H^{2}MT: SEMANTIC HIERARCHY-AWARE HIERARCHICAL MEMORY TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer-based large language models (LLMs) are used in language processing, yet when handling long context, they most often restrict the context window. Furthermore, many existing solutions are inefficient and overlook the structure inherent in documents. As a result, long-context models often treat text as a flat token stream, which disregards the semantic hierarchy of the document and wastes computation by processing both relevant and irrelevant context. We present **H^{2}MT : Semantic Hierarchy-Aware Hierarchical Memory Transformer** a hierarchy-aware approach that plugs into a backbone. H^{2}MT represents the document as a semantic hierarchy tree and performs level-conditioned routing and aggregation. Thus, bottom-level's information is propagated into their ancestors to preserve relative context. We evaluate H^{2}MT on Qasper Dasigi et al. (2021) and a question-answer set derived from the Siemens Calibre technical manual for the downstream question-answering (QA) task.. H^{2}MT improves quality at a similar model size while reducing the required memory. The approach is most helpful for data with a semantic hierarchy that can be modeled as a hierarchy tree. Across Qasper and the Siemens Calibre QA set, H^{2}MT improves ROUGE-L and reduces the time-to-first-token generation.

1 INTRODUCTION

Natural language processing applications require large language models to process long, structured documents (e.g., scientific papers or technical manuals) for downstream tasks like question-answering or summarization. Since the introduction of the Transformer (Vaswani et al., 2017), it has become the dominant architecture in language models. However, the quadratic cost of dense self-attention in sequence length makes long inputs compute- and memory-intensive on common hardware. like GPUs. In practice, researchers propose methods to extend context windows or add sparse/global tokens Zaheer et al. (2020); Wang et al. (2020); Peng et al. (2024). Moreover, to maintain inference speed and stay within memory limits, most transformer models enforce a maximum sequence length Grattafiori et al. (2024); Yang et al. (2024). These tactics help, but they still present a document as a flat stream of tokens. Especially, real-world documents have often inherent semantic structure (e.g., sections that define terms, subsections that depend on earlier parts), and ignoring this hierarchy diffuses attention over distant, often irrelevant context.

In this paper, we introduce a complementary view: preserve a document's semantic hierarchy and shift long-range reasoning from token-level attention to hierarchy-conditioned *memory* composition. Using Hierarchical Memory Transformer (HMT) (He et al., 2025), **H^{2}MT : Semantic Hierarchy-Aware Hierarchical Memory Transformer** places and composes embeddings along the document semantic hierarchy tree rather than a flat sequence: leaf units are summarized into compact embeddings via HMT; intermediate nodes aggregate child embeddings with mean or self-attention pooling, propagating related information upward to preserve context scope. Unlike HMT's sequential, global retrieval, H^{2}MT performs lightweight top-down routing at inference to prune low-relevance branches in the hierarchy tree. As a result, the retrieval will be limited to a subtree. This approach reduces interference, compute, and time-to-first-token without enlarging the context window or modifying the backbone.

Compared with structure-aware sparse-attention approaches that encode hierarchy by modifying attention patterns (e.g., anchor tokens, hierarchical sparsity) and with retrieval-augmented meth-

ods that repeatedly re-encode large contexts, H^2MT leaves the backbone unchanged. It adds only lightweight pooling and projection layers and reuses cached memory embeddings across nodes during inference. This approach yields a parameter-efficient mechanism for long structured documents. The method is most beneficial when the data's structure can be reasonably modeled as a tree; handling general cross-references (DAGs) is a natural extension.

Contributions.

- We model a document's *semantic hierarchy* as a rooted tree and cast long-document processing as *hierarchical embedding composition*: child nodes are summarized into memory embeddings and aggregated upward for reuse by ancestors.
- We instantiate this formulation with HMT to produce node-level memory embeddings and retrieval, coupled with simple, pluggable aggregation policies (mean pooling and self-attention pooling) at internal nodes.
- We provide a post-order encoding schedule that caches and reuses constant-size embeddings from bottom-levels. This approach reduces reliance on wide-span self-attention for long-range dependencies.
- We demonstrate improvements on QA task for Siemens Calibre technical manual and Qasper.

Scope and limitations. H^2MT assumes access to the document hierarchy (e.g., table of contents or sectioning) and performs best when cross-references do not violate tree structure heavily. Extending the propagation schedule to directed acyclic graphs (DAGs) is an important direction for future work.

