such processing must happen before even knowing if a question about this data will ever occur in the future at all.

### 3.2. Episodic Memory

Introduced by Tulving in the context of cognitive sciences, the term *episodic memory* refers to the part of mind that “receives and stores information about temporally dated episodes or events, and temporal-spatial relations among these events” (Tulving, 1972, p. 385f.). It is contrasted to *semantic memory*, which contains organized knowledge about words, rules, symbols and algorithms. Both memories interact dynamically, and in humans, the content of episodic memory is highly unstable, as even retrieval of

information from it may change its content.

In the meantime, plenty of researchers proposed systems for transferring the concept of EM to robotics. For instance, the  $\text{CRAM}_m$  model of Winkler et al. (2014) creates an EM of two components: On the one hand, symbolic information about executed actions, involved objects etc. are asserted to a predefined, information rich ontology (Beetz et al., 2018). This includes timestamps. On the other hand, low-level perceptual information in form of camera images and robot pose data are stored in a NoSQL database, which is then projected into the above ontology. Thereby, the system is able to answer formal questions (given as Prolog queries) about the robot’s past.

Other approaches construct an EM using deep neural networks. In the case of Rothfuss et al. (2018), EM is constructed from a sequence of images by using a convolutional LSTM network (Shi et al., 2015), simultaneously auto-encoding the last and predicting the next few frames. Similarly, video embeddings can be constructed in a self-supervised manner using temporal cycle consistency learning (Dwivedi et al., 2019). Moreover, Graves et al. (2016) present the Differentiable Neural Computer, which consists of a controlling neural network equipped with an external memory to remember structured inputs and perform algorithmic tasks on these episodic memories.

Alternatively to backpropagation-based learning, adaptive resonance theory (ART) neural models can be used to build an EM (Wang et al., 2012). Chang et al. (2017) show that this also enables an explicit representation of time in the model, and Park et al. (2018) apply a deep ART model to various tasks in robotics.

Finally, in the predecessor of this thesis, Bärman et al. (2021) propose to use several linear layers in combination with a two-layered LSTM to construct an EM from robot experiences in the context of the EMV task. However, this approach has two major limitations: First, the episodic memory grows linearly with the number of experiences fed into the model, and thus, the storage – as well as the computation cost at question time – is unbounded. Second, the experimental results presented by the authors revealed a severe decrease of correct answers in the EMV task with increasing number of experiences. Hence, with the goal of improving on these two points, several works which do not explicitly talk about EM – but present promising techniques which may be applied for constructing an enhanced version thereof – will be reviewed in the following.

### 3.2.1. Learning from Provided Structure

Constructing a deep-learning-based episodic memory from a temporal stream of multimodal robot experiences essentially boils down to creating a latent representation of a (potentially very) long sequence of inputs. In this light, the choice of Bärman et al. (2021) to use an LSTM for encoding EM is natural. However, a possible explanation of the observed problems in handling larger numbers of inputs might be a missing inductive bias towards hierarchical representations of the past. While a certain hierarchy is definitely present in robot experience data (e. g. movements – actions – sub-tasks – goals), a plain LSTM has no explicit mechanism to support such structure. An early proposal of hierarchical recurrent networks was made by El Hihi et al. (1995) (talking about plain RNNs, as LSTMs were not yet introduced at that time). Their basic idea is to use multiple recurrent connections with different delays, i. e. individual neurons operate on different time scales. While this induces some hierarchy to the processing of the input, it resorts to fixed hyperparameters specifying the time scales. However, if an external hierarchy signal is present in the data, it could also be used to dynamically trigger updates of neurons on a different level of the hierarchy. This might be applicable to the EMV task, as e. g. the end of a certain action is known to the robot’s planning system. With such a hierarchy present in the data, the use of Hierarchical Attention Networks as introduced by Yang et al. (2016) in the context of document classification is possible, too. Here, based on multiple levels of GRUs (one per hierarchy level), attention with respect to a learned query vector is performed on each level to provide the input to the next one. In the document classification setting, this means each sentence is processed by a word-level GRU, and the resulting sequence is summarized into a sentence embedding using word-level attention. Then, in turn, the sequence of sentence embeddings is processed by a sentence-level GRU, and sentence-level attention is applied to obtain a single vector summarizing the complete document. A similar strategy could be applied to the different abstraction levels in robot experience data, if the model is properly adjusted for

live usage on a robot (i. e. a setting where the length of the input sequence is not known in advance, and it is potentially infinite).

Although recurrent architectures are the most prominent approach of dealing with temporal sequences, Bai et al. (2018) suggest and empirically support that simple convolutional neural networks can compete and even outperform out-of-the-box recurrent models. In particular, they propose the Temporal Convolutional Network (TCN), which basically is a modern formulation of the Time-Delay Neural Network (Waibel et al., 1989; Waibel et al., 1987), supplementing it with modern tricks like dilated convolutions (F. Yu et al., 2016) and residual layers (He et al., 2016). While automatically providing an inductive bias towards hierarchical structures due to the convolution layers, the TCN cannot be used as is for the EMV task because it has a limited effective context length (which depends on the number of layers, size of convolution kernels, and dilation factors).

### 3.2.2. Self-Supervised Structure Learning

**Learning Segment Boundaries** While the architectures described above rely on either fixed hyperparameters or an explicit signal in the data to utilize hierarchical structures, the idea of automatically inferring such structure during learning of an overall task goal is intriguing, as it might both improve model performance as well as lead to a higher interpretability of its internal workings. With these goals, Chung, Ahn, et al. (2017) propose the Hierarchical Multiscale RNN (HM-LSTM), capable of learning a hierarchical structure on sequence data without providing explicit boundaries. This is achieved by introducing discrete, binary boundary variables as part of each LSTM cell, controlling whether the current time step should trigger an update of the subsequent layer. Boundary variables are computed similarly to the other LSTM gates, using a constant-slope activation function called hard sigmoid, and then binarized using a step function or Bernoulli sampling. Depending on the previous time step's and the preceding layer's boundary value, an LSTM cell either performs a COPY operation (i. e. nothing) in case no boundary was detected, an UPDATE (i. e. the usual LSTM cell computation) if the preceding layer found the end of a segment, or a FLUSH operation (resetting the LSTM's state) in case the current layers output was delivered to the upper layer in the previous time step. Although boundary variables are binarized, back-propagation is used for training the model by utilizing the straight-through estimator trick, i. e. by replacing the non-differentiable function in the model with a differentiable approximation in the backward pass. Experimental results on character-level language modeling show that the model reaches state-of-the-art performance, and Chung, Ahn, et al. (2017) also provide examples and, based thereon, claim that the boundary variables learn to detect an intrinsic hierarchical structure (i. e. words and phrases) in the data. However, Kádár et al. (2018) conduct reproduction experiments and observe that the learned structure is highly sensitive to certain hyperparameters, producing a fairly uninterpretable segmentation without impacting the good performance on language modeling, thereby shedding doubt on the usefulness of the HM-LSTM architecture.

Transferring the self-supervised structure learning to video input is presented by Baraldi et al. (2017). They use two LSTM layers, where the first one receives image representations, updates its memory cell, and then decides whether the current time step is the end of a video segment. If so, the hidden state gets passed to the second layer, and the first layer state is reset. The final hidden state of the second layer is then used as a representation of the whole video and passed on to the downstream task (in this case, video captioning). Their experimental results indicate that, although the learned segmentation roughly aligns with video shots, the model performs better with its learned boundaries than with using ground-truth shot decomposition.

Building on a similar idea, Campos et al. (2018) present the Skip RNN which deals with long sequences by skipping irrelevant inputs. Here, the straight through estimator is used to differentiate through the model containing binary skip variables, which are calculated for each input step and decide whether to skip the next few time steps. Experiments on a number of benchmark sequence modeling tasks show that the Skip RNN can improve computational efficiency while keeping performance competitive to a (non-skipping) LSTM baseline. Relating to the creation of EM from robot experience, the high redundancy in the data, particularly the subsymbolic information stream, makes the Skip RNN considerable. However, if used without modification, this could cause important, yet unforeseeable events like execution failures

to be skipped, which is highly undesirable.

**Learning Tree Structures** A different line of research is concerned with more explicitly inferring tree structures from sequential data. For instance, Choi et al. (2018) introduce the Gumbel Tree-LSTM, which uses a straight-through gumbel softmax estimator (replacing discrete argmax in forward pass with continuous softmax in backward pass) to decide what elements of the sequence to fuse, creating the next level of the tree. The authors present competitive results on a number of natural language tasks, and also show examples of nicely inferred sentence tree structures. Conversely, Williams et al. (2018) further investigate the trees learned by the Gumbel Tree-LSTM and similar models, and conclude that they do not correlate with (human-annotated) grammatical parse trees. Nevertheless, on various downstream tasks, they perform as well or better than models with access to ground-truth parse trees, which indicates the learned structures, despite being hard to interpret, are indeed useful.

Shen et al. (2019) present the *ordered neurons* LSTM (ON-LSTM). This model builds on an LSTM with multiple layers, adding so called master gates which ensure synchronization of neurons on different hierarchy levels. For instance, if a larger constituent like a sentence ends, then contained smaller constituents (current phrase, word) also need to end. This is enforced by the ON-LSTM architecture. While the model itself does not involve discrete decisions (and thus does not need any trick for applying backpropagation), structure trees can be inferred by using a greedy algorithm on the model's activation values.

### 3.2.3. Long-Range Transformer

Besides recurrent and convolutional architectures, recent work more and more resorts to self-attentive models, i. e. variants of the Transformer of Vaswani et al. (2017). Despite its clear advantage of providing a direct connection from every sequence position to each other one (or, as phrased by Vaswani et al., having a maximum path length between two tokens in  $O(1)$ , in contrast to convolutional or recurrent architectures), the computational complexity of  $O(N^2)$  is infeasible for large input sequence lengths  $N$ . If considering a self-attention-based model for constructing an EM, large numbers of inputs must be expected, as there is a continuous stream of experiences to be processed.

**Hierarchy** A simple approach to tackle this problem is splitting up the input into smaller entities and processing them individually. Of course, this is only applicable if the input has an inherent hierarchy. As this is the case for text documents (where words form sentences, and sentences a document), Zhang et al. (2019) propose HIBERT, a hierarchical transformer model for creating document embeddings (illustrated in the upper middle part of Fig. 3.1). The basic architecture is similar to the Hierarchical Attention Network (Yang et al., 2016) mentioned above, however, the combination of GRUs and attention is replaced by an all self-attention model, and a sentence-masking method for pretraining is introduced.

**Sparsity** If there is no external hierarchy in the data, or its use is prohibited or undesirable, a different way of decreasing the  $O(N^2)$  computational cost of the full self-attention has to be found. For such efficient transformer networks, plenty of approaches were presented in the last two years (for a more in-depth review, see e. g. Tay et al. (2020)). One possible approach is to restrict the range of attention, instead of letting every token attend directly to each other. Indeed, the original Transformer paper of Vaswani et al. (2017) already suggests to constrain attention of each input token to its local neighborhood for very long input sequences. However, this destroys the benefit of short information flow paths. Therefore, in recent work, the Extended Transformer Construction (ETC) of Ainslie et al. (2020) restricts attention of each input token to its local neighborhood, but additionally defines special *global* tokens which attend to all input and global tokens and to which all input tokens can also attend to (see Fig. 3.1, upper right). With this mechanism, information can still flow indirectly (i. e. by using a multi-layered model) from every position to each other one, but computational cost is significantly reduced. Moreover, by using learned relative positional encodings in combination with separate  $g2g$ ,  $l2l$ ,  $g2l$  and  $l2g$  attention masks

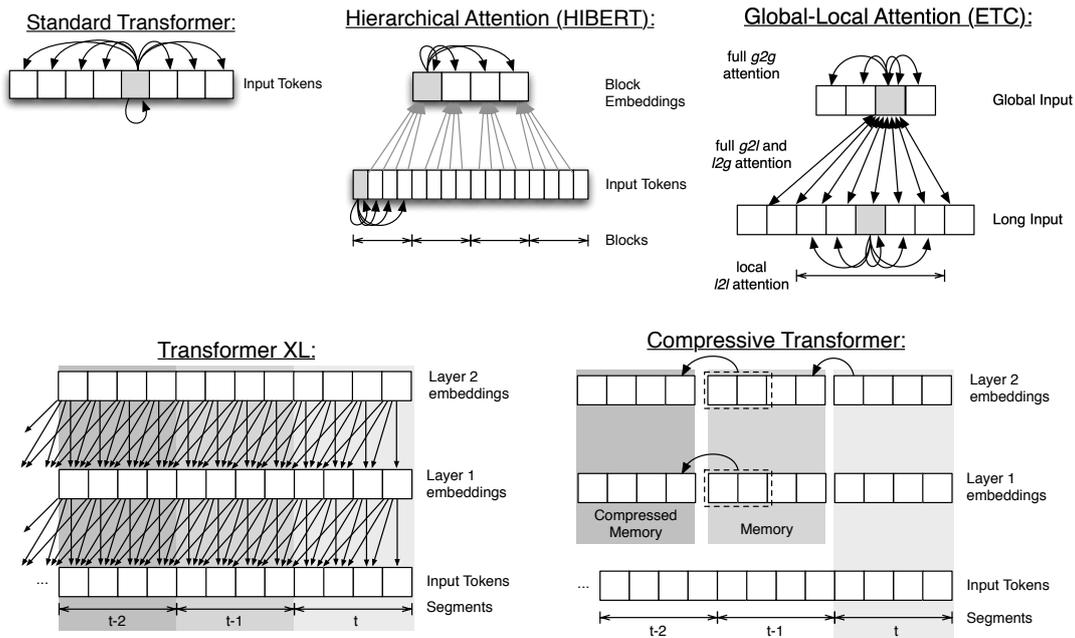


Figure 3.1.: Overview of different approaches of how to scale self-attention based models to long input sequences. The graphics are taken from Ainslie et al. (2020).

( $g$  = global,  $l$  = local,  $2$  = to), known input structure in the data can be leveraged by biasing the model, as well.

As a direct successor of ETC, the BigBird model (Zaheer et al., 2020) supplements it with a decoder, and additionally allows all input tokens to attend to a fixed number of random other positions in the input (in addition to their local neighborhood and the global tokens). On a number of challenging question-answering datasets, BigBird achieves state-of-the-art results. Concurrently to ETC and BigBird, Beltagy et al. (2020) introduced the Longformer model, based on very similar ideas. In their formulation, global attention is allowed for a small subset of the inputs, instead of an artificially inserted set of global tokens. However, this cannot beat the results of BigBird.

**Recurrence** With respect to the EMV task, an important short-coming of these sparse attention models is that they require the complete input to be available before processing it, which does not apply to a continuous stream of robot experiences. This can be circumvented by processing the input segment-by-segment and reintroducing some form of recurrence. Based on this idea, Dai et al. (2019) present the Transformer-XL, as shown in the lower left of Fig. 3.1. They split up the long input sequence into fixed-length segments, and process each segment with self-attention one after another. To enable long-term information flow, a segment level recurrence is added. In particular, the previous hidden states of each layer are cached in a queue called the “memory”, and the attention mechanism considers not only the previous layer’s activations, but also the hidden states of the previous layer from the earlier segments, i. e. the contents of memory. This recurrent state reuse, in combination with a relative formulation of learned positional encodings, led to state-of-the-art results on various language modeling datasets (at the time of publication). However, since the recurrence in layer  $k$  depends on the memory of layer  $k - 1$ , the context length of the Transformer-XL model is limited.

On top of that, Rae, Potapenko, et al. (2020) introduce the Compressive Transformer architecture (lower right of Fig. 3.1), which supplements the Transformer-XL with an additional compressed memory. Instead of discarding memory states when they get pushed out of the queue, they get compressed using a learned compression function and then added to a queue of compressed memories. The compression function is trained with a special attention reconstruction loss, separate from the task-specific objective

function. With this architecture, context length can be further increased while keeping computational cost at an affordable level. Thereby, the authors are able to beat the results of Dai et al. (2019) on language modeling, especially on datasets featuring long-range dependencies.

For constructing an EM, the limited context length is still a problem, even though it is extended by compression. To solve this problem, a same-layer recurrence must be introduced, so that there is theoretically no bound to the flow of information. This is done by the Memformer of Q. Wu et al. (2020), who propose a transformer model that works on input segments like above, and additionally keeps a set of latent memory vectors. In between the usual self-attention and feedforward module of a single encoder layer, a memory cross attention module is inserted, which allows the encoder to read from memory. Similarly, the encoder's output is not only passed to the transformer decoder, but also to the so-called memory slot attention module. It calculates the next time step's (i. e. the next input segment's) memories by performing self-attention over the encoder outputs for each memory vector. In particular, memory slot attention is masked so that each memory vector is only allowed to attend to itself and all the encoder outputs (i. e. , not to the other memory vectors). Moreover, the authors propose to use a variant of gradient checkpointing (T. Chen et al., 2016), storing only the memory vectors of each segment (without their computational graph) in the forward pass and replaying each time step for backpropagation of error during the backward pass.

## 4. Models and Methods

To approach the problem of episodic memory verbalization (EMV), this thesis builds upon the architecture presented by Bärmann et al. (2021). Thus, the following section 4.1 gives an overview thereof and highlights the key change made to the original architecture. Based on that, section 4.2 will elaborate on the details of the improved encoding and embedding of episode input data. Then, the mentioned key change, namely the new episodic memory encoder based on multi-level and long-term compressive transformer networks, is presented in section 4.3. Section 4.4 repeats the details of the speech encoder and decoder of Bärmann et al. (2021). Finally, supplementary decoders and loss functions are presented, highlighting the changes to the previous work, in section 4.5.

### 4.1. EMV Architecture

The Episodic Memory Verbalization model (shown in Fig. 4.1) is constituted of three major components: The first one is the **EM encoder**, which creates a latent representation of episodic memory (EM) by processing a stream of multi-modal key frames. A key frame captures all relevant experience data at one moment in time. As a first step, the multi-modal input data needs to be encoded as a vector and embedded into a common representation space. The details of this process are described in section 4.2. Then, the actual EM encoder processes this stream of embedded vectors to produce a sequence of latent vector representations, which constitute the EM. While Bärmann et al. (2021) allow the EM to grow linearly with respect to the number of input key frames (" $O(N)$  EM"), the present thesis improves on this and constrains the EM to a constant representation size, i. e. an " $O(1)$  EM" (see section 4.3 for how this is done).

The second major component is the **speech encoder**, which is responsible for handling a natural language question about the robot's past. Since a question is always asked at a specific point in time, the term *query* is defined to be a natural language question prepended with a textual representation (year, month, day, day-of-week, hour, minute, second) of the question timestamp, e. g. 2020 07 29 3 14 56 07 What did you do yesterday before noon?. Thus, the speech encoder actually receives the query as its input, i. e. it can build a latent representation of a question with resolved date-time references.

The natural language answer is then produced by the third component, the **speech decoder**. It combines the query encoding, i. e. the output of the speech encoder, with the content of EM, and is supposed to output the natural language answer.

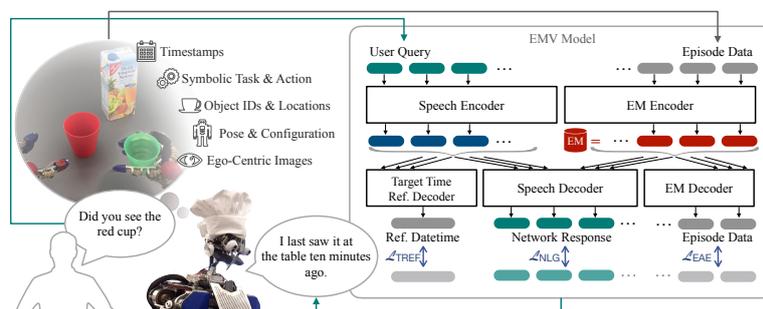


Figure 4.1.: An overview of the Episodic Memory Verbalization system as introduced by Bärmann et al. (2021). Individual components are described in detail in the following sections.

