AttentionSmithy: A Modular Framework for Rapid Transformer Development and Customization

Anonymous authors
Paper under double-blind review

Abstract

Transformer architectures have revolutionized a broad spectrum of AI applications by leveraging attention mechanisms for parallelized and long-range sequence processing. Despite their remarkable success, building and customizing transformers remains prohibitively complex for many domain experts who lack deep knowledge of low-level implementations. We introduce AttentionSmithy, a modular software package that lowers the barrier to transformer innovation by decomposing key components—attention modules, feed-forward networks, normalization layers, and positional encodings—into reusable building blocks. By disentangling architectural elements into well-defined interfaces, users can rapidly prototype, adapt, and evaluate transformer variants without extensive coding overhead. Our framework supports four distinct positional encoding strategies (sinusoidal, learned, rotary, and ALiBi) and integrates seamlessly with neural architecture search (NAS) for automated design exploration. We validate AttentionSmithy by replicating the original "Attention Is All You Need" transformer under resource constraints, demonstrating near state-of-the-art performance on a machine translation task. Leveraging the package's integrated NAS capability, we made the unexpected discovery that machine translation performance is maximized by combining all available positional encoding methods—highlighting the complementary benefits of each strategy. We further illustrate AttentionSmithy's adaptability through gene-specific modeling, where a variant of a BERT-style architecture achieves over 95% accuracy on downstream cell type classification tasks using ranked transcriptomic data. These case studies underscore AttentionSmithy's core advantage: enabling specialized experimentation across diverse application domains—from natural language processing to genomic analysis—by obviating the need for labor-intensive, low-level framework manipulation. We anticipate that AttentionSmithy will serve as a foundation for creative transformer-based solutions, expediting research and development in numerous scientific and industrial fields.

1 Introduction

The transformer architecture (Vaswani et al., 2023) has revolutionized artificial intelligence, fundamentally changing how we approach sequence processing tasks across diverse domains. As transformer-based models continue to drive technological advancement and reshape societal interactions (Haque & Li, 2024), there is growing interest in adapting these architectures for specialized applications. However, customizing transformer architectures remains a significant challenge, requiring deep expertise in both the theoretical foundations and implementation details. This complexity creates a barrier for domain experts who could otherwise leverage transformer capabilities for novel applications.

1.1 Transformer Architecture Fundamentals

While traditional recurrent neural networks like Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) excelled at processing sequential data, they faced inherent limitations in parallelization and capturing long-range dependencies. The transformer architecture introduced by Vaswani et al. (2023) overcame these constraints through its innovative attention mechanism, enabling unprecedented advances in

natural language processing (Patwardhan et al., 2023), computer vision (Pereira & Hussain, 2024), healthcare applications (Nerella et al., 2024), molecular science research (Jiang et al., 2024), and genomic analysis (Choi & Lee, 2023).

The basic building blocks of a transformer include [**Figure 1A**]:

1. Multi-head attention layers that compute and weigh relationships between all sequence elements in parallel
2. Feed-forward neural networks that process these relationships through non-linear transformations
3. Layer normalization components that stabilize training by normalizing activations across features
4. Residual connections that facilitate gradient flow and help preserve and reuse features from earlier layers
5. Positional encoding mechanisms that preserve sequence order information by encoding relative or absolute positions

1.2 Positional Encoding Strategies

A crucial aspect of transformer architectures is their handling of sequential information through positional encodings. Without such encodings, transformers would treat input sequences as unordered sets of tokens, losing critical information about both absolute positions (where exactly a token appears in the sequence) and relative positions (how tokens are ordered with respect to each other). For instance, the sentences "the dog chased the cat" and "the cat chased the dog" contain identical tokens but convey opposite meanings, while "chased cat dog the the" is syntactically invalid – distinctions a transformer could not make without position information. While the self-attention mechanism excels at capturing relationships between tokens, it is inherently permutation-invariant, necessitating an explicit method to encode positional context. Several strategies have emerged, each with unique implementation requirements:

Sinusoidal positional encodings (Vaswani et al., 2023) and learned positional embeddings (Wang & Chen, 2020) operate by adding position-specific vectors directly to input token embeddings. This straightforward approach allows for easy implementation but may have limitations in capturing relative positions effectively.