Relation to prior work. Our approach uses the memory-embedding and retrieval interface introduced by HMT (He et al., 2025), adapting it to a hierarchy-conditioned setting in which child summaries are aggregated and reused by ancestors. We leverage hierarchy in the *memory pathway* while keeping the backbone's global attention unchanged.

1.1 Related Work

Long-context modeling has advanced along two largely orthogonal lines: (i) *structure-aware* architectures that encode document hierarchy as an explicit inductive bias, and (ii) *memory-augmented* approaches that compress and retrieve information across the input.

Structure-aware long-document models. The Hierarchical Document Transformer (HDT) He et al. (2024) injects document structure directly into the attention pattern via auxiliary anchor tokens (e.g., [SENT], [SEC], [DOC]) and a sample-dependent, multi-level sparse attention kernel, complemented by hierarchical positional encodings. HDT enables communication among siblings and parent–child nodes within a document tree while maintaining sparsity, yielding efficiency gains and improved pretraining dynamics on long-document tasks. In contrast, our method, H^2MT , retains a standard backbone and routes cross-level information through a compact memory interface rather than redesigned attention masks.

Memory-augmented recurrence and retrieval. The Hierarchical Memory Transformer (HMT) He et al. (2025) provides a model-agnostic memory interface that summarizes each input portion into memory embeddings and retrieves relevant memories during subsequent processing (He et al., 2025). HMT creating memory hierarchies (sensory/short-term/long-term) and implements sequence-recurrent processing, improving long-context performance and reducing inference memory relative to training larger long-window models. We use the HMT for the context of each node in the hierarchy tree, but organize and propagate memory embeddings along the tree. Thus, child memories are aggregated and made available to their ancestors during retrieval. Thus, H^2MT complements structure-aware designs by exploiting semantic hierarchy rather than in the global attention pattern.

Long-context Transformers. A principal bottleneck in the transformers is the quadratic cost of self-attention. This motivated sparse attention for long inputs. A basic strategy is sliding-window attention, which limits long-range interactions. Extensions such as Longformer (Beltagy et al., 2020) and pooling-based variants like Poolingformer (Zhang et al., 2021) add global tokens and pooling to enlarge the effective receptive field. Retrieval-augmented approaches (e.g., Unlimiformer (Bertsch et al., 2023)) select a subset of salient tokens or spans and focus attention on those, pruning computation while aiming to preserve generation quality. However, contributions from less relevant tokens can still accumulate across long generations, and memory usage typically grows with input length because activations for selected or pooled tokens must be retained. An alternative line compresses past tokens into fixed-size states via recurrent or compressive sequence models, reducing memory by condensing information into constant-size embeddings. Our work follows this compression view via the HMT memory embeddings (He et al., 2025) while organizing and propagating them along a document tree.

2 METHODOLOGY

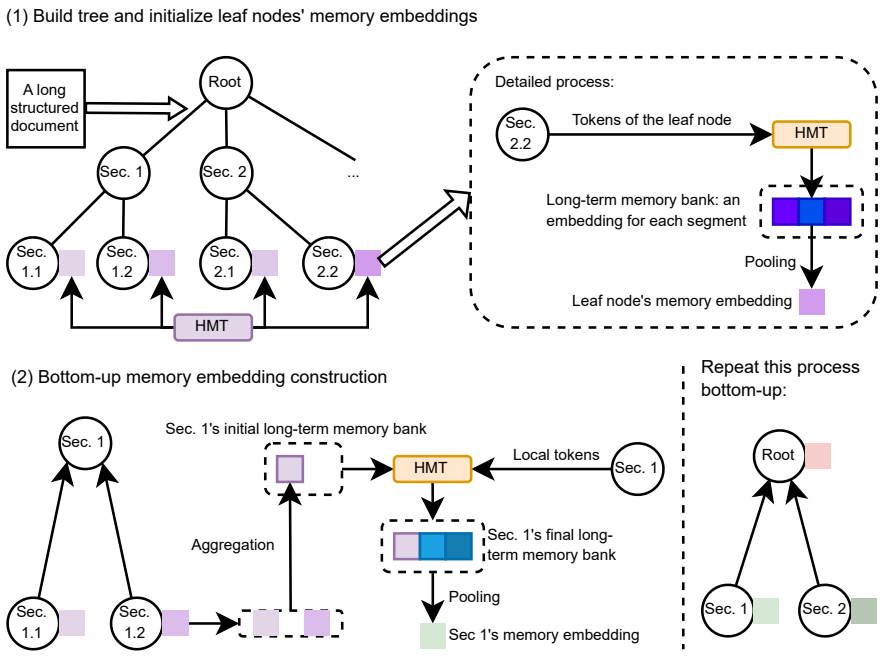

Figure 1: Overview of H^{2}MT . Documents are represented as a tree; leaf nodes produce memory embeddings (via HMT), which are aggregated bottom-up and reused during parent encoding.