## 4.2. Episode Input Data

**Encoding** As described above, the input to the EM encoder is a sequence of highly multi-modal key frames. Therefore, the first step is to represent this data in form of vectors so that it can be passed to a neural network. Because of the different modalities involved, the approach is to construct multiple vectors with very different semantics for each key frame. In detail, the following data parts of a key frame are processed separately:

1. **Timestamp.** Date and time information of a key frame is encoded in a straightforward way, as a seven-dimensional vector, with the dimensions corresponding to year<sup>1</sup>, month, day, day-of-week, hour, minute and second, respectively.
2. **Symbolic dictionary information.** The data part of each key frame containing symbolic names is encoded using a dictionary built up during pre-processing. Thus, the information is represented as a vector, where each dimension corresponds to one of the following entries: Up to five goal tokens describing the current task of the robot, one token representing an action which changed in this key frame, up to five tokens representing the action’s arguments, and up to two tokens representing the newly detected or updated objects in the robot’s environment. Unfilled entries are set to a special padding token. For an example of how the task goal tokens are parsed from the raw planner goal, see Table 4.1.
3. **Symbolic state information.** Each key frame contains symbolic information which represents the state of a certain entity or action. This data is separated from the above dictionary information because the vocabulary here is very limited and definitely predefined in advance. Thus, the data is encoded as a vector of simple integers, with the following entries: The execution status of the current task (success = 0, abort = 1, failure = 2), current action’s execution status (started = 0, success = 1 or failure = 2), and the detection status of up to two newly detected or updated objects (updated = 0, newly detected = 1). Again, unfilled entries are set to the padding token.
4. **Subsymbolic information.** These numeric values are simply concatenated to form a long vector. Included data are: object positions for up to two newly detected or updated objects (in the robot’s root coordinate system), platform position (world coordinate system), human pose data as provided by OpenPose (Cao et al., 2018) (camera coordinate system)<sup>2</sup>, and kinematic data of the robot’s joints (angles, velocities and currents). Every dimension of this subsymbolic vector is normalized with respect to its mean and standard deviation in the training set.
5. **Pre-Encoded Latent vector.** The image associated with each key frame is converted to a latent representation using an off-the-shelf, advanced convolutional neural network architecture pre-trained on ImageNet (Russakovsky et al., 2015). Specifically, the features (i. e. inputs to the classification layer) of the ResNeXt-50 32x4d model from Xie et al. (2017) are used as image embeddings. However the image encoder, is not trained together with the EMV model, i. e. the produced representations are treated as an ordinary part of the input data.

With respect to the previous work of Bärman et al. (2021), the encoding of episode key frames is modified in two points: First, the handling of task goals is improved. Previously, the complete planner goal was treated as an atomic token, whereas now a set of symbolic predicates is parsed and used to represent the complete goal. This has the major advantage that unknown complex goals can be represented, as long as their individual predicates occurred at least once in the training set. Additionally, it avoids very rare tokens, since (assuming a fixed set of objects, locations and actions) the number of atomic predicates is much lower than the count of all possible planner goals (due to combinatorial explosion). Second, instead of representing images with latent vectors produced by the Deep EM of Rothfuss et al. (2018), ResNeXt embeddings are used as described above.

<sup>1</sup>Actually, (*year* – 2020) is written into the vector. This facilitates learning by preventing the otherwise large year value from dominating the vector.

<sup>2</sup>Note that human pose data is never present in the dataset collected from robot simulation.

raw goal	(existsK(?a1 : obj_agent_robot, ?l1 : loc_placesetting2, ?l2 : loc_placesetting2, ?o1 : obj_vanillacookies, ?o2 : obj_nesquik, ?o3 : obj_coffeefilters, ?h1 : obj_righthand) K(objectAt(?o1, ?l1)) & K(objectAt(?o2, ?l2)) & K(grasped(?a1, ?h1, ?o3)))
parse result	objectAt(vanillacookies,placesetting2) objectAt(nesquik,placesetting2) grasped(righthand,coffeefilters)

Table 4.1.: Example of parsing a raw task goal as delivered by the robot’s planner. Each of the lines in the result is treated as an atomic token.

**Embedding** The second step of allowing a neural network to work with the highly multi-modal data is to embed the different encoded vectors into a latent space. Due to the variations in semantics, individual approaches need to be used for the different data parts listed above.

The timestamp vector is simply processed by a linear layer with a bias term, extending the dimension to support higher-accuracy internal representations. An analogous method is applied to the symbolic state information vector. For subsymbolic and latent information, the (separate) linear layers compress from a high input to a lower output dimension.

Bärmann et al. (2021) also embed the symbolic dictionary information using a linear layer. However, this induces a semantically wrong ordering on the symbolic tokens based on their position in the dictionary. Thus, this procedure is adjusted and a learned embedding matrix is used instead, i. e. each symbolic token is treated as an index to the embedding matrix, selecting one of its columns. An equivalent formulation would be to use one-hot vectors for the symbolic data, and then multiply these with the learned embedding matrix. This is done for all tokens, concatenating the resulting vectors. Consequently, the vector of symbolic tokens is heavily expanded in dimension, since its length is multiplied with the embedding size. Differing from the above, to embed the symbolic goal tokens, an embedding bag is used, i. e. the output vector for this data part is the mean of the individual token embeddings.

### 4.3. Episodic Memory Encoder

In context of the EMV model, the “episodic memory” is simply a sequence of latent vectors. It is produced by the EM encoder, which receives the embedding  $e_i$  ( $i = 1 \dots N$ ) of each key frame (created as described above) and outputs the set of latent EM vectors  $EM = \{mem_j \mid j = 1 \dots M\}$ . The general architecture allows for this component to be replaceable, so that different approaches can be compared.

For an EM implementation to be actually applicable to the EMV system running on a real robot, it needs to be able to output the EM on each input key frame. Of course, the EM may (and should) be updated incrementally, but there must be no dependency on any kind of end-of-input signal, since there is simply no such signal in the real world. A question to the robot and consequently access to the current EM state may occur at any point in time, during or after execution of some tasks. Nevertheless, when training an EM model, the length of the input sample is known in advance. Therefore, if there is any benefit regarding computational efficiency, the EM may only be constructed on a “final” input time step in this setting. The model just needs to ensure that it is still applicable if *every* time step is treated as the “final” one, as is the case when actually deploying to a robot system.

As already mentioned above, the size constraint applied to EM is another important aspect when comparing different EM encoders. The most straightforward approach is to allow one latent vector per input key frame ( $M = N$ ), i. e. to allow the EM to grow with  $O(N)$  where  $N$  is the number of input key frames. However, this would not scale to deployment on a real robot, because there is neither unlimited disk space nor are there unlimited computational resources when answering a question. Thus, it is desirable to constrain the size of EM by an upper bound, i. e. to construct an  $O(1)$  EM ( $M < N$ ,  $M = const.$ ). The following subsections will present the different model architectures compared in this work.

### 4.3.1. Baseline Models

**Text-Only Baseline** The baseline Bärman et al. (2021) compare their model to is the *guess* or text-only baseline. It does not use the episode input at all, i. e. the sequence of  $e_i$  is ignored and exactly one  $mem_1 = 0$  is yielded as EM. While this is obviously no proper implementation of episodic memory, using this “fake” EM encoder in context of the complete EMV architecture allows to account for the regular nature of the natural language questions and closed-world assumption concerning the episode content. In particular, the performance of the text-only model serves as a lower bound to the episode-informed models, showing how many questions can be answered correctly resorting solely to the biases present in the training dataset.

**LSTM  $O(N)$**  As an episode-informed model, Bärman et al. (2021) use a simplistic approach, creating one latent EM vector per input key frame, i. e. they present an  $O(N)$  EM. The input sequence of key frame embeddings  $e_i$  is passed to an LSTM, followed by a linear layer:

$$\begin{aligned} h_0, c_0 &= 0 \\ h_i, c_i &= \text{LSTM}(e_i, h_{i-1}, c_{i-1}) \\ mem_i &= W_{EM}h_i \end{aligned}$$

with  $W_{EM}$  being a learned transformation matrix mapping from the LSTM hidden size to the latent EM size. Thus, the resulting episodic memory is

$$\text{EM} = \{mem_i \mid i = 1 \dots N\}$$

**LSTM  $O(1)$**  A straightforward  $O(1)$  EM baseline can be obtained using the LSTM  $O(N)$  model of Bärman et al. (2021) as described above, but keeping only the latest  $M$  latent vectors in EM. Here,  $M$  is a hyperparameter controlling the representation capacity of the memory. More precisely, the resulting EM is

$$\text{EM} = \{mem_j \mid j = (N - M) \dots N\}$$

### 4.3.2. Compressive Transformer Models

**CT** Rae, Potapenko, et al. (2020) introduce the Compressive Transformer (CT) model for dealing with very long input sequences featuring long-range dependencies in the context of language modeling (see section 3.2.3). As an extension of the Transformer-XL architecture (Dai et al., 2019), it splits up a long input sequence of tokens  $x_1, \dots, x_N$  into blocks of length  $B$ . Each block is processed with self-attention, i. e. there is an input buffer of length  $B$ , and the tokens are processed all at once when the buffer is full. Thus, the input is treated as a sequence of blocks, where each block is also referred to as an *input time step* in the following.

Additionally, some layers of the transformer are equipped with a memory of length  $K$  and a compressed memory of length  $C$ . Rae and Razavi (2020) showed that having only some of the layers have a memory (instead of all layers) very efficiently trades model performance for memory and computation costs. Thus,  $\mathcal{M}$  shall denote the index set of layers equipped with such a memory (e. g.  $\mathcal{M} = \{4, 5\}$  means only the fourth and fifth transformer layer are enriched in that way). The memory is essentially a queue keeping the respective transformer layer’s hidden states from the last input steps. When the memory is full and the oldest  $B$  memory vectors fall out of the queue, they are compressed with a compression function with ratio  $r$ , and the resulting vector is put into the compressed memory, which itself is a similar queue. When the compressed memory is full too, the  $B/r$  oldest compressed vectors are discarded. A depiction of how these memory queues work can be seen in Fig. 4.2. To use the introduced memory, the transformer’s self-attention at layer  $k$  is modified to attend not only to the hidden states of layer  $k - 1$  (or the input, for the first layer), but also to the memory and compressed memory vectors at the preceding layer.

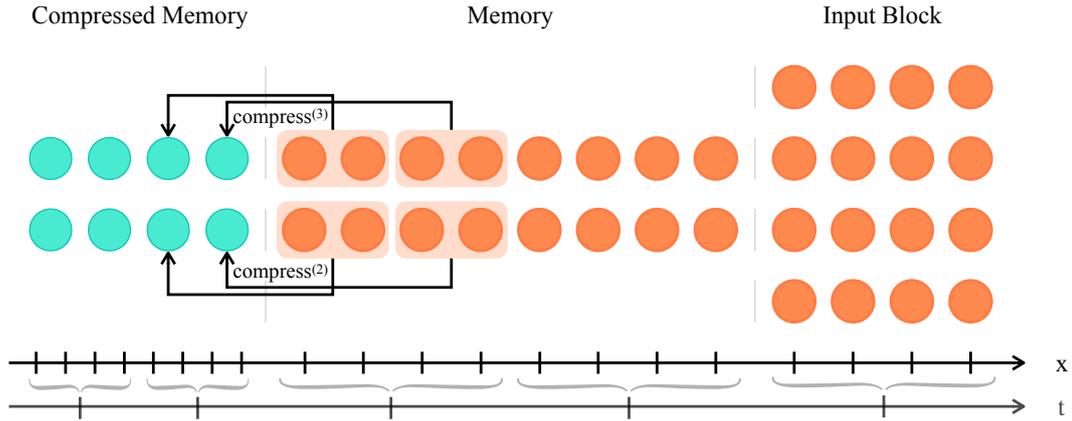


Figure 4.2.: The Compressive Transformer (CT) model as introduced by Rae, Potapenko, et al. (2020). The input block length here is  $B = 4$  (right), memory length  $K = 8$  (middle), compressed memory length  $C = 4$  (left), and compression ratio  $r = 2$  (each two memories are processed into one compressed memory). As suggested by Rae and Razavi (2020), not all of the CT’s layers are memory-equipped (in this example,  $\mathcal{M} = \{2, 3\}$ ). The upper time axis (“x”) represents the input key frames, while the lower one (“t”) stands for the input time steps, i. e. blocks of  $B$  key frames.

More formally, given a layer  $k \in \{1, \dots, L\}$ , where  $L$  is the number of layers of the transformer, and an input step  $t$ ,  $\mathbf{h}_t^k$  is defined to be the sequence of the  $B$  hidden states in that transformer layer at that input step. Similarly, let  $\mathbf{m}_t^k$  and  $\mathbf{c}_t^k$  be the sequence of the  $K$  memory and  $C$  compressed memory vectors at layer  $k$  and input step  $t$ , respectively. In the following, for a sequence  $\mathbf{s}$  of vectors, the syntax  $\mathbf{s}[a : b]$  represents the sub-sequence of  $\mathbf{s}$  from index  $a$  (inclusive, starting at 0) to index  $b$  (exclusive), while the operator  $\mathbf{s} \oplus \mathbf{t}$  concatenates two such sequences. With that definitions, the compressive transformer model involves the following calculations: First, the input from the previous layer is processed by the transformer part (as defined in Eq. (2.21)) of the model as

$$\mathbf{h}_t^k = \text{TransformerEncLayer}(\mathbf{c}_{t-1}^{k-1} \oplus \mathbf{m}_{t-1}^{k-1} \oplus \mathbf{h}_t^{k-1}) \quad (4.1)$$

where for the first layer,  $\mathbf{h}_t^0 = e_{t \cdot B}, \dots, e_{(t+1) \cdot B - 1}$  is the input embedding at input step  $t$ , and both the memory vectors  $\mathbf{m}$  and compressed memory vectors  $\mathbf{c}$  are omitted if  $(k-1) \notin \mathcal{M}$ . If layer  $k$  is not memory-equipped, the calculation proceeds with the next layer. Otherwise, i. e. if  $k \in \mathcal{M}$ , the memory is updated by inserting the new hidden states into the queue:

$$\begin{aligned} \mathbf{m}_t^k[0 : K - B] &= \mathbf{m}_{t-1}^k[B : K] \\ \mathbf{m}_t^k[K - B : K] &= \mathbf{h}_t^k \end{aligned}$$

If present, the compressed memory is updated similarly, with oldest memories  $\mathbf{m}_{t-1}^k[0 : B]$  getting compressed using a learned compression function  $\text{compress}^{(k)}$  with compression ratio  $r$ :

$$\begin{aligned} \mathbf{c}_t^k[0 : C - B/r] &= \mathbf{c}_{t-1}^k[B/r : C] \\ \mathbf{c}_t^k[C - B/r : C] &= \text{compress}^{(k)}(\mathbf{m}_{t-1}^k[0 : B]) \end{aligned}$$

The compression function is realized using a one-dimensional convolution with kernel size and stride set to  $r$ , i. e. the sequence of  $B$  old memories is divided into groups of  $r$  vectors, and each of these groups

yields one compressed vector:

$$\text{compress}^{(k)}(\mathbf{s})[i] = b^k + \sum_{h=1}^H \bar{e}_h \cdot (\text{kernel}_h \star S_i)$$

for  $i = 1, \dots, B/r$ , where  $S_i = (\mathbf{s}[i \cdot r : (i+1) \cdot r]) \in \mathbb{R}^{H \times r}$  is the matrix created by using the vectors of the  $i$ -th group of input vectors,  $H$  is the hidden size of all the involved latent vectors (including memories, compressed memories),  $\bar{e}_h \in \mathbb{R}^H$  is the unit vector with one at dimension  $h$ ,  $\star$  refers to cross-correlation, and  $\text{kernel}_h \in \mathbb{R}^{H \times r}$  as well as  $b^k \in \mathbb{R}^H$  denote the set of learned kernels and the bias vector of the compression function, respectively. This means, the convolution treats the hidden size dimension as its input and output channels, and cross-correlates its  $h$ -th kernel with the  $r$  vectors at the according position in the input sequence  $\mathbf{s}$  to produce the value in the  $h$ -th dimension of the output vector.

The compression function could be trained with the overall loss function, as used for the complete EMV model. However, Rae, Potapenko, et al. (2020) suggest to use an attention-reconstruction loss instead, and separate gradient flow so that the compression function is only trained using its own loss, and the rest of the model solely using the usual (external) loss function. The purpose is to train the compression function to learn solely to compress, but using the definition of importance given by the actual task (i. e. by using the learned attention parameters from the transformer model itself). More precisely, at each layer  $k$  and input step  $t$ , the contribution to compression loss is

$$\mathcal{L}_{\text{compress},t}^k = \|\text{SelfAttention}(\mathbf{h}_t^{k-1}, \mathbf{c}_{\text{new}}) - \text{SelfAttention}(\mathbf{h}_t^{k-1}, \mathbf{m}_{\text{old}})\|_2 \quad (4.2)$$

with the new compressed memories  $\mathbf{c}_{\text{new}} = \mathbf{c}_t^k[C - B/r : C]$ , the old memories  $\mathbf{m}_{\text{old}} = \mathbf{m}_{t-1}^k[0 : B]$  and  $\text{SelfAttention}(Q, Z) = \text{Attention}(QW_Q, ZW_K, ZW_V)$  referring to the self-attention function of the transformer, reusing its parameter matrices  $W_Q, W_K$  and  $W_V$  (but without letting the gradient flow to these parameters). Note that for calculating the compression loss, full attention is used (as opposed to multi-head attention). Thus, the parameter matrices used here are actually the concatenations of the respective per-head parameter matrices used in multi-head attention.

The resulting EM at the (final) input step  $T$  is given by the sequence of all memories and compressed memories at that time step (for each memory-equipped layer), as well as the final layer's output of the transformer network:

$$\text{EM} = \left\{ \mathbf{c}_T^k[i] \mid k \in \mathcal{M}, i = 1 \dots C \right\} \cup \left\{ \mathbf{m}_T^k[i] \mid k \in \mathcal{M}, i = 1 \dots K \right\} \cup \left\{ \mathbf{h}_T^L[i] \mid i = 1 \dots B \right\}$$

The compressive transformer model is very appealing for constructing an episodic memory, since the ability to compress and abstract old memories is key for reaching a long-range EM. Additionally, it fulfills the requirements for deployment to a real robot, as it processes the input block by block and thus does not require to store the complete input or wait for a never-occurring end-of-input token. However, because the self-attention at layer  $k$  only attends to the activations and memories from the preceding layer  $k-1$ , the model has a limited context length (a property inherited from the Transformer-XL architecture). This means, for an input sequence longer than the context length, there is no dependency path from the last output to the first input token. Thus, the above construction of EM is an  $O(1)$  EM, however with no chance of remembering events longer ago than the context length, which is  $B + L \cdot (K + C \cdot f)$ .