Rotary positional embeddings (Su et al., 2023) take a different approach, modifying the attention computation itself by applying rotation transformations to the query and key matrices. This method has shown particular promise in capturing relative positioning information while maintaining consistent attention patterns across sequence lengths.

ALiBi positional encodings (Press et al., 2022) introduce position-specific bias terms to the attention score matrix, effectively modulating the attention weights based on relative positions. This approach has demonstrated advantages in extrapolating to longer sequences than those seen during training.

1.3 Architectural Experimentation and Search

The modular nature of transformer architectures presents significant opportunities for systematic architectural exploration. Neural architecture search (NAS) has emerged as a promising approach for discovering optimal neural network configurations, but its application to transformers remains limited. While specialized NAS frameworks have been developed for transformers (Chitty-Venkata et al., 2022; Liu et al., 2022), they are typically purpose-built for specific research objectives, making it difficult for practitioners to adapt them for novel transformer architectures and unique application domains. Traditional implementations often tightly couple architectural elements, making it challenging to define a comprehensive search space that includes variations in attention mechanisms, positional encodings, and feed-forward networks.

1.4 Current Tools and Limitations

While popular libraries like HuggingFace (Wolf et al., 2020), PyTorch (Ansel et al., 2024), and TensorFlow (Abadi et al., 2015) provide implementations of standard transformer variants, they typically offer limited