H^{2}MT contains three main stages: (i) generating the document's semantic hierarchy tree (Section 2.1), (ii) bottom-up tree traversal to produce *memory embeddings* for each node using HMT, and for the intermediate nodes (iii) aggregating child node's memory embeddings when encoding the intermediate nodes in the semantic hierarchy. The design targets long-context settings while keeping the backbone architecture unchanged.

2.1 SEMANTIC HIERARCHY TREE

We represent a document's semantic structure as a rooted tree $\mathcal{T} = (V, E, r)$. Each node $v \in V$ is a semantically coherent unit (e.g. section, subsection), E is the parent–child edge set, and r is the root node. Let $\mathrm{children}(v)$ denote the set of children of v and $\mathrm{parent}(v)$ its parent (undefined for r).

Edges are interpreted as context scope: if $(\text{parent}(v), v) \in E$, then the parent provides definitional or notational context for v. Also, Each node v contains tokens related to it $X_v = (x_1, \ldots, x_{|X_v|})$. We partition X_v into segments of length at most L tokens:

$$X_v = \{X_{v,1}, \ldots, X_{v,S_v}\}, \qquad |X_{v,s}| \leq L, \qquad S_v = \left\lceil \frac{|X_v|}{L} \right\rceil. \tag{1}$$

Note that, because the semantic hierarchy is constructed from the original long-context document, a node's local content often fits within a single segment.

2.2 Bottom-Up Memory Embedding Construction

First, we explain how we derive the memory embeddings of the nodes and then we explain the traversal.

2.2.1 Memory Embeddings

In each node, we consider the tokens of its context and its segments $X_{v,1}, ..., X_{v,S_v}$. Each segment $X_{v,s}$ is encoded by the backbone (with a summarization prompt) into a single *memory embedding* $m_{v,s} \in \mathbb{R}^d$. We collect node-level memory embeddings as $M_v = \{m_{v,1}, \ldots, m_{v,S_v}\}$. When encoding $X_{v,s}$ when v is an intermediate node, a set of previously computed memory embeddings may be made available for retrieval; we denote this available set by $\mathcal{L}_v^{\text{in}}$ (defined precisely in Sec. 2.2).

Interpretation. M_v summarizes X_v into a small set of vectors compatible with HMT's retrieval mechanism. These provide the only cross-node information pathway and enable bottom-up message passing in H^{2}MT.

2.2.2 Bottom-Up Tree Traversal

We compute memory children in post-order so that children are encoded before their parents:

Leaves. For v with $\text{children}(v) = \emptyset$, set $\mathcal{L}_v^{\text{in}} = \emptyset$ where $\mathcal{L}_v$ is the long term memory and compute M_v (memory embedding) using HMT on $\{X_{v,s}\}_{s=1}^{S_v}$.

Internal nodes. Assume M_u has been computed for all $u \in \text{children}(v)$. We use aggregation policies to combine its children's memory embeddings. This schedule reuses already-computed, constant-size embeddings of descendants instead of re-encoding long child texts.

2.2.3 Aggregation Policies

We summarize $\mathcal{C}_v = \{c_1, \ldots, c_{n_v}\}$ (child memories of intermediate node v) with lightweight, permutation-invariant aggregated memory embedding. To aggregate the memory embeddings, we can use mean aggregation or self-attention pooling.

Mean Aggregation. We take the arithmetic mean of the child memory embeddings. This aggregation policy is parameter-free and symmetric across children.

Self-Attention Pooling. We run dot-product attention over the child memory embeddings: each child is linearly projected to query/key spaces and attends over the others. We convert the resulting attention matrix into a single weight per child by aggregating how much attention each child node receives across all queries; the weights are normalized to sum to one and used to form a weighted average of the child memories. This adds only two small projections.