**CT + LT** To construct an actually useful  $O(1)$  EM, a long-term (LT) memory should be added so that an unlimited context length is possible. For this purpose, instead of just discarding the oldest  $B/r$  compressed memory states, they are passed to a recurrent long-term memory component. It first applies attention to all the incoming (compressed) states with respect to the last LT state, yielding an LT update candidate vector. This is then given to the recurrence realized by a Gated Recurrent Unit (GRU). The basic idea is similar to the Memformer model of Q. Wu et al. (2020), but combined with the compressive transformer architecture. Fig. 4.3 shows a graphical overview of this Long-Term Compressive Memory Transformer (CT + LT) model.

In particular, at the current input step  $t$  and suppressing the layer indices, let  $\mathbf{c}_{t-1}[0 : B/r]$  be the

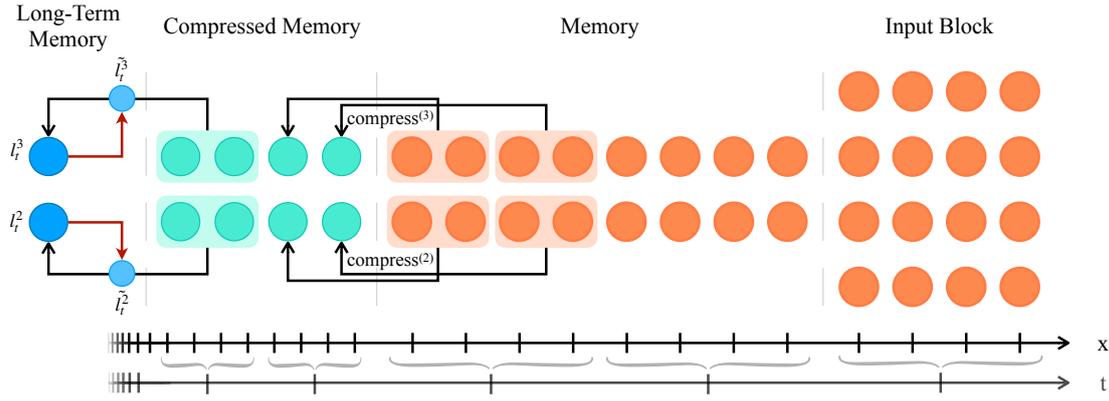


Figure 4.3.: The Long-Term Compressive Memory Transformer (CT + LT) model. Parameters in this example are as in Fig. 4.2.

oldest compressed memory states about to be discarded in the plain CT architecture. Defining  $\Gamma = (\mathbf{c}_{t-1}[0 : B/r])$  to be the matrix aggregating these states, the LT update candidate  $\tilde{l}_t$  is calculated using scaled dot-product attention as defined in Eq. (2.17), with the query  $l_{t-1}$  and transformed versions of  $\Gamma$  as keys and values:

$$\tilde{l}_t = \text{Attention}(l_{t-1}, \Gamma W_{\text{LT-K}}, \Gamma W_{\text{LT-V}}) \quad (4.3)$$

where  $W_{\text{LT-K}}$  and  $W_{\text{LT-V}}$  are learned key and value transformations, respectively. Then, the LT memory is updated using a GRU:

$$l_t = \text{GRU}(\tilde{l}_t, l_{t-1})$$

For the first input step touching the LT memory, i. e. the step after the first one completely filling the compressed memory,  $l_{t-1}$  is replaced with 0 in the calculation of  $\tilde{l}_t$  and the GRU update is skipped, i. e.  $l_t$  is simply set to  $\tilde{l}_t$ .

To make use of the introduced LT memory, Eq. (4.1) is modified to also attend to the long-term memory state of the previous layer:

$$\mathbf{h}_t^k = \text{TransformerEncLayer}([l_{t-1}^{k-1}] \oplus \mathbf{c}_{t-1}^{k-1} \oplus \mathbf{m}_{t-1}^{k-1} \oplus \mathbf{h}_t^{k-1})$$

Similarly, the resulting EM at the (final) input step  $T$  is the same as for the CT model, but enriched with the last LT state for each memory-equipped layer:

$$\begin{aligned} \text{EM} = & \{l_T^k \mid k \in \mathcal{M}\} \cup \{\mathbf{c}_T^k[i] \mid k \in \mathcal{M}, i = 1 \dots C\} \\ & \cup \{\mathbf{m}_T^k[i] \mid k \in \mathcal{M}, i = 1 \dots K\} \\ & \cup \{\mathbf{h}_T^L[i] \mid i = 1 \dots B\} \end{aligned}$$

**ML CT + LT** As a further extension of the compressive transformer model, multiple compression levels are introduced. This is independent of the long-term memory presented above, but might also be combined. In particular, when the oldest compressed memories are about to be discarded (for CT) or merged into the LT memory (for CT + LT), they instead get compressed again and put into another queue of level-2 compressed memories. Obviously, this can be repeated for multiple compression levels, as long as the number of memories falling out of one level per time step remains evenly divisible by the next levels compression ratio. An example of this architecture is shown in Fig. 4.4.

To formalize the above, instead of using a single compression ratio  $r$ , each compression level  $\lambda \in \{1, \dots, \Lambda\}$  introduces its own ratio  $r_\lambda$ . Similarly, the compressed memory length  $C$  is replaced by

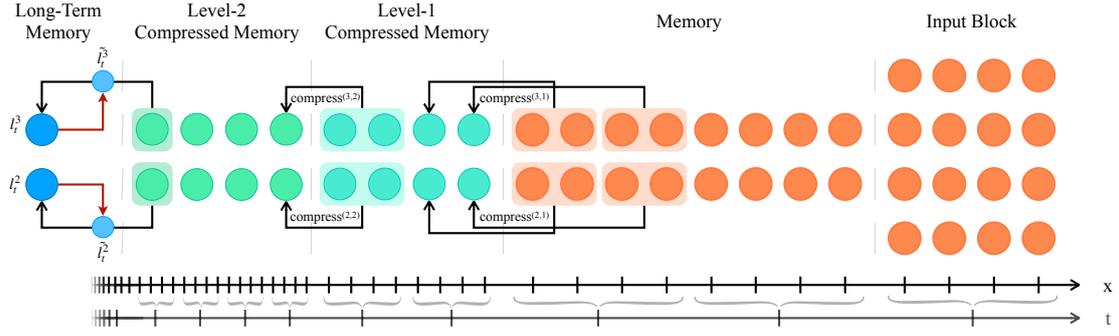


Figure 4.4.: The Multi-Level compressive transformer model with long-term memory (ML CT + LT). This is an example with  $\Lambda = 2$  compression levels,  $B = 4$ ,  $K = 8$ ,  $C_1 = C_2 = 4$ , and  $r_1 = r_2 = 2$ .

multiple  $C_\lambda$ . Note that all (memory-equipped) transformer layers share these hyperparameters. Let  $\mathbf{c}_t^{k,\lambda}$  be the sequence of vectors in the level- $\lambda$  compressed memory at layer  $k$  and input step  $t$ . Then, the multi-level compressive transformer with long-term memory (ML CT + LT) calculation on a memory-equipped layer  $k$  is as follows:

Self-Attention from Eq. (4.1) is extended to include both LT memory as well as compressed memories from all compression levels from the previous layer:

$$\tilde{\mathbf{c}}_{t-1}^{k-1} = \bigoplus_{\lambda=1}^{\Lambda} \mathbf{c}_{t-1}^{k-1,\lambda}$$

$$\mathbf{h}_t^k = \text{TransformerEncLayer} \left( [l_{t-1}^{k-1}] \oplus \tilde{\mathbf{c}}_{t-1}^{k-1} \oplus \mathbf{m}_{t-1}^{k-1} \oplus \mathbf{h}_t^{k-1} \right)$$

Memory  $\mathbf{m}_t^k$  is updated as in the CT model. Let  $\mathbf{m}_{t-1}^k[0 : B]$  be the  $B$  vectors falling out of the memory at input step  $t$ . Then, compressed memory is updated iteratively, for each compression level  $\lambda = 1 \dots \Lambda$ :

$$B_\lambda = \frac{B_{\lambda-1}}{r_\lambda}$$

$$\mathbf{c}_{t-1}^{k,\lambda}[0 : C_\lambda - B_\lambda] = \mathbf{c}_{t-1}^{k,\lambda}[B_\lambda : C]$$

$$\mathbf{c}_{t-1}^{k,\lambda}[C - B_\lambda : C] = \text{compress}^{(k,\lambda)}(\mathbf{c}_{t-1}^{k,\lambda-1}[0 : B_{\lambda-1}])$$

Here,  $B_\lambda$  is the compressed length of the input block of (uncompressed) length  $B$  at compression level  $\lambda$ . For the first compression level  $\lambda = 1$ , the references to the previous level refer to the (non-compressed) memory, i. e.  $B_0 = B$  and  $\mathbf{c}_{t-1}^{k,0} = \mathbf{m}_{t-1}^k$ . Each compression level (on each layer) has its own learned parameters for the compression function, hence the indices  $\text{compress}^{(k,\lambda)}$ . Compression loss is calculated as done for the plain CT model, but separately for each compression level:

$$\mathcal{L}_{\text{compress},t}^{k,\lambda} = \|\text{SelfAttention}(\mathbf{h}_t^{k-1}, \mathbf{c}_{\text{new}}^\lambda) - \text{SelfAttention}(\mathbf{h}_t^{k-1}, \mathbf{c}_{\text{old}}^{\lambda-1})\|_2 \quad (4.4)$$

with the indices new and old defined equivalently to Eq. (4.2), and the base case  $\mathbf{c}_{\text{old}}^0 = \mathbf{m}_{\text{old}}$ .

The ML CT + LT architecture also includes a long-term memory. For this purpose, a straightforward modification to the CT + LT model is possible, as in the calculation of  $\tilde{l}_t$ , the old compressed memories simply need to be replaced by the old level- $\Lambda$  compressed memories. In particular, in Eq. (4.3),  $\Gamma$  is modified to be

$$\Gamma = \left( \mathbf{c}_{t-1}^{k,\Lambda}[0 : B_\Lambda] \right)$$

i. e. the matrix created by concatenating the last compression level's old vectors, which would otherwise

be discarded at this input step. The remaining equations for calculating  $l_t^k$  stay as defined for the CT + LT model.

The resulting EM at the (final) input step  $T$  combines the output hidden states with all memory states, i. e. the memory itself, all levels of compressed memories, and the LT memory:

$$\begin{aligned} \text{EM} = & \{l_T^k \mid k \in \mathcal{M}\} \cup \left\{ \mathbf{c}_T^{k,\lambda}[i] \mid k \in \mathcal{M}, \lambda = 1 \dots \Lambda, i = 1 \dots C \right\} \\ & \cup \left\{ \mathbf{m}_T^k[i] \mid k \in \mathcal{M}, i = 1 \dots K \right\} \\ & \cup \left\{ \mathbf{h}_T^L[i] \mid i = 1 \dots B \right\} \end{aligned}$$

## 4.4. Speech Encoder and Decoder

The two parts of the EMV model dealing with natural language, i. e. the speech encoder and decoder, are adopted unmodified from the previous work of Bärmann et al. (2021). Both utilize the transformer architecture as introduced by Vaswani et al. (2017). Mathematical details of this model are explained in section 2.3.

Specifically, the “small” T5 model of Raffel et al. (2020) is used. They pretrain their model on a mixture of unsupervised and supervised tasks, which are all cast to a uniform text-to-text format. This enables them to apply a single transformer encoder-decoder model and the same cross-entropy loss function to all the tasks. For the unsupervised task, they use a denoising objective, i. e. certain spans of tokens in a sentence are masked, and the target is to predict the missing words. Concerning supervised objectives, they mix data from machine translation, question answering, abstractive summarization, and text classification.

Both the speech encoder and decoder are taken from the pretrained model of Raffel et al. (2020) without any modifications. The encoder receives the query (i. e. the question together with the question timestamp) as input, and yields a latent sequence of the same length, referred to as the query encoding. To enable the decoder to attend to both the query encoding as well as the EM content, a simple approach is chosen: Since both the query encoding as well as the EM content are sequences of latent vectors, they can be simply concatenated to form an extended context passed to the transformer decoder. However, to enable this, the vectors of both sequences need to have the same dimensionality. Although a straightforward solution would be to set the latent EM vector size to match the hidden size used by T5, this would also incur a high number of parameters in the EM encoder, leading to a more complex optimization problem. Thus, a simple workaround is used instead: The hidden size of EM vectors is only constrained to be a divisor of T5’s hidden size. Then, when concatenating both latent sequences, the EM vectors are repeated so that their number of hidden dimensions matches that of T5. Afterwards, the speech decoder uses the resulting combined sequence as its encoder input, and the target utterance as its primary input, and generates the response word by word in evaluation or in parallel with teacher-forcing during training or validation. Loss is calculated using cross-entropy between the model’s output and the target utterance.

## 4.5. Supplementary Decoders & Loss Functions

While the EMV model could be trained solely with the natural language generation (NLG) loss, Bärmann et al. (2021) show that it is beneficial to add additional decoders and additive loss contributions to facilitate and analyze learning of the EMV task.

**Episode Decoder** Inspired by the auto-encoder loss for the Deep EM by Rothfuss et al. (2018), the model is enriched with an episode decoder. It receives the output of the EM encoder, i. e. the sequence of latent EM vectors, to map it back to its original input, i. e. the sequence of key frames. If the EM grows linearly in size with respect to the number of input key frames, as is the case in the work of Bärmann

et al. (2021), a straight-forward approach can be used, with a linear transformation mapping each latent EM vector back to the input key frame separately and independently. This is not possible with new  $O(1)$  EM, because there are less latent EM vectors than key frames to reconstruct. Thus, another transformer decoder is used, receiving the embedded key frames as target sequence and attending to the latent EM vectors via its encoder-decoder attention. After a correct-length sequence of embeddings is produced in that way, each vector  $h_i$  of this sequence is processed identically to how the latent EM vectors are processed by Bärman et al. (2021):

$$\begin{aligned} o_{\text{DT}}^i &= \text{logsoftmax}(W_{\text{DT}}h_i + b_{\text{DT}}) \\ o_{\text{Sym}}^i &= \text{logsoftmax}(W_{\text{Sym}}h_i) \\ o_{\text{Sub}}^i &= W_{\text{Sub}}h_i + b_{\text{Sub}} \\ o_{\text{Latent}}^i &= W_{\text{Latent}}h_i + b_{\text{Latent}} \end{aligned}$$

where the  $o_*^i$  are the output values for date-time, symbolic, subsymbolic and latent data parts, respectively, and the  $W_*$ ,  $b_*$  are learned weight matrices and bias vectors. The dimensions are

$$\begin{aligned} \dim(W_{\text{DT}}) &= (60 \cdot 7) \times H \\ \dim(W_{\text{Sym}}) &= (S \cdot V) \times H \\ \dim(W_{\text{Sub}}) &= U \times H \\ \dim(W_{\text{Latent}}) &= L \times H \end{aligned}$$

with  $60 \cdot 7$  being the maximum number of possible values in each date-time dimension (i. e. the DT vocabulary size) times the number of date-time dimensions used (year, month, day, day-of-week, hour, minute and second),  $S$  the number of symbolic key-frame entries,  $V$  the symbolic dictionary size,  $U$  the number of sub-symbolic entries,  $L$  the dimension of latent image vectors and  $H$  the hidden dimension as produced by the EM encoder. The episode auto-encoder (EAE) loss with respect to the input key frames is then computed as

$$\mathcal{L}_{\text{EAE}} = \beta_1 \mathcal{L}_{\text{DT}} + \beta_2 \mathcal{L}_{\text{Sym}} + \beta_3 \mathcal{L}_{\text{Sub}} + \beta_4 \mathcal{L}_{\text{Latent}} \quad (4.5)$$

Date-time loss  $\mathcal{L}_{\text{DT}}$  and symbolic loss  $\mathcal{L}_{\text{Sym}}$  are classification loss functions, i. e. cross entropy, whereas subsymbolic and latent loss contributions  $\mathcal{L}_{\text{Sub}}$  and  $\mathcal{L}_{\text{Latent}}$  directly regress their target values, using MSE and L1 loss functions, respectively.

**Time-Reference Decoder** For a further analysis and also to support learning of the natural language task, Bärman et al. (2021) also add an additional *time-reference decoder* to the model. It uses a simple formulation of softmax attention (Bahdanau et al., 2015) on the query encoding, with the goal of mapping it to the date-time referenced in the query. Output is encoded in a one-hot manner, and cross-entropy loss is used to inject the additional loss contribution. Target values, i. e. the correct timestamp references, are available because the dialog part of the training dataset is generated automatically, as explained in section 5.1.2. The purpose of this supplementary decoder is to analyze and facilitate the learning of how to resolve date-time references in the query. For instance, given a current timestamp and the question “What did you do three days ago at noon?”, the model must be able to understand that reference to look up the correct item in episodic memory.

**Complete Loss Function** To summarize, the complete loss function  $\mathcal{L}$  for training the EMV model is

$$\mathcal{L} = \alpha_1 \mathcal{L}_{\text{NLG}} + \alpha_2 \mathcal{L}_{\text{EAE}} + \alpha_3 \mathcal{L}_{\text{TREF}} + \mathcal{L}_{\text{compress}}$$

with  $\mathcal{L}_{\text{NLG}}$ ,  $\mathcal{L}_{\text{EAE}}$  and  $\mathcal{L}_{\text{TREF}}$  being the loss from natural language generation, episode auto-encoder and time-reference reconstruction, respectively. This is nearly identical to Bärman et al. (2021), solely adding the compression loss  $\mathcal{L}_{\text{compress}}$  from the newly introduced compressive transformer model, which is calculated by summing the contributions from Eq. (4.2) – or, in case the ML CT is used, Eq. (4.4)

– over all time steps, transformer layers (and compression levels, in case of ML CT), and normalizing by the number of memory-equipped layers and length of the input sequence. The  $\alpha_i$ , as well as the  $\beta_i$  from Eq. (4.5), are hyperparameters, with  $\sum_i \alpha_i = \sum_i \beta_i = 1$ . As argued by Rae, Potapenko, et al. (2020),  $\mathcal{L}_{\text{compress}}$  does not require a similar mixing weight, since the set of parameters where gradient can flow for this and the other loss parts is disjoint ( $\mathcal{L}_{\text{compress}}$  gradient only affects the parameters of the compression functions, while all other loss contributions optimize everything of the model excluding the compression functions).

## 5. Evaluation

### 5.1. Datasets

A dataset sample for the EMV task consists of two major ingredients: 1) A robot episode recording (RER), which states what the robot experienced in the past and 2) A pair of a natural language question and an appropriate answer, according to the content of the RER.

As in Bärmann et al. (2021), a RER is defined as a stream of symbolic information (task, goal, action and action arguments as stated by the symbolic planner, action execution status (started, running, interrupted, finished with success or failure), detected objects IDs) as well as subsymbolic data (camera images, robot configuration, estimated platform position, estimated object poses, detected human poses) and timestamps. From this continuous stream of multi-modal data, so called “key frames” are subsampled. This is achieved by applying a time window of five seconds; for instance, only the latest robot pose and camera image are kept in each time window. However, when the symbolic action or action status changes, a new time window is set up immediately, flushing the latest data to the previous key frame and starting a new one. The purpose thereof is to prevent important events from being suppressed by the constant five seconds interval. In the following, an episode is defined to be a contiguous sequence of such key frames, concerned with achieving a single planner goal. Furthermore, the term “history” denotes a concatenation of multiple episodes.

Second, natural language questions and answers fitting to the RERs need to be collected. Hereby, the task is constrained such that only questions related to robot experiences are included. For instance, pure image recognition questions like “what does the text on the bottle read?”, are excluded, while image queries concerned with episode data like “how many people did you meet yesterday?” are allowed. The question also comes with a timestamp of when this question is asked (which obviously needs to be temporally after all events in the corresponding RER).