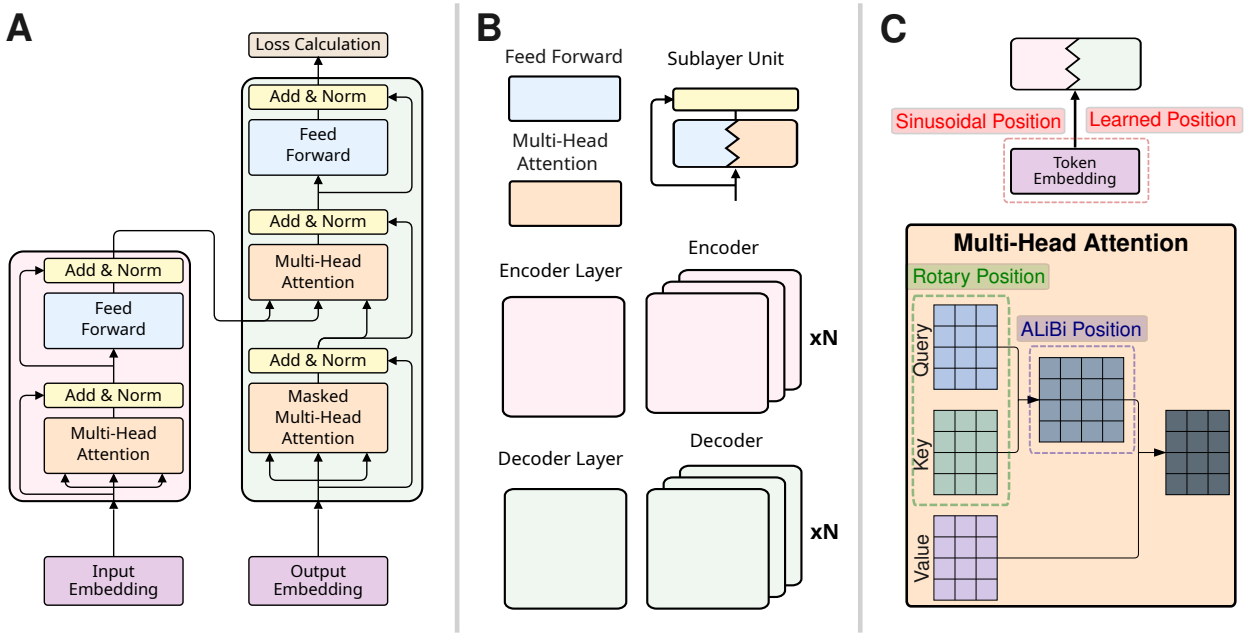


Figure 1: **Components of the transformer model architecture as coded in AttentionSmithy**. (A) The original transformer architecture introduced by Vaswani et al. (2023) (B) Labelled AttentionSmithy classes include implementations of the feed forward network, multi-head attention, sublayer unit (consisting of layer normalization and a residual connection surrounding an exchangeable feed forward network or multi-head attention class), encoder/decoder layers, and full encoders/decoders (which consist of an "N" number of layers determined by the end user). (C) Explicitly encoding position is a requirement of transformer models, but how position is encoded varies dramatically in implementation requirements from strategy to strategy. Four common strategies and their implementation details are outlined. The sinusoidal and learned positional embedding methods (red) involve directly adding vectors representing absolute positions to token embeddings before entering encoder or decoder layers. The rotary positional embedding method (green) requires adjusting the query and key matrices of the attention calculation directly. The ALiBi position embedding method (blue) adds negative values to the output of the query and key matrix multiplication that accumulate over greater positional distances.

flexibility for architectural customization. These frameworks often implement positional encoding methods as integral parts of specific architectures, making it difficult to experiment with different encoding strategies or architectural variations. This tight coupling not only impedes manual experimentation but also makes it challenging to implement automated architecture search strategies. The complexity of transformer customization calls for a modular, component-based approach aligned with established software design principles (Ousterhout, 2018; Vogel et al., 2011).

We present a novel software package called AttentionSmithy aimed at democratizing transformer development for domain experts. Drawing inspiration from design patterns (Gamma et al., 1995), AttentionSmithy implements a "building block" architecture that promotes code reuse and maintainability while preserving architectural clarity. This modular design simultaneously serves multiple purposes: it facilitates rapid prototyping, enables systematic architecture search, and serves as an educational tool for understanding transformer architectures. By abstracting transformer components into reusable modules, we enable rapid experimentation while maintaining architectural clarity. Furthermore, AttentionSmithy includes positional encoding strategy implementations that empower developers to select the methods most suitable for their use case. Our approach addresses a critical gap in the current landscape, where despite the proliferation of transformer variants (Amatriain et al., 2024), implementing customized architectures remains a complex undertaking that often requires building from scratch.

2 Methods

2.1 Software Architecture

Our software package implements a component-based design philosophy to facilitate the creation of customized transformer architectures. The core architecture breaks down transformer components into modular, reusable units that can be easily assembled and modified. This approach enables researchers to experiment with architectural variations while maintaining code readability and understanding of the underlying mechanisms.

The implementation utilizes PyTorch as its foundation and comprises distinct classes for each major transformer component. These components include multi-head attention mechanisms, feed-forward networks, normalization and dropout layers (implemented together as a "sublayer unit"), encoder/decoder layers, and complete encoder/decoder structures [**Figure 1B**]. Additionally, we provide both greedy and beam search generators for sequence generation tasks.

2.2 Key Features

2.2.1 Flexible Positional Encoding Framework

A key architectural feature is the implementation of a positional embedding strategy pattern that manages various numeric embedding approaches. A strategy manager serves as an intermediary for selecting and applying different positional encoding implementations within the transformer architecture, allowing for seamless integration of different approaches without requiring modifications to the core architecture.