2.3 Downstream Tasks

During the bottom-up traversal, we compute memory embeddings for every node. For intermediate nodes, these memory embeddings are obtained conditioned on their descendants' memories. Thus, they summarize both local and child context. After precomputing document-level memory embeddings, we fine-tune for downstream tasks as needed. For question answering, we attach the question to a pertinent node v and its memory embedding.

2.4 TOP-DOWN INFERENCE-TIME ROUTING

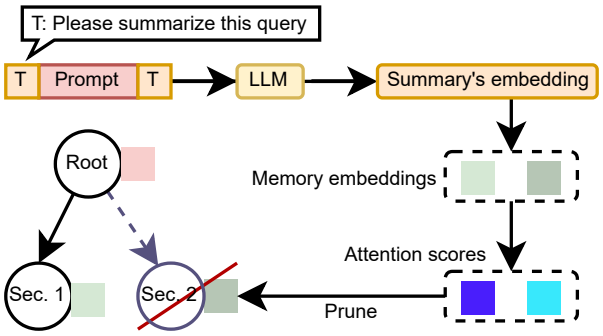

Figure 2: Inference in H^{2}MT . The backbone first produces a short summary embedding (query). The query attends over semantic hierarchy-level memory embeddings at the root, producing attention scores that rank sections. Low-scoring branches are pruned and only the selected sub-tree is read, reusing cached, bottom-up aggregated memories at each node. This top-down routing focuses computation on relevant context and reduces time-to-first-token.

For each context (e.g., the question about a paper or a technical manual), we first obtain a short summary vector by running the backbone once with the HMT memorization prompt. This yields a summary embedding that we use as a query. Using this query, we perform lightweight cross-attention over the memory embeddings of high-level nodes in the document semantic hierarchy (e.g., top-level sections). The resulting scores identify the few sections most relevant to the query. This approach causes routing computation to the pertinent branches of the tree. In other words, we prune the document to the most relevant sub-tree.

For instance, in question answering, if the question pertains to Section 1, the mechanism concentrates on Section 1's memory embeddings, rather than attending broadly across unrelated sections. This yields a top-down inference procedure that reuses the precomputed bottom-up summaries while limiting retrieval to the parts of the hierarchy most likely to influence the output.

3 EXPERIMENTS

In this section, we evaluate H^{2}MT 's performance on multiple structured-datasets with different aggregation methods.

3.1 DATASETS

We use two types od structured datasets: (1) Qasper and (2) questions and answers from Siemens Calibre technical manual. The Qasper dataset is from a widely acknowledged long-context dataset and it is publicly available. The question-answering from the Siemens Calibre technical manual is a private dataset from Siemens, which is a manual for their Calibre tool. The question and answers are extracted from this manual. The QA generation of the Siemens Calibre dataset is described in the appendix.

3.1.1 HIERARCHY TREE GENERATION

Qasper. Qasper contains questions answers from papers, and we have the paper's context as well. Also, annotators have marked the golden evidence containing the part that the answer found from. For each paper, we make the semantic hierarchy considering starting from each section in the paper as the children of the paper and the subsections as the children of the sections. For example we have Experiments, a Dataset node under the Experiment node, Experiment setup as another child. As a result we would have the semantic hierarchy graph for all papers in the datasets.

Siemens Calibre Technical Manual QA. For technical manuals, we have table of contents that shows the file references and their relationships. We build the semantic hierarchy tree based on it.

3.2 BASELINES

HMT. We compare H^{2}MT with HMT, which is supposed to handle long-context well.

Backbone Model. we compare H^{2}MT with the backbone models Llama 3.1 8B Instruct and Qwen 2.5 14B Instruct.

3.3 EVALUATION

We evaluate H^{2}MT 's question-answering performance on the Qasper and Siemens Calibre QA datasets. These tasks require the model to understand full-paper context (Qasper) and technical-manual context (Siemens Calibre).