#### 5.1.1. Simulated Episode Recordings

Since gathering huge amounts of training data is crucial for training a deep neural model successfully, Bärmann et al. (2021) propose to collect RERs in an automated way. Therefore, the simulation component of the ArmarX robot framework (Vahrenkamp et al., 2015) is used to randomly create scenes with different objects at different places in a kitchen environment. Then, an episode recorder component is used to capture single-episode RERs, automatically iterating over all created scenes and possible planner goals to execute. In the present work, the episode data collection procedure is further improved by the following points:

First, diversity with respect to the objects handled by the robot is significantly increased. Before, there were only three objects that could be moved around the scene. Now, the RERs involve a total of 14 objects the robot interacts with. Similarly, location diversity is slightly improved. In particular, there is one new start location for the objects (i. e. one new location the robot can go to pick up objects).

Moreover, the set of possible planner goals is extended. In the previous work, tasks were constrained to be either “grasp object A with your left/right hand”, “move object A to location X” or “move object A to location X and object B to location Y”. This was extended by adding three more possibilities: “move object A to location X, B to Y and C to Z”, “move object A to location X, B to Y and also grasp object C with your left/right hand” and “grasp object A with your left and object B with your right hand<sup>1</sup>”.

---

<sup>1</sup> Actually, there was no successful simulation run of this goal. However, since the goal of this work is not to fix problems of the ArmarX simulation environment, this was just accepted and thus there are no such RERs in the dataset.

Table 5.1.: Statistics on the simulated robot experience data collection. “old” refers to the dataset of Bärmann et al. (2021), while “new” is the dataset collected in the present thesis.

		#RERs	#episodes	#histories of length				
				1	2	3	5	10
old	train	708	2,832	11,326	11,326	8,496	5,660	
	valid	152	304	6,080	6,080	6,060	6,000	
	test	151	151	100	100	100	100	100
new	train	3,354	3,350	6,700	6,700	6,696	6,698	
	valid	419	419	8,380	8,360	8,340	8,300	
	test	419	418	100	100	100	100	100

Furthermore, the random scene generator was improved to place up to six objects into a scene. It also generates a more realistic placement of the objects in the kitchen (e. g. no overlapping textures etc.).

Finally, the robot grasps were reviewed and manually improved where needed. Because of that, less simulation runs fail, and the put-down actions of the robot look more realistic. Especially, some effort was done to prevent objects from floating in the air above the surface where the robot puts them down. For exemplary content of a simulated RER in the newly collected dataset, see Fig. C.1 in the appendix.

Table 5.1 provides an overview of the dimensions of the collected dataset of simulated recordings, comparing it to the dataset from previous work. The number of RERs is the actual number of executed robot simulations (excluding the ones that failed immediately, e. g. due to the robot’s planner finding no plan to achieve the requested goal). For the old dataset, this is smaller than the number of episodes in the dataset, as Bärmann et al. (2021) augment the recordings by creating copies with noise added to subsymbolic data. As the number of recordings in the new dataset is significantly higher, this procedure is not adopted, and the number of RERs matches the number of episodes (except for minor differences due to erroneous recordings). Besides containing more of them, the episodes in the new dataset are also more complex and contain more key frames than the old ones (this is illustrated by Fig. A.1 in the appendix). By picking random episodes and concatenating them (adjusting date-times accordingly), histories of different lengths are randomly generated. Note that for the train and validation set, the history length refers to the maximum history length in this set (e. g. when training with the histories of length three, there actually is an equal portion of histories of length one, two and three in the dataset), while for the test set, only histories with exactly the given length are used. Sampling is performed without replacement until all episodes are used once, then resetting the set of episodes and repeating this process for a certain number of iterations (for train and validation) or until the desired number of histories is reached (for test). The numbers of iterations were chosen differently to reach a reasonable dataset size. While the validation set contains more histories due to a different dataset generation scheme as explained in the next section, the test set size is kept constant at 100 histories of the respective length.

### 5.1.2. Grammar-Generated Q&A

To efficiently make use of the large number of simulated RERs, this work uses the grammar-based QA generation script proposed by Bärmann et al. (2021). The script considers the symbolic information in the recordings, randomly chooses a question time (after the last episode), and then conditionally generates a huge number of possible questions and answers per episode.

In total, three different grammar-generated datasets were used to produce the results presented in section 5.4. Their content and characteristics are explained in the following paragraphs:

**previous** This is the dataset presented in Bärmann et al. (2021). The simulated episodes here work with a very limited set of objects, actions and locations. This thesis conducts experiments on the *previous*

dataset to enable a fair comparison to the previous work. For completeness, a list of the different types of questions and answers as generated by this grammar follows:

- *Current date & time*: The question asks for the current date or the current time, and the answer depends on the given query time. Examples:
  - “What is the time?” → “It is 17:45.”
  - “What day is today?” → “It is August 05.”
- *Action*: The question asks for a specific task the robot performed. Tasks (and corresponding arguments) are derived from the symbolic planner goals. In case the question does not contain a time reference, the answer refers to the last episode in the history. Otherwise, a specific episode is referred to by its time. Examples:
  - “What did you do at noon?” → “i moved the vitamin juice to the small table and the red cup to the round table.”
  - “Explain your exercises on September 23 at 02 PM!” → “i grasped the green cup.”
- *Problems*: The question asks if the robot had any problems during a specific time interval or when executing a specific action. Examples:
  - “Explain how your task to grasp the cocoa worked at noon!” → “Everything went fine.”
  - “Did you have any problem yesterday?” → “Yes, i failed to grasp the green cup yesterday at 19:12.”
- *Past time*: Here, the robot is supposed to say when it performed a specific task or when it first or last interacted with or saw a specific object. Examples:
  - “When did you grasp the oil?” → “I did it 215 days ago at about 14 o clock.”
  - “When did you last see the corn?” → “It was yesterday at 10:20.”
- *Location*: The question asks for where the robot was at a specific point in time. Examples:
  - “Where have you been at 09:47?” → “I just arrived at the countertop.”
  - “Where have you been three days ago at 10:36?” → “I just arrived at the setting place.”
- *Object*: Here, the robot has to remember which object it moved at a specific time in the past. Examples:
  - “Explain which object you moved to the setting place before noon!” → “I moved the vanilla cookies.”
  - “Tell me which object you moved to the countertop in the evening!” → “I think you are talking about the popcorn.”
- *Duration*: The question asks for the duration of an action in the past. Examples:
  - “How long did it take to release the corn yesterday before noon?” → “It took about two minutes.”
  - “How long exactly did it take to grasp the oil today at noon?” → “It took 25 seconds.”

**short dialogs** This dataset builds on the newly collected set of simulated robot episode recordings as described in section 5.1.1. To produce questions and answers, the grammar was only minimally adjusted to allow the use of the newly involved objects and locations. No new question types were added.

**long-term dialogs** This dataset builds upon the *short dialogs* set, but the grammar was extended to produce special questions and answers requiring long-term information storage. The added Q&A are:

- *Summarize*: Dialogs, where the question is asking for a summary of events in a given time period (today, this week, etc.), and the answer is a long list of what the robot did in this period. Answer sentences are connected with temporal conjunctions like “two days later”. Examples:
  - “Summarize your activities of today!” → “I did not do anything today.”
  - “Explain everything you did this month!” → “Four days ago at 14 o clock, I moved the corn and the popcorn to the sink. One day later, I tried to but failed to move the vitalis to the setting place,

Table 5.2.: Number of generated question-answer pairs making up the *previous* and *long-term dialogs* datasets splits. The columns refer to different history lengths.

		1	2	3	5	10
<i>previous</i>	train	15,625,862	22,268,398	21,804,631	21,424,698	
	valid	60,048	60,296	60,042	59,385	
	test	67,047	126,884	187,192	309,087	609,350
<i>long-term dialogs</i>	train	15,296,450	21,349,717	27,507,619	40,112,705	
	valid	83,800	83,600	83,400	83,000	
	test	115,713	199,417	296,560	482,583	944,945

the orange juice to the sink and the oil to the countertop. Then, I tried to but failed to grasp the apple tea.”

- *How often*: Here, the question refers to a specific object resp. location and the answer states how often the robot grasped this object resp. moved to or has been at that location. Examples:
  - “How often did you grasp the cookies?” → “I grasped it four times.”
  - “How often have you been at the fridge?” → “I have never been there.”

For an overview on the number of QA pairs generated for the individual dataset splits, see Table 5.2. Note that the number of validation samples is kept low, although the validation set contains more histories than the training set (see Table 5.1). The cause for this is a different dataset generation scheme: While the training set simply contains all possible dialogs for each history (thus leading to huge numbers), the validation set subsamples only a few possible QA samples per history. This strategy aims at keeping the time cost for validation runs low, while still using a representative set of histories (especially with respect to date-time variation). For the test set, QA sampling is not used that extensively to ensure that none of the above question types is underrepresented.

### 5.1.3. Generalization Test Sets

Bärmann et al. (2021) present additional small test sets to assess the generalization capabilities of the model.

First, instead of collecting RERs from a simulated robot, the actions of a real robot can be recorded. For this purpose, the same episode recorder component already employed in simulation can be used. Excerpts from such a recording are illustrated by Fig. C.2 and Table C.1 in the appendix. However, since data acquisition on a physical robot system is very work-intensive and error-prone, this thesis did not collect additional real robot samples.

Second, because of the limited nature of grammar-generated questions and answers, collecting language data from human annotators is desirable. In case the data is supposed to be used for training purposes, both questions as well as answers need to be entered. In contrast, if only a natural-language generalization test set is to be created, annotating solely questions is sufficient. Bärmann et al. (2021) already introduced a website<sup>2</sup> for this purpose, which was extended in the present thesis to prepare for a future large-scale, crowd-sourced Q&A collection (especially working on log-in mechanism, sample distribution and credit assignment). Screenshots of this website are shown in appendix D. While such a data collection is highly desirable to enhance a systems capability to generalize to unseen natural language utterances, it stays out of the scope of this thesis. Particularly, in contrast to the natural task of answering a question about a given episode, the collection of interesting questions in a crowd-sourced setting is not straightforward. Simply providing a few examples and then stating the task to ask the robot a question would very likely lead to tedious repetitions of the sample sentences. As a possible solution,

<sup>2</sup><https://em.dataforlearningmachines.com/emv-data-collection/introduction>

the Q&A data collection could be embedded into a game-like environment with a goal easily accessible by human annotators unacquainted with robotics. This task is left up to future work.

To summarize, generalization test sets are formed from all combinations of grammar-generated or human-annotated questions and executions from simulation or a real robot. Using the definitions of Bärmann et al. (2021), this yields the following datasets: The *simulated-robot-grammar-generated* test set is split up with respect to the different grammars as described in section 5.1.2. Since human Q&A data acquisition is postponed to future work, the *simulated-robot-human-annotated* set is used without modifications. Similarly, as mentioned above, no new recordings on a real robot were collected. Thus, both the *real-robot-grammar-generated* as well as the *real-robot-human-annotated* set are kept unchanged.

## 5.2. Evaluation Metrics

Evaluating the quality of natural language answers in context of episodic memory verbalization is far from trivial. For instance, the same answer can be expressed in very different ways, both concerning the syntactical and the lexical formulation of the utterance. This can be illustrated by the question “Did you have any problems yesterday?” with its two semantically equivalent but lexically very different answers “No.” and “Yesterday, everything went fine”. Moreover, the questions themselves might be ambiguous or unspecific, thus allowing for multiple possible correct answers on the semantic level. Considering a history of the robot where it performed two different tasks today, either one of them as well as a sentence combining both would be a valid response to the question “What did you do today?”.

To account for these difficulties, and assuming one is given an EMV sample along with a target utterance, Bärmann et al. (2021) propose to semantically classify EMV response hypotheses using the following catalog of categories:

- *Correct*: The answer is correct, i. e. it contains all the facts given in the target utterance, and no more.
- *TMI correct*: The answer is correct, and there is some additional information given in the hypothesis, but these additional facts are semantically valid with respect to the history of robot experiences in this sample (“tmi” = “too much information”).
- *TMI wrong*: The answer is correct in the sense of above, i. e. all information contained in the target utterance is given in the hypothesis. However, there are some additional facts present in the hypothesis, and those are semantically wrong with respect to this EMV sample.
- *Partially correct*: Some of the facts contained in the target utterance are given in the hypothesis, but some information is missing or wrong. This category is also used if all target facts are present in the hypothesis, but they are combined in a wrong way, as illustrated by the target “I moved the green cup to the sink and the red cup to the countertop” and the hypothesis “I moved the red cup to the sink and the green cup to the countertop”.
- *Partially correct, only action*: The hypothesis contains none of the facts contained in the target utterance, except for one which deals with the robot’s action. This is defined as a separate category to account for that fact being easy to guess because of the very limited number of actions (grasp, move, put down) in the simulated dataset.
- *Wrong*: None of the facts in the target is present in the hypothesis. Nevertheless, the answer intent is correct, i. e. the hypothesis could be considered a valid response given the question without any context of what actually happened in the robot’s past.
- *Inappropriate*: The answer is not even related to the question (e. g. “what did you do?” → “it is tuesday”), or it makes no sense (syntactically or lexically) at all.

In this context, a *fact* is a particular piece of information related to the intent of the question. For example, in context of the question “What did you do today?”, the utterance “I moved the green cup to the sink in

the morning” contains four facts: “move”, “green cup”, “sink” and the time reference “in the morning”. Using this definition, complementary to the above categorization system, it is also possible to count the number of correct facts  $C$  and incorrect facts  $I$  in the hypotheses, as well as the number of facts in the target utterances, i. e. the expected facts  $E$ . Thereby, *information precision*  $p$  and *information recall*  $r$  can be defined as

$$p = \frac{C}{C+I} \qquad r = \frac{C}{E}$$

With the above metrics defined, the question remains of how to calculate these numbers given a model’s test outputs. For this purpose, this thesis extends the heuristic evaluation script of Bärman et al. (2021). It is applied to evaluate the model hypotheses on the datasets with grammar-generated questions and answers, assigning each hypothesis to one of the presented answer categories by inspecting the model output with regard to the complete EMV sample (i. e. the question, question timestamp, target utterance and episodic content). First, the question is analyzed to distinguish the different types of grammar-generated questions defined in section 5.1.2. Based thereupon, different procedures extract the relevant facts from the target utterance and use them to assess the quality of the model’s hypothesis, based on the known sentence structure induced by the known grammar which generated the data. Per sample and hypothesis, the result is an assignment to one of the categories, as well as the numbers  $C$ ,  $I$  and  $E$  as defined above. These get aggregated over different sets of test samples, as further elaborated in section 5.4.

For assuring the reliability of the heuristic evaluation script described above, the script’s output is manually reviewed for all cases of questions and answers. To emphasize again, this is only possible because the grammar-generated data has a known structure, hence a reverse analysis of all known cases is possible. For evaluating a model’s responses to human-annotated questions, human assessment is used, i. e. a given question, hypothesis and video of robot actions including date-time information is inspected and assigned to one of the presented categories by hand (using the interface shown in appendix D).

### 5.3. Training Procedure

Bärman et al. (2021) suggest a three-step procedure to train the EMV model: First, a simple dimensionality reduction is learned on the latent image vectors as produced by the Deep EM of Rothfuss et al. (2018). Second, an episode auto-encoder (EAE) model is trained by using only the EM encoder and EM decoder part of the complete EMV model. Lastly, the EMV model is initialized with the weights from EAE pretraining and then optimized with respect to the overall loss function, whereby a curriculum learning strategy concerning the number of episodes per history is applied. In particular, EMV training starts with histories of length one and then increases maximum history length to two after eight epochs, to three after epoch 12 and to five after epoch 16. In the present thesis, the above procedure is improved by simplifying it in the following points:

First, in view of the negative results of Bärman et al. (2021) regarding usefulness of image data as present in the current form of the EMV dataset, the image modality is neglected for most of the experiments (if used, it is explicitly mentioned). Thus, the dimensionality reduction on the latent image data is also abandoned. Currently, ongoing work is concerned with improving the robot simulation to produce photo-realistic images, which then might be more useful. Exploring this aspect is left for future work.

Second, early experiments on both the *previous* as well as the *short dialogs*<sup>3</sup> dataset indicated that the EAE pretraining mentioned above hurts performance. Fig. 5.1 shows the results of these experiments with the EMV model using the CT EM encoder on the *short dialogs* dataset. It compares the percentage of correct answers between one model with EM encoder and decoder initialized randomly, one with these components initialized from EAE pretraining, one using the pretrained parameters and additionally freezing them for the first two epochs, and the text-only baseline. It is clearly visible that the model without

<sup>3</sup>As explained in section 5.1.2, the *short dialogs* set is based on the new, more diverse set of robot episode recordings and is a subset of the *long-term dialogs* dataset used in the following section.

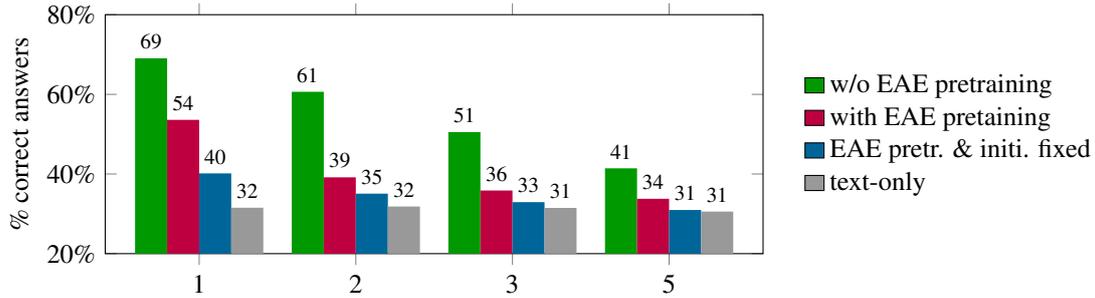


Figure 5.1.: Percentage of correct answers by history length on the *short dialogs* test set, exploring the usefulness of episode auto-encoder (EAE) pretraining.

EAE pretraining outperforms the other ones by a large margin. On the first look, this is contra-intuitive since Bärmann et al. (2021) introduced the EAE pretraining with the intention of learning a useful compact representation of the multi-modal input stream on the one hand, and increasing date-time diversity by frequent regeneration of the training data during EAE pretraining on the other hand. However, the results indicate that the pure auto-encoder goal contradicts the loss coming from verbalization. Hence, this thesis does not use EAE pretraining in all of the following experiments. Nevertheless, Bärmann et al. (2021) already showed that using  $\mathcal{L}_{\text{EAE}}$  as a contribution to the overall loss during EMV training provides a notable performance benefit and thus, this is adopted without modification. To summarize, separate pretraining of the EAE is dropped, while the additive loss contribution  $\mathcal{L}_{\text{EAE}}$  is still used during EMV training.

Third, regarding the curriculum learning strategy with respect to the history length, the ablation studies of the preceding work strongly support its usefulness. While keeping the general idea of curriculum learning, the training schedule is modified due to observations made during the early experiments mentioned above. Instead of working with the epoch counter to control the process, the number of optimizer updates is used to select the point of time for switching the curriculum stage (i. e. increasing the maximum history length). This has two reasons: First, due to the increased size of the new datasets (compared to *previous*), training for multiple epochs would take an unacceptable amount of time. Second, an analysis of the learning curves of all the models trained during the above experiments revealed that they converge early and most of the training effort is wasted. Thus, taking these observations into account, the schedule was modified for all of the following experiments, training on histories of length one for 50K optimizer updates, maximum length two until update 80K, three until 100K, and five until 115K<sup>4</sup>.