Our implementation currently supports four distinct positional encoding strategies: sinusoidal, learned, rotary, and ALiBi embeddings, chosen as representative examples of popular approaches in the field. Each strategy is implemented as a separate class and managed through the embedding strategy manager, with the architecture designed to readily accommodate additional encoding strategies as they emerge. This flexibility is particularly valuable because different positional encoding strategies require fundamentally different implementations within the transformer architecture: sinusoidal and learned positional embeddings are added to input token vectors, rotary positional embedding requires adjusting the query and key matrices in the attention calculation, and ALiBi adds values to the attention score matrix (the product of the query and key matrices) [**Figure 1C**]. Our extensible design allows users to activate or deactivate these varied encoding strategies independently, enabling direct comparisons of their effectiveness in various applications, while also providing a framework for implementing and testing novel positional encoding approaches.

2.2.2 Modular Attention Mechanisms

The multi-head attention implementation allows for varying attention methods to be specified at initialization. This design choice facilitates future extensions, such as the incorporation of sparse attention patterns like Longformer (Beltagy et al., 2020) or Big Bird (Zaheer et al., 2021), while maintaining a consistent interface for the rest of the architecture [**Supplemental Figure 1**].

2.2.3 Neural Architecture Search Compatibility

The modular design of AttentionSmithy facilitates automated architecture optimization through neural architecture search (NAS). Components can be easily swapped or modified programmatically, allowing for systematic exploration of architectural variations while maintaining code interpretability.

The NAS workflow was based on the “Multi-Objective NAS with Ax” workflow tutorial on the official PyTorch website, utilizing Meta’s Ax package to do so. The process includes designing a search space as a separate python script that accepts variables that dictate the model structure, setting up a torchx runner and scheduler for submitting model training scripts concurrently, and defining optimization requirement configurations. Ax uses Bayesian optimization to evaluate and compare model configurations and their predictive accuracy, highlighting the impact specific architecture decisions have on the final loss.

As the BLEU score (Papineni et al., 2002) (reported on a scale of 0-100) was used as the primary metric in the original transformer paper (Vaswani et al., 2023), we used it as the evaluation metric for the NAS. The search space used for the machine translation task consisted of 6 adjustable parameters. Each of the four implemented positional encoding methods were able to be switched on or off, the dropout rate, and the activation function used in the feed-forward network were adjusted to optimize the BLEU score. Models were trained for 5 epochs during the search to reduce time complexity.

2.3 Code Availability

The source code for AttentionSmithy is publicly available on GitHub. The code implementing machine translation is also available and utilizes the WMT14 German-English dataset (Bojar et al., 2014) accessed through the Hugging Face datasets library. The code implementing geneformer (Theodoris et al., 2023) is publicly available as well, utilizing preprocessed data from the original geneformer implementation HuggingFace repository. All code repositories are released under the MIT license. The software originated from a re-implementation of code depicted in the Annotated Transformer article (Rush et al., 2022).

AttentionSmithy is implemented in Python using PyTorch (Ansel et al., 2024). To enhance usability and standardization, AttentionSmithy is designed to be compatible with PyTorch Lightning (Falcon, 2019), allowing researchers to easily incorporate training loops, distributed training, and other advanced features while maintaining clean, research-focused code.

2.4 LLM assistance

Claude 3.5 Sonnet and chatGPT o1 were used to help with writing this manuscript.

3 Results

3.1 Validation Studies

We conducted three validation studies to demonstrate the efficacy and versatility of AttentionSmithy: a replication of the original vanilla transformer model, an optimized model determined by a neural architecture search (NAS), and a bioinformatics application.

3.1.1 Original Transformer Replication

We implemented the transformer architecture as described by Vaswani et al. (2023) with two practical constraints: utilization of a single A100 GPU and a maximum context window of 100 tokens. These constraints were imposed due to computational resource limitations. Like the original paper, we trained on the WMT 2014 English-German dataset, which consists of 4.51M sentence pairs (approximately 9.03M sentences total). Due to our context window restriction, we had to truncate 63,227 sentences that exceeded 100 tokens. After 40 epochs of training, our vanilla implementation achieved a BLEU score of approximately 21 on the machine translation task [**Figure 2A,** red line]. While this falls short of the original paper's BLEU score of 25, it represents a reasonable achievement given the restricted context window and computational resources.