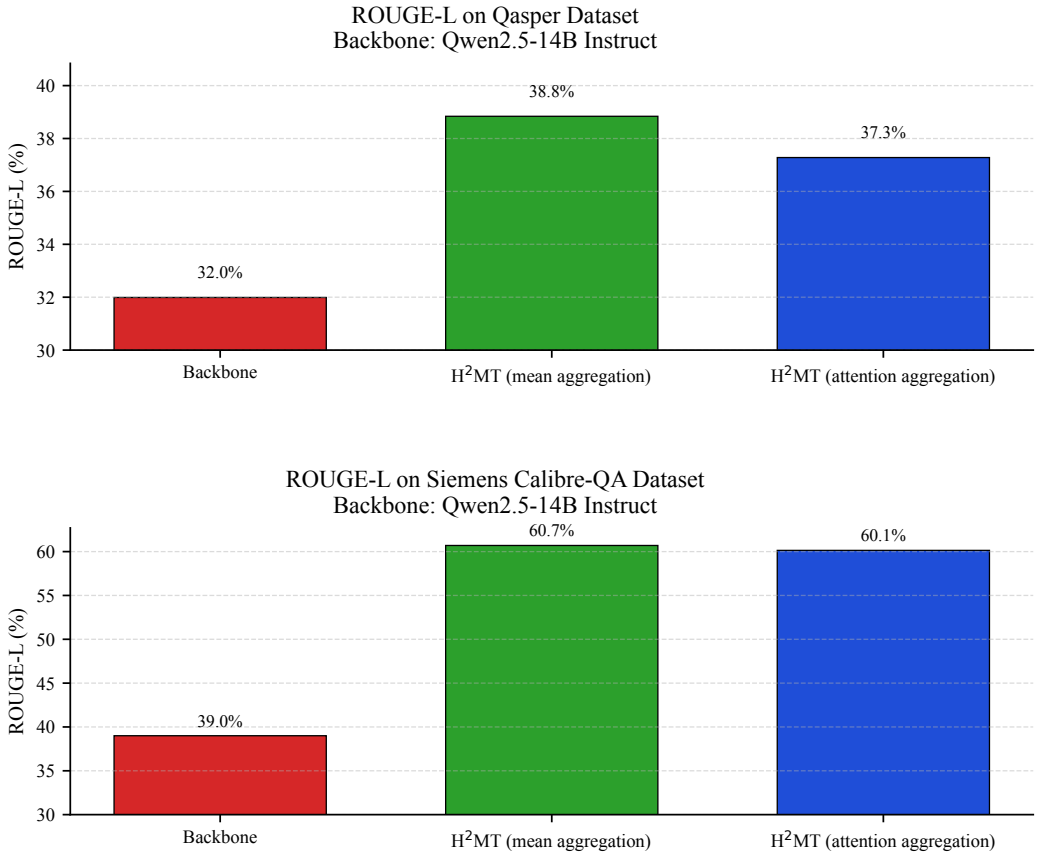

Figure 3: ROUGE-L on QASPER (top) and Siemens Calibre QA (bottom) with a Qwen 2.5 14B Instruct backbone. Bar 1: backbone-only; Bars 2–3: H^{2}MT with two aggregation policies.

Quality. Based on Fig. 3, mean aggregation is slightly stronger on both datasets, but both policies substantially outperform the backbone. One reason the attention policy method is not better than the mean policy can be the lossy summarization we make, and based on that, we apply aggregation weights which further emphasize the loss.

Latency (TTFT) on Llama 3.1 8B Instruct. Table 1 summarizes time-to-first-token (TTFT; lower is better). Using Llama 3.1 8B Instruct as the backbone, H^{2}MT cuts mean TTFT from 1.2015 s to 0.2211 s ($\approx$5.4$\times$ faster; 81.6% lower) Overall, H^{2}MT delivers faster and more stable first-token latency than HMT on the same model.

Table 1: Time-to-first-token (TTFT) in seconds (lower is better). The backbone model is Llama 3.1 8B Instruct

Model	Mean	Std	Min	Max
H^2MT	0.2211	0.1411	0.1150	0.5295
HMT	1.2015	0.6115	0.3175	2.9232

Method	ROUGE-L (%)
HMT	54.5
H^2MT (mean aggregation)	60.7
H^2MT (self-attention aggregation)	60.1

Table 2: ROUGE-L result to compare H^2MT with HMT with Qwen 2.5 14B Instruct backbone.

ROUGE-L: H^2MT vs. HMT (Qwen 2.5 14B Instruct) With the Qwen 2.5 14B Instruct backbone, H^2MT improves ROUGE-L over HMT from 54.5 to 60.7 (mean) and 60.1 (self-attention aggregation) (Table 22). The 0.6 gap between aggregators is minor, indicating that mean aggregation suffices; significance will be assessed with multi-seed runs.