For some model architectures, multiple experiments with configurations differing in learning rate schedule or other hyperparameters were performed. In a few cases where the loss function diverged at the beginning of training<sup>5</sup>, exactly the same configuration was also ran multiple times<sup>6</sup>. To stay with a valid methodology, model selection among these slightly different or identically configured runs was performed using results on the validation set. In particular, after each 5K epochs of training, semantic score (i. e. the percentage of correct answers determined using a fast approximation of the heuristic evaluation script introduced above) was determined on the validation set. Then, among a set of similar or identical configurations, the one with the highest maximum validation semantic score was chosen. Results reported in the following section were produced on the test set by the last model checkpoint of these configurations.

<sup>4</sup>Since the actual number of used training samples also depends on the batch size, this must be mentioned here for completeness: histories of length 1 use a batch size of 96, two 64, three 48 and five 32. The batch size is reduced with the number of episodes because longer histories require more GPU memory.

<sup>5</sup>See appendix B for more details about learning curves during EMV training.

<sup>6</sup>Identically configured runs may – although the random number generator seed is fixed and the runs were performed on the same hardware – nevertheless behave differently because there are multiple other sources of nondeterminism. Particularly, GPU-optimized functions using `atomic_add` provide no guarantees about execution order, thus leading to different results when used with floating-point arithmetic. The EMV model includes such operations, e. g. by using an embedding bag for the episode goal.

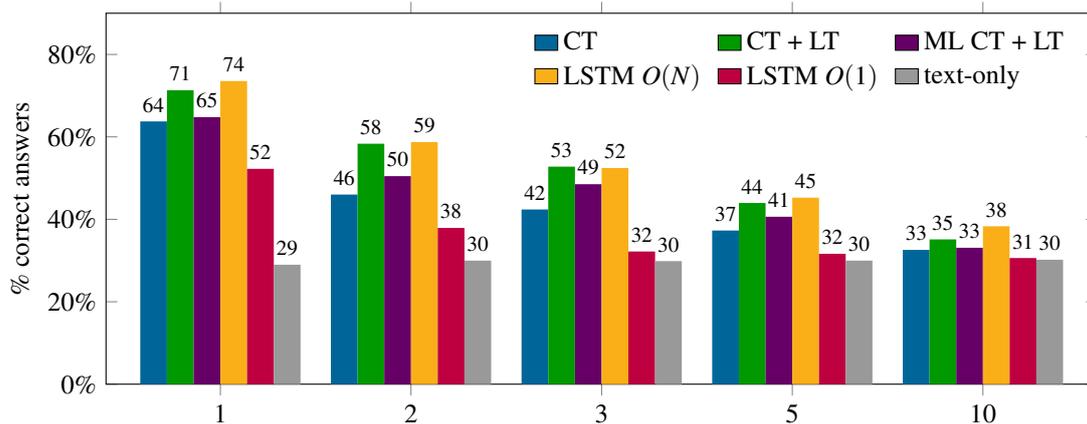


Figure 5.2.: Percentage of correct answers by history length on the *long-term dialogs* test set.

## 5.4. Results

The EMV model was trained and evaluated using various EM encoder alternatives as described in section 4.3, as well as using different datasets as explained in section 5.1.2. In the following, the results of these experiments are presented:

**long-term dialogs** To analyze the capabilities of the newly introduced  $O(1)$  EM models, the different architectures were evaluated in the EMV task on the test split of the *long-term dialogs* dataset. Fig. 5.2 compares the percentage of correct answers over the history length among EMV models with the EM implementations compressive transformer (CT), CT with long-term memory (CT + LT), multi-level CT with LT memory (ML CT + LT), as well as the baseline models LSTM  $O(1)$ , LSTM  $O(N)$  and text-only. Since the text-only baseline does not have access to the episode data at all, its performance is independent of the length of the history used for evaluation. With no chance to know what the robot actually did, it accounts for the biases present in the dataset, i. e. it serves for assessing how many questions can be answered correctly solely by educated guessing. Thus, the gray bars in Fig. 5.2 provide a lower bound to the performance of the episode-informed models.

Similarly, the LSTM  $O(N)$  model can be seen as a soft upper bound to the models with a tighter restriction concerning the size of EM. Assuming the speech decoder performs similarly across EM implementations in its task to retrieve information from EM given an encoded query, the LSTM  $O(N)$  model obviously has an advantage because it is not forced to forget any of its inputs. The upper bound is only soft, since ideally the size constraint in  $O(1)$  EM models might also lead to more relevant information being selected, thus actually easing the information retrieval task of the speech decoder. However, the results show that this benefit is not properly exploited, and the LSTM  $O(N)$  model outperforms the size-constrained EM architectures.

To enable a fair competition for the compressive transformer EM implementations, the LSTM  $O(1)$  serves as a simple baseline for such a size-constrained EM. For the sake of comparability, the number  $M$  of LSTM states to keep is set equal to the total number of EM vectors in the (ML) CT + LT EM configurations. The LSTM  $O(1)$  performance is notably worse than that of LSTM  $O(N)$ , and approaches the text-only results for history length three and above. In particular, the ability of the LSTM to retain useful information does not extend beyond a length of two episodes. Moreover, just keeping the latest LSTM states and thereby having no dedicated mechanism of information selection visibly limits the performance even for short histories.

Contrary, the compression functions of the different CT architectures explicitly train to filter relevant information using their separate compression loss. The usefulness of this approach can be clearly seen in the results, as all of them perform notably better than the LSTM  $O(1)$  baseline. For instance, looking

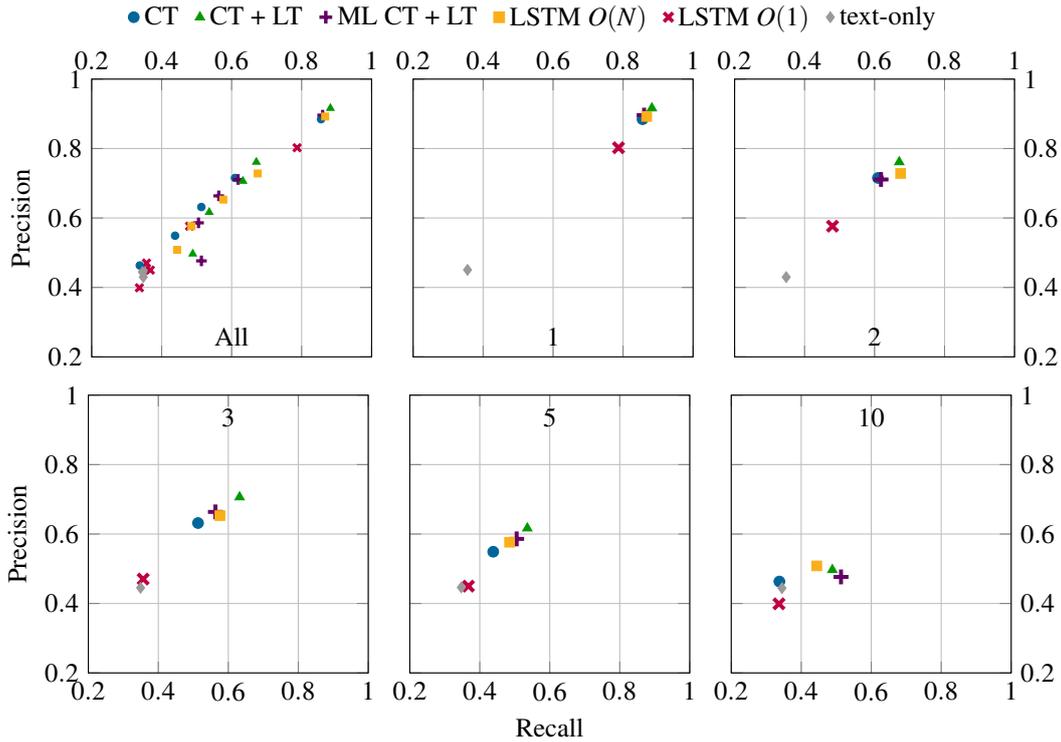


Figure 5.3.: Information precision and recall of evaluating the different EM architectures on the *long-term dialogs* test set using histories of length one to ten. The “All” plot, shows all history lengths at once.

at the plain CT EM model on short histories of one or two episodes, there is an improvement of 12 % resp. 8 % of correct answers compared to the LSTM  $O(1)$  baseline. However, as expected due to the limited context length of the plain CT model (as explained in section 4.3.2), there is a huge decrease of correct answers with increasing history length, since this model is only able to remember the most recent information.

With the purpose of removing that limitation, the CT + LT introduces its long-term memory component. Because of that, the expectation is for it to be able to keep its performance on a high level even with increasing history length. On the one hand, when looking at the results, this expectation is partly met by the CT + LT model consistently outperforming the plain CT model for all history lengths. The higher percentage of correct answers even for short histories can be explained by the variations in episode length, i. e. there are some complex episodes in the dataset that on their own are already longer than the context length of the plain CT model. On the other hand, the CT + LT model still suffers from a dramatic decline of performance for longer histories, although less than the other  $O(1)$  models. This sheds light on two central issues: First, constraining the EM to a constant size by definition requires the model to forget some less relevant details about the past. Thus, with the dataset containing questions which ask for details of episodes long ago, a moderately lower performance compared to the  $O(N)$  model is unavoidable. Second, the same dramatic decline of correct answer percentage with increasing history length can also be observed for the LSTM  $O(N)$  model. Since it is not limited by the size of EM, the constraining factor here must be the retrieval of information from EM which is performed by the speech decoder. In summary, this shows that the addition of LT memory to the CT architecture successfully serves its purpose, while there is still room for improvement concerning the mechanisms for accessing EM content.

The last architecture to discuss in Fig. 5.2 is the ML CT + LT. Here, a further improvement over

CT + LT was expected, since multiple compression levels might lead to better abstraction capabilities. For instance, if the (plain) memory remembers information on the key frame level, level-1 compressed memory can capture individual action steps, level 2 complete actions, and level 3 (and the LT memory) complete tasks. Unfortunately, the results indicate this expectation is not met. While outperforming the plain CT model, ML CT + LT is not able to compete with its single-level companion. One possible explanation is that the dataset in its present form does not support such abstraction, as mentioned with the detailed questions about far episodes above. Furthermore, the ML CT + LT architecture turned out to be quite sensitive concerning the learning rate and hyperparameter settings, quickly causing a model to diverge early during training. Thus, it is a challenge for future work to both extend the dataset to better support semantic abstraction and solve the optimization problems coming with the ML CT + LT model.

As explained in section 5.2, looking solely at the number of completely correct answers can only provide a partial view of a model’s overall performance. Therefore, Fig. 5.3 shows the information precision and recall from the same experiments as above, also separated by history length. For all models (except for the text-only baseline), a clear trend with increasing history length from the upper right to the lower left corner of the PR diagram can be observed. Furthermore, the plot clearly reveals which models solely guess based on the biases in the dataset: the data points in the lower left of the “All” plot, where all the text-only points lie. It can be seen that – as expected – the CT model for history length ten as well as the LSTM  $O(1)$  for histories of length three and above fall into this category. On the other hand, all the other architectures are clearly separated from that guessing area, even for the longest history length. Additionally, although not significant, the LT-memory-enhanced models tend to have a higher recall than the other  $O(1)$  models at each respective history length. Compared to the LSTM  $O(N)$  baseline, the CT + LT model performs similar, and for one, three and five episodes even better in both precision and recall. While the benefit of being required to compress and filter input information did not manifest itself in the number of completely correct answers, this can be seen as an indicator thereof.

**previous** To enable a comparison of the new models and methods with the existing work of Bärman et al. (2021), the evaluation was also performed on the test split of the *previous* dataset as presented in section 5.1.2. Fig. 5.4 compares all EM architectures on this dataset, additionally showing “previous work” numbers referring to the “without images” results reported by Bärman et al. (2021). This model essentially is the LSTM  $O(N)$  without the improvements in episode encoding and embedding introduced in section 4.2. Notably, the direct comparison of LSTM  $O(N)$  with “previous work” reveals that these minor modifications have a huge impact on the model’s performance. This is reasonable, since the previous encoding implicitly defined an unintended order on the symbolic tokens by not using one-hot encoding properly. Concerning the text-only baseline, two things can be noticed: First, the reported percentage of correct answers is slightly higher (2 %) than that of Bärman et al. (2021), which is explained by minor improvements in the heuristic evaluation script. Second, although guessing an answer in context of the more diverse *long-term dialogs* dataset should intuitively be harder, the text-only baseline has lower performance on the easier *previous* data. This relates to the differences between the two datasets regarding the frequency of the various question types and is further elaborated in the appendix (see Fig. A.3).

Nevertheless, the *previous* dataset’s low diversity manifests in the performance of the other models. Here, the numbers are notably higher than for the *long-term dialogs* results, confirming the task in this dataset is actually easier to solve. Similarly, this is a possible explanation of why the CT model, despite its limited context length, outperforms most other models on this dataset. Due to the simpler task – and shorter episodes in particular – the model is able to keep a higher performance than on the *long-term dialogs* test set while using the same context length. With that in mind, the long-term memory appears to not get properly utilized in this setting. Only for histories of length ten, the CT + LT performs marginally better than the plain CT model. Looking at the ML CT + LT, it suffers from the same problems as on the *long-term dialogs* set, thus consistently performs worse than its single-level counterpart.

Despite the problems described above, it should be stated that all the newly introduced  $O(1)$  models clearly outperform the LSTM  $O(1)$  baseline (which is the one to do fair comparisons with). Particularly, for history length three and above, the margin is very expressed, supporting the usefulness of the presented model architectures.

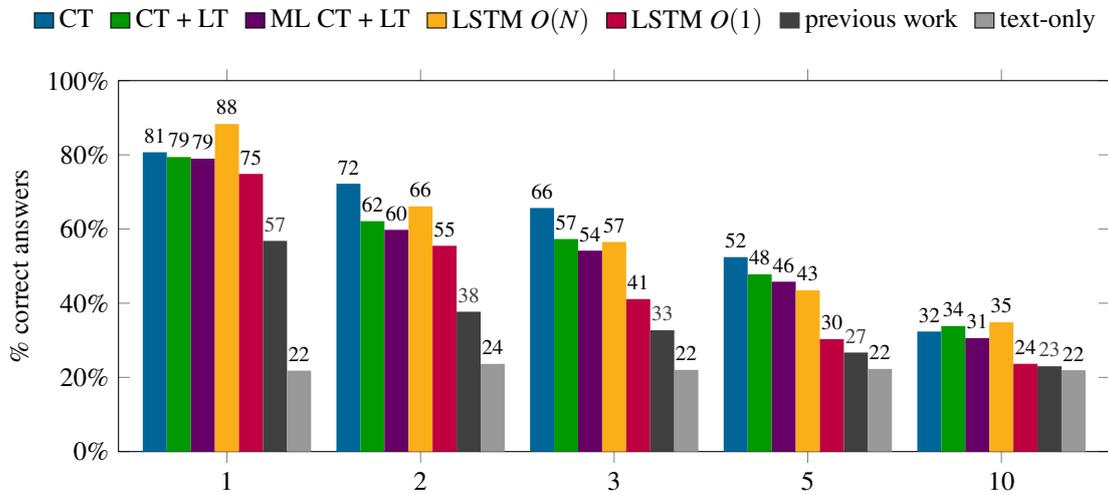


Figure 5.4.: Percentage of correct answers by history length on the *previous* test set. The “previous work” numbers refer to the “without images” results reported by Bärman et al. (2021).

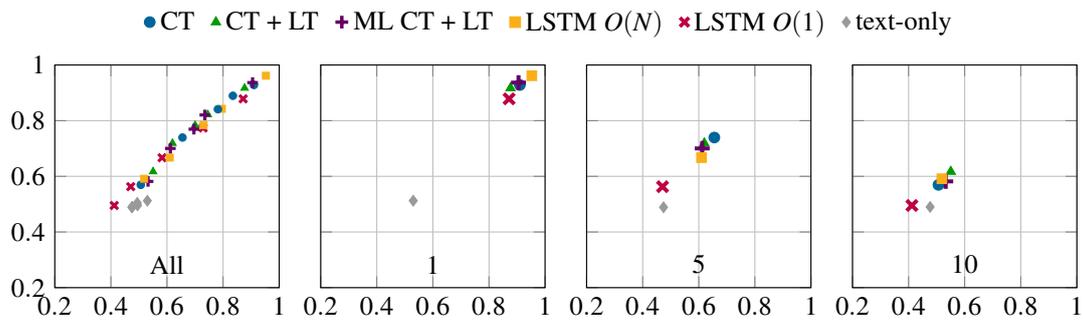


Figure 5.5.: Information precision (y) and recall (x) of evaluating the different EM architectures on the *previous* test set using histories of length one to ten. The “All” plot, shows all history lengths (1, 2, 3, 5, 10) at once.

Comparing the PR curves for the *previous* dataset shown in Fig. 5.5 to that in Fig. 5.3 further confirms that the new tasks are much harder to solve, as both information precision and recall of the text-only model on the *previous* set are about 10 % higher than on *long-term dialogs*. Due to the easier task, the *previous* results do not have the guessing area as clearly separated from the other models as for *long-term dialogs*. Notably, despite its EM size constraint, the CT + LT model outperforms the LSTM  $O(N)$  in both precision and recall on histories of length ten.

**Analysis by question type** To account for the grammar-generated test data and to analyze limitations of this procedure, it is beneficial to evaluate the results separately for the different types of questions as defined in section 5.1.2. Thereby, two aspects can be observed: On the one hand, it is interesting to see how well the models are able to handle individual question types. Particularly, this may shed some light on what tasks a model can solve and where it struggles. On the other hand, especially when inspecting the performance of the text-only model, evaluating by question type effectively is also an analysis of the dataset itself, allowing to find data biases or imbalances.

Table 5.3 shows the details of this analysis on both the *long-term dialogs* and *previous* datasets. For each question type, it compares the performance of the CT + LT model (as this is the best  $O(1)$  model in the experiments presented above) with the text-only baseline. The numbers indicate the percentage of correct answers with regard to the total number of questions of this type, evaluated on histories of different lengths (from one to ten).

On the first look, three question types with very high scores both for the CT + LT as well as the text-only baseline stand out: Concerning the *Current date & time* questions, high numbers are expected as this is the easiest task because the answer (i. e. the current date or time) is already encoded in the query (i.e the question plus timestamp). Thus, in theory both models could easily reach a perfect score here. However, for the other two high-scoring types *Problems* and *Duration*, the high guessing performance indicates biases in the generated dataset.

For *Problems* questions asking if the robot had any failures during a specific time or action, the problem is grounded in an intrinsic dataset imbalance. Naturally, execution problems are rare compared to the number of actions which succeed. Indeed, an analysis of the *long-term dialogs* test set with history length five shows that there are 71 failed in contrast to 2478 successful actions (2.8 %), and correspondingly, the dialog generation process leads to only 2.6 % of all *Problems* answers actually mentioning any failures. This perfectly fits to the text-only model having a question-type specific correct answer score of 97.4 % on this data, i. e. it simply always states there were no problems and performs well with this strategy in nearly all cases. Unfortunately, there is no difference between the text-only baseline’s and the CT + LT model’s performance on the *Problems* questions, i. e. the episode-informed model does not actually learn anything useful about execution failures but overfits identically to the guessing baseline.