To highlight the ease of NAS enabled with AttentionSmithy, we designed a search space around the original model components to identify architectural changes that may enhance performance. The optimized model from NAS had a more rapidly increasing BLEU score across training steps, and in the end achieved a higher BLEU score of approximately 24 after 40 epochs [**Figure 2A**, blue line], approaching the performance of the original implementation despite our hardware constraints. Key modifications identified from NAS included: simultaneous utilization of all four positional encoding strategies (sinusoidal, learned, rotary, and ALiBi), removal of dropout (reduction from 0.1 to 0.0), and replacement of ReLU with tanh activation in feed-forward networks. This led to generally better translations, an example of which is shown in **Figure 2B**.

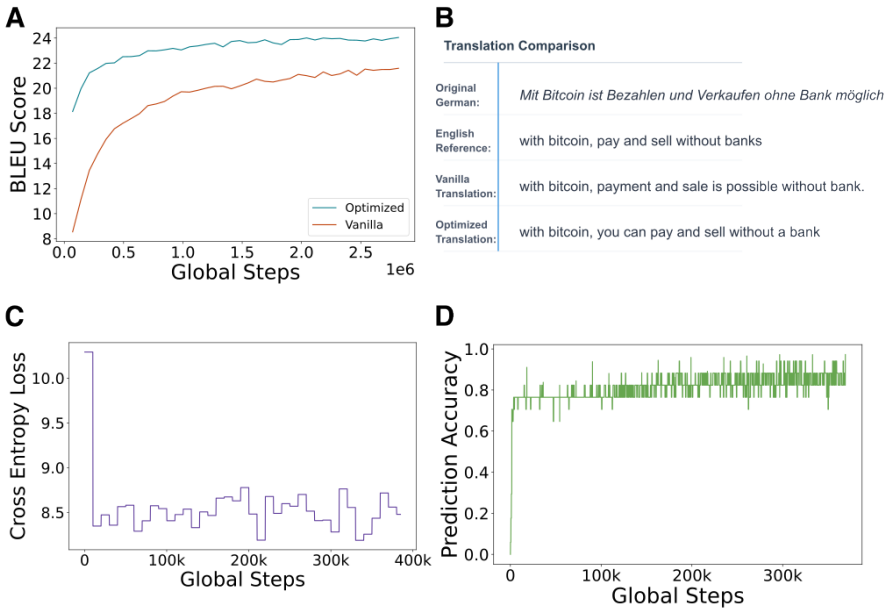


Figure 2: **Validation studies of programs built with AttentionSmithy.** (A) BLEU scores comparing translation quality between a vanilla transformer architecture (Vaswani et al., 2023) and an optimized transformer architecture derived through neural architecture search. (B) Representative examples of German-to-English translation outputs, showing source text, reference translation, and outputs from both transformer variants. The BLEU score in this specific example jumped from 16.1 for the vanilla translation to 34.4 in the optimized translation. (C) Validation loss trajectory during pretraining of the Geneformer foundation model plotted against global training steps. (D) Cell type classification accuracy on the validation set during fine-tuning of the pretrained Geneformer model, also plotted against global training steps.

3.1.2 Bioinformatics Application

To demonstrate domain adaptability, we replicated the Geneformer model for transfer learning for transcriptomic single-cell data tasks (Theodoris et al., 2023). Following the original paper's methodology in designing the model, we pre-trained it using a BERT-style architecture on rank-based transcript data, with genes serving as tokens. To verify that the model captures contextual information even after plateauing [**Figure 2C**], we fine-tuned it for cell type classification using this dataset, freezing the first two pre-trained layers and adding a classification layer as specified in their methodology. We used their published human_dcm_hcm_nf dataset for this task, which contains 579,159 cells representing 21 distinct cell types from cardiac tissue from 29 individuals. This implementation achieved over 95% accuracy on the validation dataset, demonstrating successful replication of the Geneformer architecture's ability to transfer contextual relationship information for downstream gene expression analyses [**Figure 2D**].