4 Conclusion

We presented H^2MT , a semantic hierarchy-aware framework for making the long-context language modeling on structured data better. H^2MT represents a document as a rooted tree, summarizes each unit into compact memory embeddings via the HMT, and performs a post-order, bottom-up aggregation that passes information from children to parents. During parent encoding, retrieval over the aggregated child memories conditions the backbone without expanding its architecture. This shifts long-range reasoning from wide-span token attention to a small set of cached summaries while keeping parameter count and implementation overhead modest. The results, together with an analysis of compute/memory trade-offs, suggest that explicitly organizing memory along a document's semantic hierarchy is a practical way to exploit structure in long inputs.

Ethics Statement. Our experiments use (i) QASPER, a public dataset commonly used in research, and (ii) a proprietary Siemens Calibre technical manual used only for internal evaluation. In H^2MT , we compute and cache per-node memory embeddings during the bottom-up traversal and reuse them for downstream tasks and inference. Although they are lossy summaries, they may still encode sensitive content. To mitigate risk, we compute embeddings only on the aforementioned datasets, store them with access restricted to the authors, never transmit them to third parties or repurpose them for additional pretraining. Furthermore, no human subjects or personally identifiable information are involved, and no additional data were collected from individuals. For prospective deployments, we recommend opt-in caching or document-level access control.

Reproducibility Statement. We delineate the modeling components in §2: the semantic hierarchy (§2.1), HMT-based memory embeddings (§2.2.1), bottom-up construction and aggregation (§2.2, §2.2.3), and inference-time retrieval (§**??**). Datasets and hierarchy construction are described in .... For QASPER, we follow the standard dataset split and evaluation protocol as cited. For the Siemens Calibre manual, we specify the task formulation and hierarchy construction in the main text; the proprietary manual is not redistributed. To facilitate replication, we will provide configuration details (backbone choices, segment length L, and other hyperparameter), along with scripts for preprocessing, hierarchy building, and evaluation, in the code release of H^2MT .

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL `https://arxiv.org/abs/2004.05150`.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. Unlimiformer: Long-range transformers with unlimited length input. *arXiv preprint arXiv:2305.01625*, 2023. NeurIPS 2023.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, jun 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365/.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Anthony Grattafiori et al. The llama 3 herd of models. 2024. URL https://arxiv.org/abs/2407.21783.

Haoyu He, Markus Flicke, Jan Buchmann, Iryna Gurevych, and Andreas Geiger. Hdt: Hierarchical document transformer. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/pdf?id=dkpeWQRmlc. Published as a conference paper at COLM 2024.

Zifan He, Yingqi Cao, Zongyue Qin, Neha Prakriya, Yizhou Sun, and Jason Cong. Hmt: Hierarchical memory transformer for efficient long context language processing. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8068–8089. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.naacl-long.410.pdf.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008, 2017. URL https://arxiv.org/abs/1706.03762.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2006.04768.

An Yang et al. Qwen2 technical report. 2024. URL https://arxiv.org/abs/2407.10671.

Manzil Zaheer, Guru Guruganesh, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2007.14062.

Hang Zhang, Yeyun Gong, Yelong Shen, Weisheng Li, Jiancheng Lv, Nan Duan, and Weizhu Chen. Poolingformer: Long document modeling with pooling attention. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2021.

A Appendix

A.1 Siemens Calibre QA Construction

We derive the Siemens Calibre QA set directly from the product manual's HTML corpus. Each HTML file is first typed as *rich* or *light* by simple structural cues. We mark a file as *rich* if it contains any of: <pre> or <code> blocks, tables with header rows, "Warning/Note/Caution" callouts, option matrices, or numbered constraint lists; files without these cues are labeled *light*. We then reconstruct the document hierarchy from the manual's table of contents (TOC).We define a file's *related* context as its immediate children in this tree.

We instruct an open-source generator (DeepSeek-R1DeepSeek-AI (2025)) to produce one to N question–answer pairs that are answerable based on the file context. To fit the file type, we restrict the question categories: for *rich* nodes, we allow {definition, usage, syntax, parameter, warning, constraint, list_children, where_to_find_child, code_example}; for *light* nodes, we allow {definition, scope, usage, list_children, syntax}. By construction, every QA item is grounded in the union of a node and its immediately related files in the TOC hierarchy.