Similarly, the results on the *Duration* questions also indicate biases present in the dataset, leading to a better performance than actually using the episode data and thus preventing the episode-informed model from learning more than the text-only baseline does. In this case, however, the imbalance is not that obvious, as there is a more diverse spectrum of possible answers to *Duration* questions: Although there is one highly overrepresented answer (“It took about one minute” makes up 42 % of all *Duration* answers referring to the same test set as above), it does not explain the performance of about 90 %. Consequently, this means the model is able to learn some correlation between the actions referred to in the question and the corresponding answer. For instance, the action “move to the round table” might in most cases take about ten seconds. Thus, the model can learn this correlation, and answer the question “How long did it take to move to the round table yesterday?” correctly with a high chance without even looking at the content of the episode.

In contrast to that, for the question types *Action*, *Object*, *Location* and *Past time*, an improvement of the CT + LT model over the baseline can be observed. Especially for *Action* questions like “What did you do yesterday in the morning?”, there is a huge gap between the episode-informed model and the guessing baseline, which basically always fails. Due to the very limited set of actions in the *previous* dataset, the guessing performance is slightly higher here, but the same benefit applies to the CT + LT model, leading to an even more expressed gap. Nevertheless, both datasets show a strong decrease in the number of

Table 5.3.: Percentage of correct answers, by history length and question type, on the *long-term dialogs* and the *previous* dataset. Question types are as defined in section 5.1.2. *DT* = *Current date & time*, C = CT + LT model, T = text-only model

		<i>long-term dialogs</i>					<i>previous</i>				
		1	2	3	5	10	1	2	3	5	10
<i>DT</i>	C	99	100	96	99	97	99	100	99	99	99
	T	92	91	94	92	85	95	98	97	99	98
<i>Problems</i>	C	95	97	98	97	98	92	92	92	92	93
	T	95	97	98	97	98	92	92	92	92	93
<i>Duration</i>	C	91	92	90	91	91	90	90	91	92	92
	T	90	92	91	92	91	91	92	92	92	93
<i>Action</i>	C	73	51	40	24	7	84	61	54	40	19
	T	0	1	0	0	1	3	4	2	2	2
<i>Object</i>	C	73	65	54	42	25	99	94	94	86	79
	T	11	12	10	11	11	66	69	65	65	65
<i>Location</i>	C	42	37	34	33	34	47	32	37	35	33
	T	20	22	22	20	20	27	35	29	28	31
<i>Past time</i>	C	14	4	3	2	1	20	6	4	3	1
	T	3	1	1	1	0	2	2	1	0	0
<i>How often</i>	C	56	58	58	51	41	-				
	T	63	67	60	56	45	-				
<i>Summary</i>	C	75	39	43	46	37	-				
	T	23	23	25	25	27	-				
total	C	71	58	53	44	35	79	62	57	48	34
	T	29	30	30	30	30	22	24	22	22	22

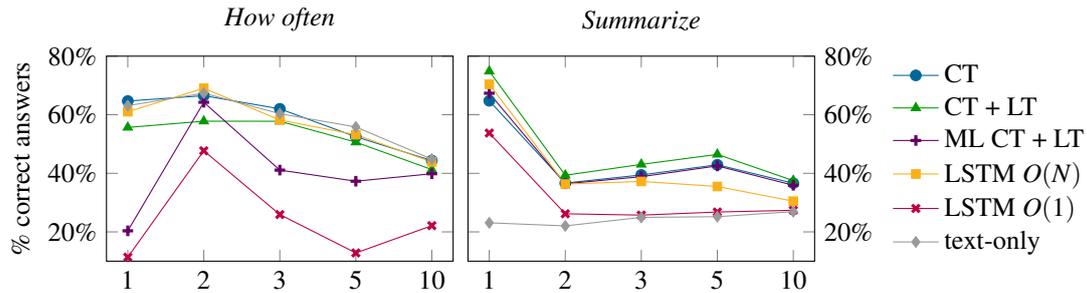


Figure 5.6.: Percentage of correct answers by history length, evaluated on the two question types specifically added to the *long-term dialogs* dataset.

correct answers on this question type with increasing history length. Similar observations can be made when looking at the *Object* question type, where the user asks about which object the robot acted on at a particular point in time. Here, next to the CT + LT model clearly outperforming the text-only baseline, and despite the decline in performance with increasing history length, the differences in dataset diversity express themselves even more explicitly in the number of answers correctly guessed. While it is at about 11 % for the *long-term dialogs* set, guessing solves about 65 % of such questions on the *previous* dataset.

Moving on to the *Location* (“Where did you put ... ?”) and *Past time* (“When did you ... ?”) question type, the performance gap between episode-informed and text-only model further decreases. While the CT + LT model consistently outperforms the guessing baseline on the new, more diverse dataset, no clear trend is visible on the *previous* test data concerning the *Location* questions. Furthermore, the *Past time* task appears to be too hard to solve for the current model architecture. Although better than guessing, the episode-informed models reach only very low percentages of correct answers. This indicates a huge room for improvement concerning the handling of date and time information in the overall architecture.

The last two question types to analyze are those introduced in the *long-term dialogs* set to specifically facilitate and inspect long-term learning capabilities of the presented models. Here, the results are twofold: For the *How often* dialogs, there appears to be a dataset bias problem similar to the *Duration* questions. On the one hand, “I did/moved ... only once” is a highly overrepresented answer (51 %), on the other hand, there are correlations between question and answer, as certain locations are more frequently occurring in the simulated recording than others. Interestingly, as shown in Fig. 5.6, text-only guessing actually performs better than most of the episode-informed models including the CT + LT on this question type. Analyzing the histogram of model outputs for *How often* question for the text-only as well as the CT + LT model (as shown in Fig. A.2) reveals that – while the text-only model heavily overfits on the most probable “only once” answers – the CT + LT model shows more diversity in its answer, trying to detect “never” and “two times” cases. However, the attempt to solve the task more adequately all in all leads to worse results than blind guessing as done by the text-only model.

Contrary, the episode-informed model seems to be able to learn useful, task-specific skills for answering the *Summarize* questions. The high baseline performance is induced by the frequent “I did not do anything at this time” type of answers. Interestingly, the CT model and its variants outperform not only the text-only baseline, but also the LSTM  $O(N)$  model on histories of length five and ten (see Fig. 5.6). This might be indicating that compression is actually superior to keeping all information for this type of task.

**Analysis by reference episode** All analyses so far were differentiated by the length of the history used to evaluate the model. This strategy is useful to determine how well a given model is capable of handling a stream of information as it increases in length. However, it does not account for which part of the (potentially long) input stream the question actually refers to. For instance, answering to “What did you do five minutes ago?” does not require any knowledge about the episodes before that, and thus

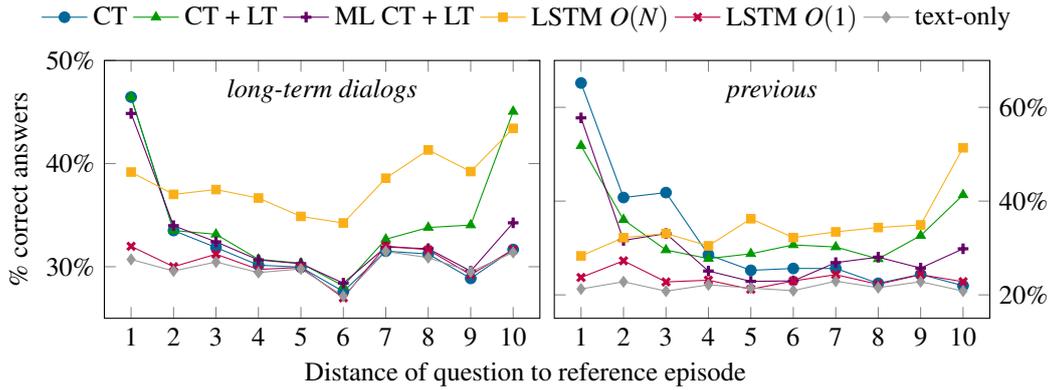


Figure 5.7.: Percentage of correct answers on the *long-term dialogs* (left) and *previous* (right) test data with history length ten, evaluated by distance to the episode referred to in the question.

is in theory independent of the history length used for evaluation<sup>7</sup>. Therefore, the following section will deal with an analysis of performance differentiated by the distance between the question and the episode it refers to. Ground-truth reference episodes are available because the data-generating grammar already stores these time references for purposes of the time reference loss explained in section 4.5. Thus, the distance between a question and its reference episode can easily be determined in the heuristic evaluation script by inspecting this target time reference, the question timestamp, and the history of episodes. However, there are some types of questions not referring to any specific episode, particularly some *Problems* questions asking for failures in a specific time interval, all *Current date & time* questions, as well as both question types newly introduced for the *long-term dialogs* dataset. Consequently, these are excluded from the following analyses.

Fig. 5.7 shows the results of evaluating the EMV model with the various EM architectures on the *long-term dialogs* and *previous* test set. Thereby, the history length is fixed to ten episodes, since in this setting there are the most possible reference distances. On both datasets, some interesting observations can be made: First, the LSTM  $O(1)$  barely produces more correct answers than the text-only baseline, independent of the distance to the question’s reference episode. This is as expected, since the results from above already indicated that this model completely falls back to guessing for long histories. Moreover, the LSTM  $O(N)$  does not show a clear trend of performing better or worse with increasing distance to the question’s reference, as expected for the unconstrained memory. However, for the *previous* data, there is a peak towards the first input episode (i. e. the one farthest away from the question), which is not explained easily. A possible reason might be that this LSTM learned to be biased towards its first input data.

Besides that, the limited context length of the plain CT model is clearly visible, as it outperforms the text-only baseline only up to reference distance three or four, depending on dataset complexity. The same effect is visible for the enhanced CT variants, as they only augment the EM as produced by the plain CT. However, in addition to the peak for small reference distances, the CT + LT also shows a peak for the farthest input data (i. e. distance ten). This means, the long-term memory appears to be used mainly for remembering the initial inputs to the EM. Indeed, the design of LT memory as presented in section 4.3.2 might facilitate this effect, as attention in Eq. (4.3) is conditioned on the old LT state, thus selecting similar content only. To further investigate this, another variant of the CT + LT model is examined, with a learned query vector for producing the LT update candidate (i. e. replacing  $l_{t-1}^k$  in Eq. (4.3) with a learned, layer-dependent vector  $q_k$ ). Results of evaluating this model are shown as “CT + LT learned” in Fig. 5.8. It can be seen that this architecture change does not improve performance, and still shows the peak for the farthest input data, while producing poor results in the middle. Thus, this peak might be

<sup>7</sup>Of course, in practice history length still matters, as a poorly performing model may choose to collapse for longer input streams and resort solely to guessing, as illustrated by the analysis of the LSTM  $O(1)$  results on histories of length  $\geq 3$  in Fig. 5.3.

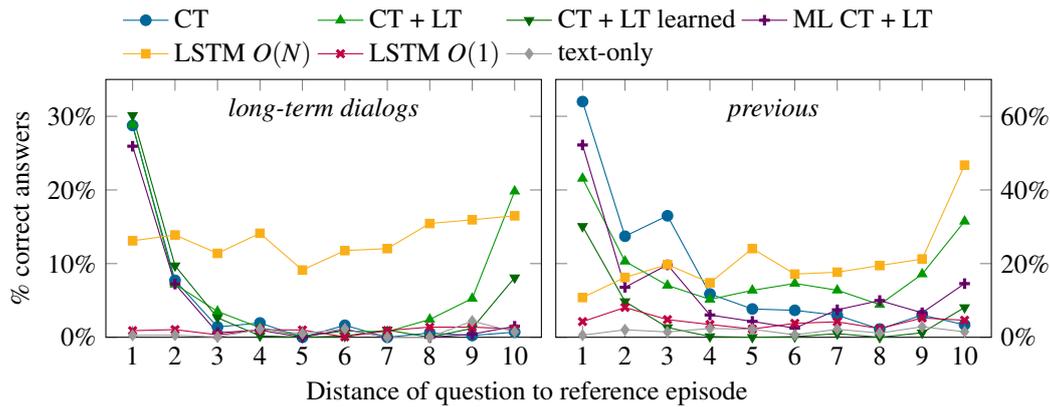


Figure 5.8.: Percentage of correct answers by distance to reference episode, on the *Action* questions of the *long-term dialogs* (left) and *previous* (right) test data with history length ten.

induced by the GRU, and not the attention mechanism. Also using a GRU cell, the same effect should theoretically be observable for the ML CT + LT model too, however, this is only marginally visible and probably prevented by the optimization and dataset issues elaborated on earlier.

On top of that, the results can also be viewed separately by both question type and reference episode distance. Accounting for the fact that answers to some types of questions are easily guessed, the already mentioned Fig. 5.8 presents the results by reference distance on the *Action* data only. This is the question type where the text-only performance is worst and the gap to the episode-informed models is the largest. Consequently, analyzing and comparing the introduced EM models on this question type removes unwanted effects induced by the grammar-generated utterances. Thus, Fig. 5.8 provides a clearer view on the analyses made above. Particularly, it confirms the effect of the CT + LT model performing best on the closest and the farthest episode data, i. e. the newest and the oldest content of EM. Additionally, the marginal peak of the ML CT + LT for the oldest episode data is slightly better visible, especially on the *previous* data.

**Modality ablation study** While all the experiments presented so far did ignore the image data part of each episodic key frame, the following paragraph compares results of models trained on different subsets of the available modalities. As outlined in section 4.2, each key frame contains multi-modal information of the following types: Date and time information, symbolic execution data, subsymbolic sensor data, and a latent representation of the current image from the robot’s camera. Since timestamps are crucial for solving the EMV task, they are included in all ablation experiments. Fig. 5.9 compares the percentage of correct answers produced by the CT + LT model<sup>8</sup>, trained on the *long-term dialogs* set using different sets of modalities. The “symb + subsymb” results are produced with access to the same modalities as all experiments above (i. e., these data stem from the same run as the “CT + LT learned” results in Fig. 5.8), and thus provide the baseline for comparison. Additionally giving the model access to latent image representations does not produce a notable difference, as shown by the “symb + subsymb + latent” results. Similarly, using symbolic data only neither has a significant impact. In contrast to that, dispensing with symbolic data and relying solely on subsymbolic and latent data causes the model to struggle, notably dropping performance. Nevertheless, results on shorter episodes are still above the text-only baseline by a fair margin. To further investigate whether this effect is caused by the latent images or by the subsymbolic data, these two modalities are also experimented with on their own. The results show that, while “latent” performs comparable to “subsymb + latent”, the “subsymb” only run completely falls back to text-only guessing.

To summarize, for the current dataset and model architecture, the symbolic data mainly drives performance in the EMV task. This is reasonable, as the grammar for generating questions and answers uses

<sup>8</sup>including the learned LT update query vector, as introduced above.

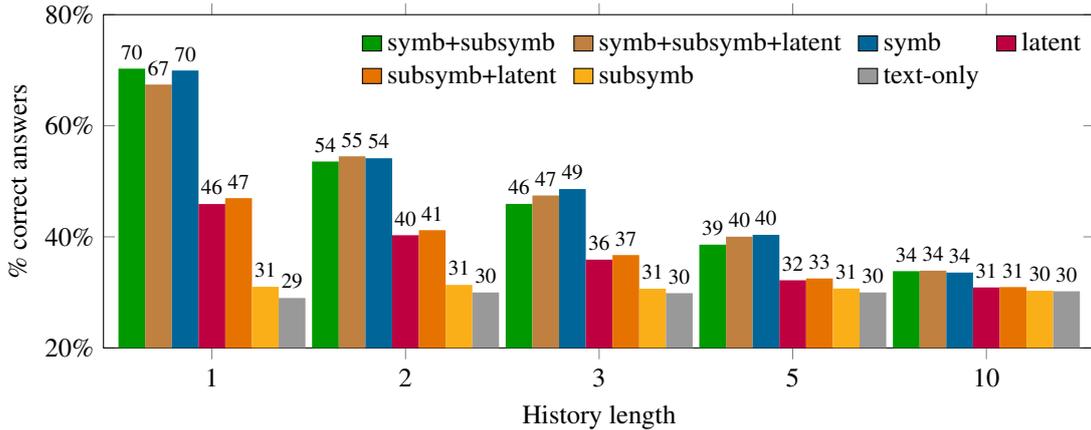


Figure 5.9.: Percentage of correct answers over history lengths, evaluated on the *long-term dialogs* dataset with access to different sets of episode modalities.

the symbolic data to decide what utterances to generate. Interestingly, the model is to some extent able to infer relations of episode content and image data. Contrary, the subsymbolic data is – in its current form – mostly useless to the model.

This is in line with the findings of Bärman et al. (2021). However, a small improvement concerning performance without access to symbolic data can be observed when comparing models trained on the *previous* (7.8 % correct *Action* questions for “subsyb + latent” vs. 3.2 % for text-only on histories of length one) and *long-term dialogs* (27.3 % “subsyb + latent” vs. 0.2 % text-only) dataset. This can be explained by the new dataset being more diverse and significantly larger, providing far more training images and hence leading to a better performance. For future work, it indicates that using a larger and more realistic set of images is a promising direction.

**Generalization study** Using the generalization test data from Bärman et al. (2021) as presented in section 5.1.3, the model’s adaption capability to natural language variations and real robot recordings is assessed. Human evaluation results of answers to *simulated-robot-human-annotated* questions as produced by the CT + LT model, once trained on the *previous* and once on the *long-term dialogs* set, are compared to the performance of the model in the preceding work in Fig. 5.10a. While there is still a huge amount of inappropriate and wrong answers, two observations can be made: First, the architecture changes, especially the new episodic input encoding (section 4.2) as well as introduction of the compressive transformer, clearly have a positive effect, as these are the major differences between the model of Bärman et al. and the CT + LT model trained on the *previous* dataset. Second, the increased diversity of the simulated episodes in the *long-term dialogs* set also aids with generalization on the language level, as the model trained on this set performs slightly better than the other ones, with more correct and less inappropriate answers.

Furthermore, experiments with episode recordings from a real robot system were performed. Using the *real-robot-grammar-generated* test set shows improved results over the previous work, too, as shown in Fig. 5.10b. Most of the remaining wrong answers stem from the difficult question types as already discussed above. In particular, the model fails about 90 % of *Past time* questions.

Finally, the *real-robot-human-annotated* set jointly evaluates both generalization capabilities. Despite the small number of samples, and although more than halve of the answers are still wrong or inappropriate, both the new model as well as training on the new dataset delivers some performance gains over the previous work (see Fig. 5.10c). For the interested reader, Table C.2 provides exemplary model responses on the discussed generalization test data.

To summarize, the main observations of the generalization study coincide with that of Bärman et al. (2021): While the transfer from simulated to real robot recordings is not a huge problem (which is

reasonable, because the included symbolic episode data already exhibit a great level of abstraction), the EMV model has severe difficulties to generalize to natural language variations. Nevertheless, this thesis improves over the previous work, despite not changing the speech encoder or decoder.

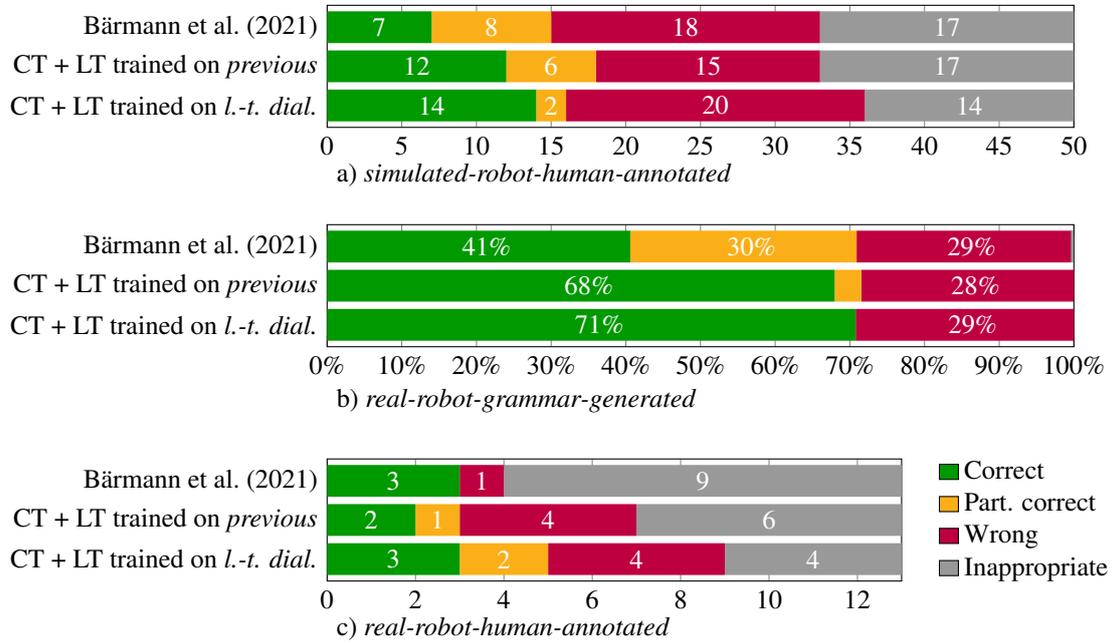


Figure 5.10.: Results of the different generalization experiments. Statistics on the human-annotated sets are in absolute numbers, while performance on grammar-generated questions is reported in percent (the total number of questions in the *real-robot-grammar-generated* set is 12913). The first bar in each plot refers to the model of Bärmann et al. (2021, Fig. 6).

## 6. Conclusion and Future Work

In this thesis, an architecture for episodic memory verbalization (EMV) is presented, including an EM encoder featuring constant memory space and query-time computation requirements. Building on the foundation work of Bärman et al. (2021), the following contributions are made: First, compared to the previous work, a far larger and more diverse dataset of robot experiences is collected. Second, the EMV architecture is enhanced at several points. In particular, for the EM encoder, the Long-Term Compressive Memory Transformer (CT + LT) model, and a multi-level variant thereof, are proposed. By providing both unlimited context length and constraining the EM to a constant size, it solves the conceptual limitations of the EM in previous work.

Utilizing the dataset from previous work as well as the newly collected one, the proposed models are trained and evaluated with respect to their ability to answer questions correctly in the EMV task. The results show that the CT + LT model outperforms both the text-only as well as the LSTM  $O(1)$  baseline by a huge margin on both datasets, thus confirming the architectural choices. Moreover, the results of Bärman et al. (2021) are topped significantly. The CT + LT model even competes well with the LSTM  $O(N)$ , although this comparison is theoretically unfair as the latter has no constraints on its EM size.

Nevertheless, the EMV model’s performance still decreases rapidly with increasing history length. Summarizing the analyses both by history length and by reference episode, two reasons for this can be identified: First, the EM information retrieval performed by the speech decoder appears to be a bottleneck, as the decreasing performance also affects the unconstrained EM of the LSTM  $O(N)$  model. Second, the compressive transformer architecture can by design only remember its last experiences, while the experiments show that adding the long-term memory component mainly keeps information from its initial inputs instead of aggregating experiences over time. In summary, this leads to a low performance on questions about events “in the middle” of a history. Although the decline of correct answers with increasing number of input episodes is severe, reviewing the information precision and recall scores indicates that, even for long histories, the proposed models do not completely collapse to text-only guessing.

A further analysis of the results on the individual question types generated by the regular grammar for QA annotation revealed a set of limitations induced by the dataset generation procedure. On the one hand, certain events, especially execution failures, are underrepresented in the simulated experience data, and thus all models ignore these rare samples during training. On the other hand, there are types of dialogs in the generated data where even the text-only model can achieve a very high percentage of correct answers by leveraging strong correlations between questions and answers, without taking the episodic context into account. While it could be considered that, when there is a lack of context, a-priori guessing of answers is also a human behavior, in backpropagation-based training, it hinders learning of the actual task, as a satisfactory performance can already be achieved without conditioning on the robot experiences. Despite these limitations, the analyses also showed that there are question types (especially *Action*) without such flaws, presenting a clearer picture on the performance of the proposed models.

To sum up, the main goal of this thesis was reached: With the Long-Term Compressive Memory Transformer, an  $O(1)$  EM is implemented and empirically validated, notably improving upon the results of the preceding work. However, generalization both to longer histories as well as to variations in natural language still is a problem, indicating a potential for future work. While collecting human-annotated QA training data is a plausible approach to improve robustness in natural language understanding, working on generalization to longer histories will most certainly require architectural modifications. In particular, the strategy of retrieving information from EM needs to be reconsidered, e. g. using a separate attention mechanism dedicated to EM access. Moreover, the model could potentially benefit from a more explicit formulation of time throughout the architecture, as finding the correct EM content given a natural language time reference is – due to the complexity of calendar algebra – a very hard-to-learn task.

As a straight-forward step for future work, the proposed multi-level compressive transformer model could be applied to the original domain of compressive transformer, which is language modeling. It might be interesting to see if multiple compression levels can facilitate long-range context awareness.

Furthermore, in the context of EMV, future work could try to broaden the used definitions to come closer to real-world applications. For instance, instead of assuming a closed-world setting and resorting to a dictionary of objects, actions and locations involved in the symbolic robot experience data, using pretrained subword embeddings to represent such tokens might lead to a more transferable system. It is also highly desirable to make more intense use of the subsymbolic and image modalities, as these senses are very unlikely to provide corrupt data (in contrast to the symbolic planner, which e. g. might not recognize an execution failure correctly). In particular, this requires improvements in dataset acquisition, producing more realistic images in simulation on the one hand and deploying a continuous data recording system to a real robot on the other hand.

As a further step, the EMV task itself could be modified, defining textual speech input to be part of the episodic data stream, and letting the model predict whether and what to respond at each time step. This could also allow a robot to start conversations proactively, for example if it meets a particular person. Concerning the EM architecture, another possible extension is to dynamically insert new long-term memory states if important or relevant events occur, comparable to human flashbulb memories (Brown et al., 1977). This might use a discrete mechanism similar to what is used in HM-LSTM or SkipRNN (Chung, Ahn, et al., 2017; Campos et al., 2018), but would need to come up with an intelligent way of how to keep the  $O(1)$  size constraint on EM. Finally, a far vision is to integrate the EMV model into a broader cognitive architecture.



# Appendices

## A. Additional Analyses

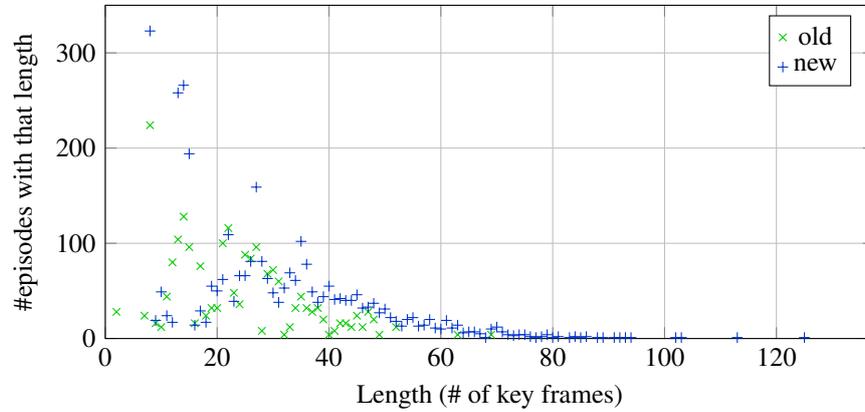


Figure A.1.: Histogram of episode length (measured by the number of key frames) in the datasets of simulated robot episodes. “old” refers to the dataset of Bärmann et al. (2021), while “new” is the one collected in the scope of this work.

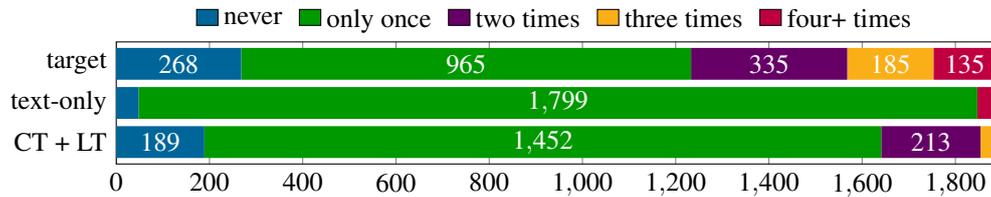


Figure A.2.: Histogram of answers to the *How often* question type, comparing the CT + LT and text-only model’s outputs to the ground truth (target) data. Evaluation was done on histories of length five in the *long-term dialogs* dataset.

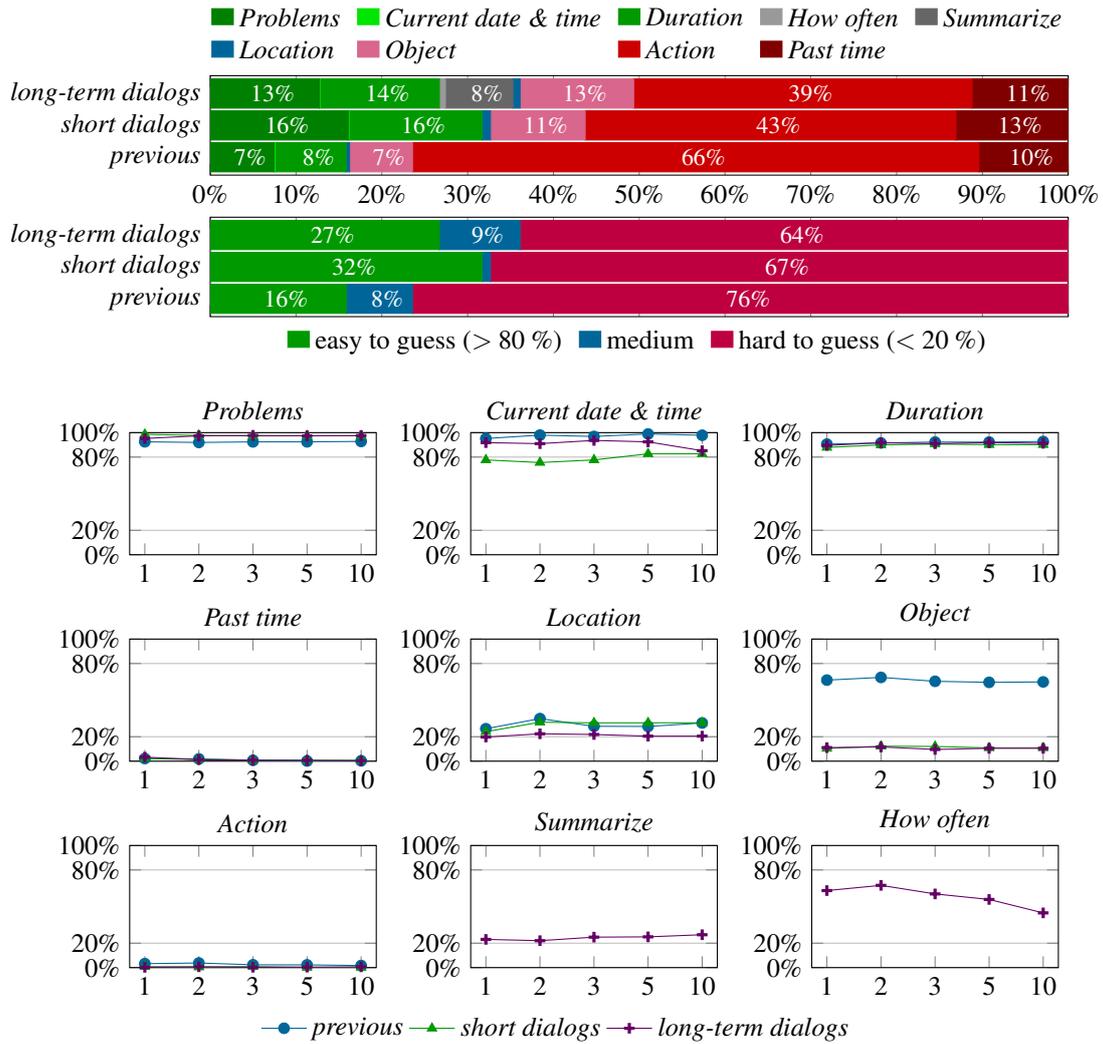


Figure A.3.: Dataset and performance analysis of the text-only baseline model. The topmost bar chart illustrates the content of the different datasets (*previous*, *short dialogs* and *long-term dialogs*), specifically the proportions of the different question types as generated for the test split on histories of length one. With the line charts in the three rows on the bottom, the percentage of correct answers produced by the text-only model on the different datasets (on histories of length one to ten) is compared. Defining a question type to be “easy to guess” if the guessing performance stays above 80%, “hard” if it is below 20% and “medium” otherwise, the question types can be categorized as follows: *Problems*, *Current date & time* and *Duration* are “easy” for all datasets. *Location* is always “medium”; for the *previous* data, *Object* is also “medium”; for the *long-term dialogs* data, the two new question types *Summarize* and *How often* are “medium”, too. The remaining question types are categorized as “hard”. With this definitions, the datasets can be analyzed with respect to their fraction of easy-, medium- and hard-to-guess questions. This analysis is presented in the second bar chart. It adequately explains the performance differences observed for the text-only model when treating the datasets as a whole (i. e. not evaluating per question type), as for all datasets, the equation  $E \leq P \leq E + M$  holds, where  $E$  and  $M$  are the fractions of “easy” and “medium” questions, respectively, and  $P$  is the percentage of correct answers produced by the text-only baseline as reported in section 5.4.

## B. Learning Curves

This section shows a few exemplary learning curves of training the different EMV models. For a complete view on all curves including interactive plots and all hyperparameter details, see

- for the *long-term dialogs* experiments:  
<https://tensorboard.dev/experiment/oZWhkSQHRKGE8pZMveXJg>
- for the *previous* experiments:  
<https://tensorboard.dev/experiment/OdbSvbAmTLeG5e2YPyA3RA>
- for the modality ablation study:  
<https://tensorboard.dev/experiment/rZ9hrvbCRjs7HJdq3w2amw>

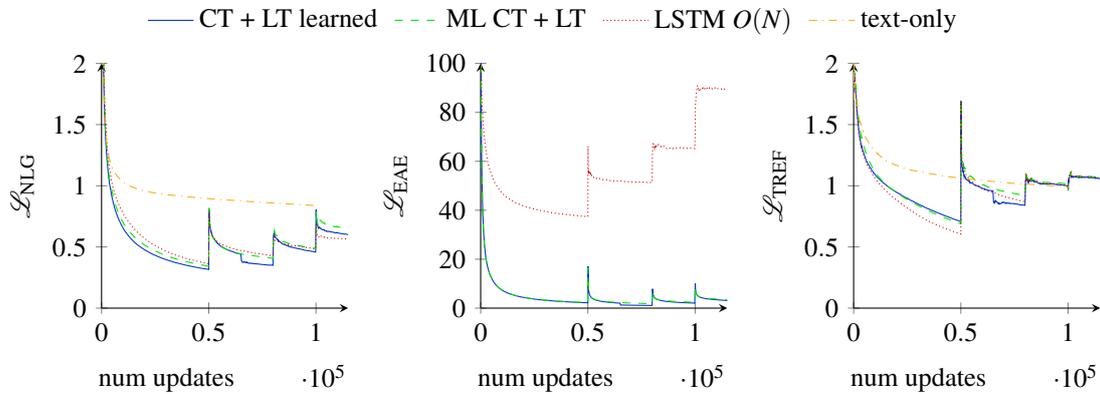


Figure B.1.: Curves of train loss contributions (answer, episode auto-encoder and time reference loss as defined in section 4.5) during training of some of the compared models. Text-only training is only performed for 100K updates (instead of 115K). The jumps of the loss function can be explained by the curriculum learning, switching to a dataset with a higher maximum history length (see section 5.3).

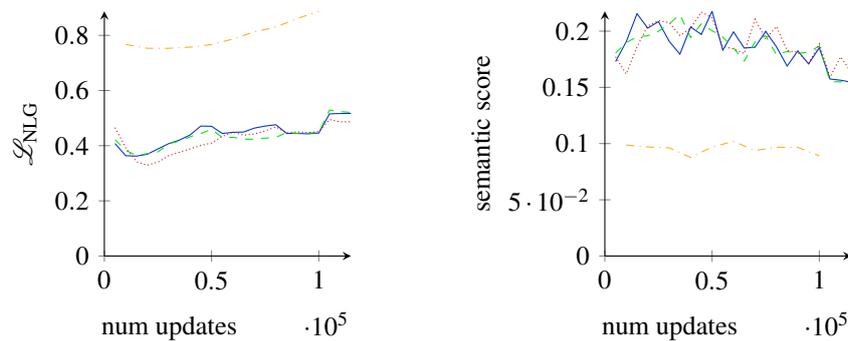


Figure B.2.: Curves of valid NLG loss (left) and semantic score (right) during training of the same models as shown in Fig. B.1 (the legend is also shared with that figure).

## C. Examples

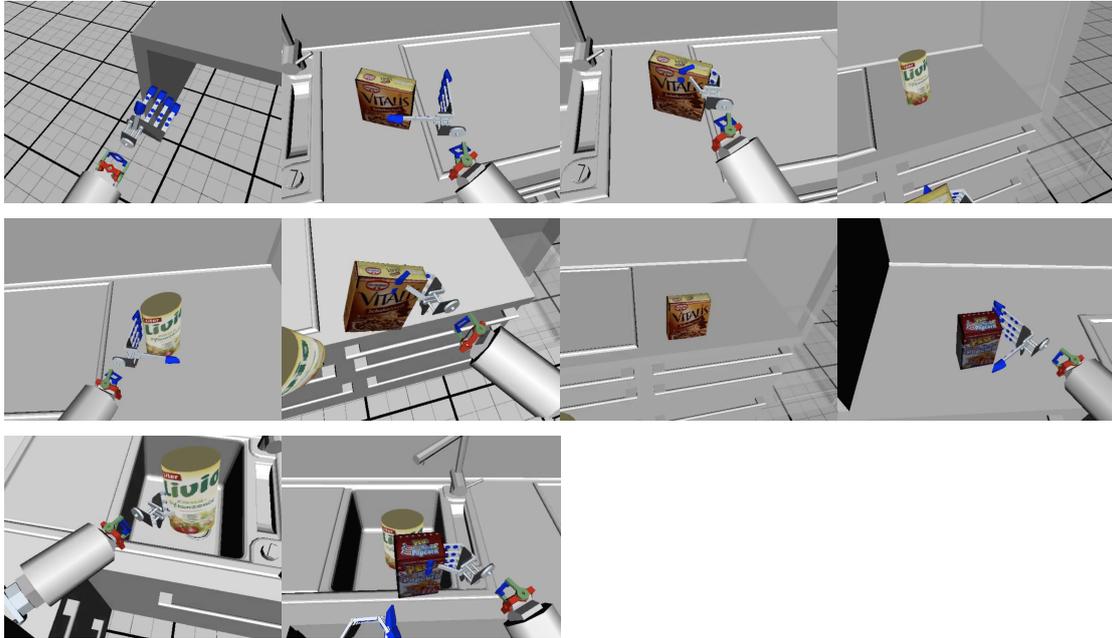


Figure C.1.: Example of a robot episode recording from the newly collected set of simulated robot executions as described in section 5.1.1. First row: The robot starts at the table, moves to the counter, grabs the cereal box, and moves on. Second row: It grabs the oil with the left hand, places the cereal box at the countertop and moves next to the dishwasher to grab the popcorn. Last row: Both the oil and the popcorn are put into the sink.



Figure C.2.: Example of a robot episode recording from the real robot recordings from Bärman et al. (2021) as described in section 5.1.3. From left to right: The robot grasps the green with its right and the red cup with its left hand, then moves to the sink, puts the red cup into it, and finally places the green cup at the countertop.

Status	Action + Arguments	Object	Position
started	move Armar3 roomcenter sideboard	greencup	462.1, 791.7, 890.8
success	move Armar3 roomcenter sideboard	handleft3a	-338.1, 529.6, 1024.4
started	grasp Armar3 handright3a sideboard greencup	...	...
failure	grasp Armar3 handright3a sideboard greencup	handleft3a	-134.6, 492.2, 1158.1
...	...	greencup	-24.4, 468.4, 1130.0
started	putdown Armar3 handleft3a sink redcup	handleft3a	-136.6, 488.3, 1156.3
success	putdown Armar3 handleft3a sink redcup		
started	move Armar3 sideboard countertop		
success	move Armar3 sideboard countertop		
started	putdown Armar3 handright3a countertop greencup		
success	putdown Armar3 handright3a countertop greencup		

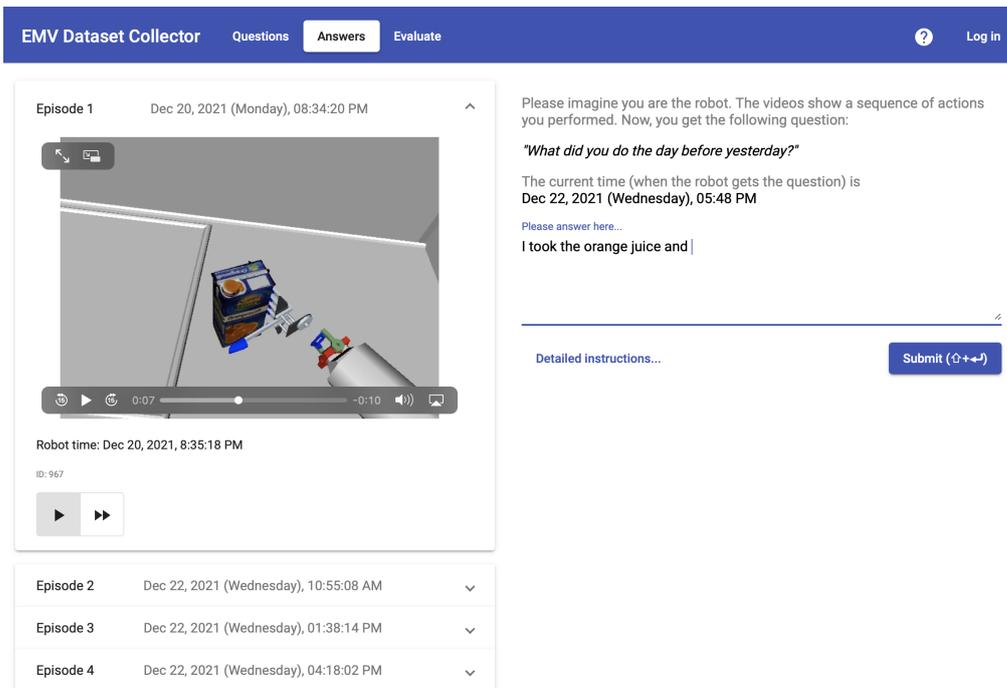
Table C.1.: Further information on the data from different modalities part of the RER shown in Fig. C.2. Left: Content of the symbolic information sequence containing action and arguments. Right: Symbolic object names with subsymbolic positions.

Category	Question	Answer
<i>real-robot-grammar-generated</i>		
Correct	explain your exercises two days ago at 03 PM	i moved the green cup to the countertop and the red cup to the sink
Wrong	how long exactly did it take to grasp the vitamin juice yesterday at noon?	it took 25 seconds ( <i>correct: 123s</i> )
<i>simulated-robot-human-annotated</i>		
Correct	What did you do in the morning?	i moved the multivitaminjuice to the roundtable
Wrong	Did you recently have any problems?	No ( <i>correct: Yes, I failed to ...</i> )
Inappr.	Which cup did you take on my Birthday, June the 24th?	i took 10 nesquik
<i>real-robot-human-annotated</i>		
Correct	What did you do after taking the Multi Vitamin Juice?	i took the vitamin juice to the countertop
Wrong	Which cup was next to the red cup?	i think you are talking about the red cup
Inappr.	Where did you put the cup?	i did it on June 20 at about 19 o clock

Table C.2.: Exemplary results produced by the CT + LT model on test samples from the respective generalization test set. Each row shows a question and answer categorized as shown. For the *real-robot-grammar-generated* test set, there were no inappropriate answers.

## D. Data Collection Site

The following screenshots show the dataset collection website of Bärmann et al. (2021), which was partly extended in this work. In the first screenshot, an answer needs to be annotated, given a question. The second picture shows the interface for manual evaluation of model responses.



**EMV Dataset Collector** Questions **Answers** Evaluate ? Log in

Episode 1 Dec 20, 2021 (Monday), 08:34:20 PM

Please imagine you are the robot. The videos show a sequence of actions you performed. Now, you get the following question:

*"What did you do the day before yesterday?"*

The current time (when the robot gets the question) is  
Dec 22, 2021 (Wednesday), 05:48 PM

Please answer here...

I took the orange juice and |

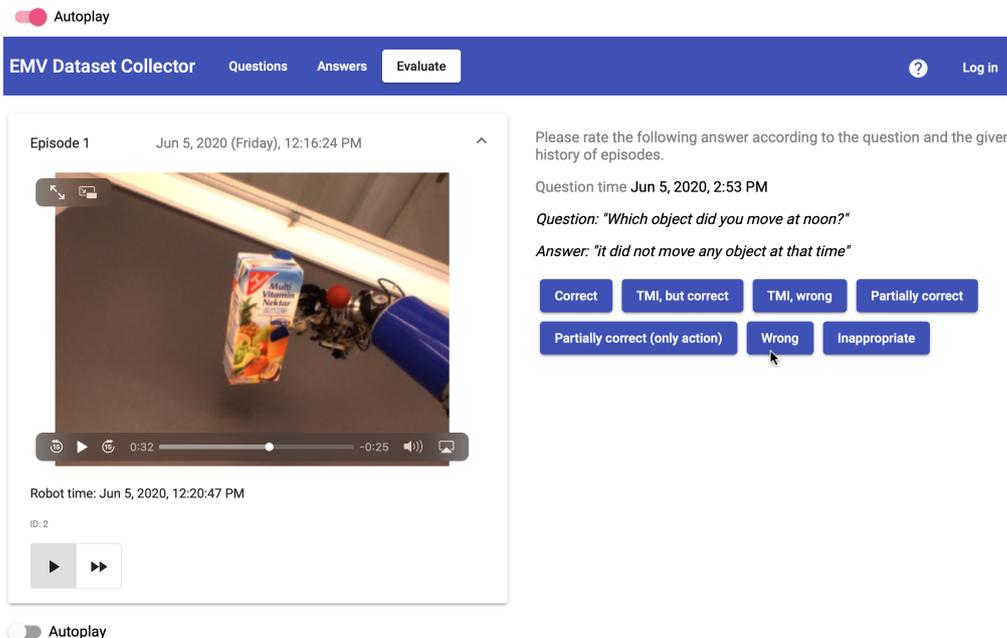
Detailed instructions... Submit (↵)

Robot time: Dec 20, 2021, 8:35:18 PM  
ID: 967

Episode 2 Dec 22, 2021 (Wednesday), 10:55:08 AM

Episode 3 Dec 22, 2021 (Wednesday), 01:38:14 PM

Episode 4 Dec 22, 2021 (Wednesday), 04:18:02 PM



**EMV Dataset Collector** Questions Answers **Evaluate** ? Log in

Episode 1 Jun 5, 2020 (Friday), 12:16:24 PM

Please rate the following answer according to the question and the given history of episodes.

Question time Jun 5, 2020, 2:53 PM

Question: "Which object did you move at noon?"

Answer: "it did not move any object at that time"

Correct TMI, but correct TMI, wrong Partially correct

Partially correct (only action) Wrong Inappropriate

Robot time: Jun 5, 2020, 12:20:47 PM  
ID: 2

Autoplay

## Bibliography

- Ainslie, Joshua et al. (2020). “ETC: Encoding Long and Structured Inputs in Transformers”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: ACL, pp. 268–284 (cit. on pp. 8, 12, 13).
- Asfour, T. et al. (2006). “ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 169–175 (cit. on p. 9).
- Ba, Jimmy Lei et al. (2016). “Layer Normalization”. In: *arXiv:1607.06450* (cit. on p. 8).
- Bahdanau, Dzmitry et al. (2015). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *International Conference on Learning Representations (ICLR)*. San Diego, CA, USA (cit. on pp. 6, 7, 24).
- Bai, Shaojie et al. (2018). “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling”. In: *arXiv:1803.01271* (cit. on pp. 3, 11).
- Baraldi, Lorenzo et al. (2017). “Hierarchical Boundary-Aware Neural Encoder for Video Captioning”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 11).
- Bärmann, Leonard et al. (2021). “Deep Episodic Memory for Verbalization of Robot Experience”. In: *IEEE Robotics and Automation Letters* 6.3, pp. 5808–5815. DOI: 10.1109/LRA.2021.3085166 (cit. on pp. 5, 6, 2, 3, 9, 10, 15–18, 23, 24, 26, 27, 29–32, 35, 36, 42–44, 48, 51, 53).
- Beetz, Michael et al. (2018). “KnowRob 2.0 — A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (cit. on p. 10).
- Beltagy, Iz et al. (2020). “Longformer: The Long-Document Transformer”. In: *arXiv:2004.05150* (cit. on p. 13).
- Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer. ISBN: 0-387-31073-8 (cit. on p. 4).
- Brown, Roger et al. (1977). “Flashbulb memories”. In: *Cognition* 5.1, pp. 73–99 (cit. on p. 45).
- Campos, Víctor et al. (2018). “Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks”. In: *International Conference on Learning Representations (ICLR)*. Vancouver, BC, Canada (cit. on pp. 11, 45).
- Cao, Zhe et al. (2018). “OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields”. In: *arXiv:1812.08008* (cit. on p. 16).
- Chang, Poo-Hee et al. (2017). “Encoding and Recall of Spatio-Temporal Episodic Memory in Real Time”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1490–1496 (cit. on p. 10).
- Chen, Tianqi et al. (2016). “Training Deep Nets with Sublinear Memory Cost”. In: *arXiv:1604.06174* (cit. on p. 14).
- Chen, Xiuyi et al. (2019). “A Working Memory Model for Task-oriented Dialog Response Generation”. In: *Annual Meeting of the Association for Computational Linguistics*, pp. 2687–2693 (cit. on p. 9).
- Cho, Kyunghyun et al. (2014). “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: ACL, pp. 1724–1734 (cit. on p. 4).
- Choi, Jihun et al. (2018). “Learning to Compose Task-Specific Tree Structures”. In: *AAAI Conference on Artificial Intelligence* 32.1 (cit. on p. 12).
- Chung, Junyoung, Sungjin Ahn, et al. (2017). “Hierarchical Multiscale Recurrent Neural Networks”. In: *International Conference on Learning Representations (ICLR)*. Toulon, France (cit. on pp. 11, 45).
- Chung, Junyoung, Caglar Gulcehre, et al. (2014). “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *NIPS Workshop on Deep Learning* (cit. on p. 4).

- Dai, Zihang et al. (2019). “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: ACL, pp. 2978–2988 (cit. on pp. 8, 13, 14, 18).
- Dwivedi, Debidatta et al. (2019). “Temporal Cycle-Consistency Learning”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 10).
- El Hihi, Salah et al. (1995). “Hierarchical Recurrent Neural Networks for Long-Term Dependencies”. In: *International Conference on Neural Information Processing Systems*. NIPS’95. Cambridge, MA, USA: MIT Press, pp. 493–499 (cit. on p. 10).
- Eric, Mihail et al. (2017). “Key-Value Retrieval Networks for Task-Oriented Dialogue”. In: *Annual SIG-dial Meeting on Discourse and Dialogue*, pp. 37–49 (cit. on p. 9).
- Graves, Alex et al. (2016). “Hybrid computing using a neural network with dynamic external memory”. In: *Nature* 538.7626, pp. 471–476 (cit. on p. 10).
- He, K. et al. (2016). “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778 (cit. on pp. 8, 11).
- Hochreiter, Sepp et al. (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780 (cit. on p. 4).
- Kádár, Ákos et al. (2018). “Revisiting the Hierarchical Multiscale LSTM”. In: *International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: ACL, pp. 3215–3227 (cit. on p. 11).
- Kumar, Ankit et al. (2016). “Ask Me Anything: Dynamic Memory Networks for Natural Language Processing”. In: *International Conference on Machine Learning (ICML)*. Vol. 48, pp. 1378–1387 (cit. on p. 9).
- Lowerre, Bruce (1976). “The Harpy Speech Recognition System”. PhD thesis. Carnegie Mellon University (cit. on p. 6).
- Olah, Chris (2015). *Understanding LSTM Networks*. URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> (cit. on pp. 4, 5).
- Olah, Chris and Shan Carter (2016). “Attention and Augmented Recurrent Neural Networks”. In: *Distill*. URL: <http://distill.pub/2016/augmented-rnns> (cit. on p. 6).
- Park, Gyeong-Moon et al. (2018). “Deep ART Neural Model for Biologically Inspired Episodic Memory and Its Application to Task Performance of Robots”. In: *IEEE Transactions on Cybernetics* 48.6, pp. 1786–1799 (cit. on p. 10).
- Perera, Vittorio et al. (2016). “Dynamic generation and refinement of robot verbalization”. In: *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pp. 212–218 (cit. on p. 9).
- Rae, Jack W., Anna Potapenko, et al. (2020). “Compressive Transformers for Long-Range Sequence Modelling”. In: *International Conference on Learning Representations* (cit. on pp. 13, 18–20, 25).
- Rae, Jack W. and Ali Razavi (2020). “Do Transformers Need Deep Long-Range Memory?” In: *Annual Meeting of the Association for Computational Linguistics*. Online: ACL, pp. 7524–7529 (cit. on pp. 18, 19).
- Raffel, Colin et al. (2020). “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *Journal of Machine Learning Research* 21.140, pp. 1–67 (cit. on p. 23).
- Rosenthal, Stephanie et al. (2016). “Verbalization: Narration of Autonomous Robot Experience”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 862–868 (cit. on pp. 2, 9).
- Rothfuss, J. et al. (2018). “Deep Episodic Memory: Encoding, Recalling, and Predicting Episodic Experiences for Robot Action Execution”. In: *IEEE Robotics and Automation Letters* 3.4, pp. 4007–4014 (cit. on pp. 10, 16, 23, 31).
- Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3. Place: USA Publisher: Kluwer Academic Publishers, pp. 211–252 (cit. on p. 16).
- Shen, Yikang et al. (2019). “Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks”. In: *International Conference on Learning Representations* (cit. on p. 12).
- Shi, Xingjian et al. (2015). “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”. In: *International Conference on Neural Information Processing Systems - Volume 1*. NIPS’15. Cambridge, MA, USA: MIT Press, pp. 802–810 (cit. on p. 10).

- Sutskever, Ilya et al. (2014). “Sequence to Sequence Learning with Neural Networks”. In: *International Conference on Neural Information Processing Systems - Volume 2*. NIPS’14. event-place: Montreal, Canada. Cambridge, MA, USA: MIT Press, pp. 3104–3112 (cit. on p. 5).
- Tay, Yi et al. (2020). “Efficient Transformers: A Survey”. In: *arXiv:2009.06732* (cit. on pp. 8, 12).
- Tulving, Endel (1972). “Episodic and semantic memory”. In: *Organization of memory* 1, pp. 381–403 (cit. on pp. 2, 9).
- Vahrenkamp, Nikolaus et al. (2015). “The robot software framework ArmarX”. In: *it - Information Technology* 57 (cit. on p. 26).
- Vaswani, Ashish et al. (2017). “Attention is All you Need”. In: *International Conference on Neural Information Processing Systems (NIPS)*, p. 11 (cit. on pp. 2, 6–8, 12, 23).
- Waibel, A. et al. (1987). “Phoneme recognition using time-delay neural networks”. In: Meeting of the Institute of Electrical, Information and Communication Engineers (IEICE). Tokyo, Japan, pp. 87–100 (cit. on p. 11).
- (1989). “Phoneme recognition using time-delay neural networks”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3, pp. 328–339 (cit. on p. 11).
- Wang, W. et al. (2012). “Neural Modeling of Episodic Memory: Encoding, Retrieval, and Forgetting”. In: *IEEE Transactions on Neural Networks and Learning Systems* 23.10, pp. 1574–1586 (cit. on p. 10).
- Wen, Tsung-Hsien et al. (2017). “A Network-based End-to-End Trainable Task-oriented Dialogue System”. In: *Conference of the European Chapter of the Association for Computational Linguistics*, pp. 438–449 (cit. on p. 9).
- Williams, Adina et al. (2018). “Do latent tree learning models identify meaningful structure in sentences?” In: *Transactions of the Association for Computational Linguistics* 6, pp. 253–267 (cit. on p. 12).
- Winkler, Jan et al. (2014). “CRAMm – Memories for Robots Performing Everyday Manipulation Activities”. In: *Advances in Cognitive Systems*. Vol. 3, pp. 47–66 (cit. on p. 10).
- Wu, C.-S. et al. (2019). “Global-to-local memory pointer networks for task-oriented dialogue”. In: *International Conference on Learning Representations (ICLR)* (cit. on p. 9).
- Wu, Qingyang et al. (2020). “Memformer: The Memory-Augmented Transformer”. In: *arXiv:2010.06891* (cit. on pp. 8, 14, 20).
- Xie, Saining et al. (2017). “Aggregated Residual Transformations for Deep Neural Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995 (cit. on p. 16).
- Yang, Zichao et al. (2016). “Hierarchical Attention Networks for Document Classification”. In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: ACL, pp. 1480–1489 (cit. on pp. 10, 12).
- Yu, Fisher et al. (2016). “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico (cit. on p. 11).
- Yu, Zhou et al. (2019). “ActivityNet-QA: A Dataset for Understanding Complex Web Videos via Question Answering”. In: *AAAI* 33.1, pp. 9127–9134 (cit. on p. 9).
- Zaheer, Manzil et al. (2020). “Big Bird: Transformers for Longer Sequences”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 17283–17297 (cit. on p. 13).
- Zhang, Xingxing et al. (2019). “HIBERT: Document Level Pre-training of Hierarchical Bidirectional Transformers for Document Summarization”. In: *Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: ACL, pp. 5059–5069 (cit. on p. 12).
- Zhao, Zhou et al. (2020). “Open-Ended Video Question Answering via Multi-Modal Conditional Adversarial Networks”. In: *IEEE Transactions on Image Processing* 29, pp. 3859–3870 (cit. on p. 9).
- Zhu, Qingxiaoyang et al. (2017). “Autonomous narration of humanoid robot kitchen task experience”. In: *IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, pp. 390–397 (cit. on pp. 2, 9).