4 Discussion

The development and validation of AttentionSmithy reveals several important insights about transformer architecture implementation and customization. Our findings not only demonstrate the software package's effectiveness but also highlight novel approaches to transformer design that merit further investigation.

4.1 Efficacy of Combined Positional Encodings

Perhaps the most intriguing finding from our neural architecture search was the superior performance achieved through the simultaneous application of all four available positional encoding methods. While previous research has typically focused on comparing and contrasting different encoding strategies, our results

suggest that these methods may capture complementary aspects of positional information. This discovery opens new avenues for research into how different encoding strategies might interact and complement each other, potentially leading to more robust transformer architectures.

The modular implementation of positional encodings in AttentionSmithy extends beyond traditional position representation. Our framework enables the application of these encoding methods to any numeric data type, offering new possibilities for representing temporal, quantitative, or other ordered information within transformer architectures. For instance, in time-series analysis, one could simultaneously encode both sequential position and temporal information using different encoding strategies, potentially capturing both local and global patterns more effectively.

4.2 Implications for Domain-Specific Applications

While our validation studies focused on established architectures, they serve primarily to demonstrate AttentionSmithy's foundational reliability. The package's true value lies in enabling researchers to develop entirely new transformer architectures for specialized applications that may not yet exist. By providing a flexible, modular framework, we empower domain experts to experiment with novel combinations of transformer components without requiring deep expertise in transformer implementation details.

This capability is particularly valuable in scientific domains where traditional transformer architectures may not perfectly fit the underlying data structures or research questions. For instance, researchers working with complex multimodal data could leverage our framework to develop hybrid architectures that process different data types through specialized attention mechanisms. The ability to experiment with multiple positional encoding strategies simultaneously opens new possibilities for representing complex relationships in data, whether they be spatial, temporal, or domain-specific ordered relationships.

The modular nature of AttentionSmithy enables researchers to focus on the unique aspects of their application domains rather than becoming entangled in transformer implementation details. This democratization of transformer development has the potential to accelerate innovation in fields where artificial intelligence applications are still emerging. For example, researchers could apply self-supervised speech representation techniques (Mohamed et al., 2022) to nanopore sequencing, enabling efficient and accurate nucleotide sequencing through pre-training and fine-tuning approaches. In mass spectrometry, developing foundation models to interpret data-independent acquisition (DIA) spectra could allow researchers to leverage these complex, chimeric signals for downstream tasks without relying on pre-existing spectral libraries. The transformative potential of this architecture extends well beyond current applications, and we anticipate that researchers across diverse scientific domains will develop innovative implementations that we cannot yet foresee.

Future development of AttentionSmithy will focus on expanding its capabilities to support emerging transformer variants while maintaining its commitment to architectural clarity and ease of use. We encourage contributions from the research community, particularly in implementing new positional encoding strategies and exploring applications in specialized domains. This could include relative positional embeddings (Shaw et al., 2018) and their T5 variant (Raffel et al., 2023), which focus explicitly on the relationships between positions rather than absolute positions. Through continued development and collaboration, we aim to further lower the barriers to entry for transformer architecture experimentation and innovation across scientific disciplines.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu,

and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Xavier Amatriain, Ananth Sankar, Jie Bing, Praveen Kumar Bodigutla, Timothy J. Hazen, and Michaeel Kazi. Transformer models: an introduction and catalog. (arXiv:2302.07730), Mar 2024. doi: 10.48550/arXiv.2302.07730. URL http://arxiv.org/abs/2302.07730. arXiv:2302.07730 [cs].

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr 2024. doi: 10.1145/3620665.3640366. URL https://pytorch.org/assets/pytorch2-2.pdf.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. (arXiv:2004.05150), Dec 2020. doi: 10.48550/arXiv.2004.05150. URL http://arxiv.org/abs/2004.05150. arXiv:2004.05150 [cs].

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. Findings of the 2014 workshop on statistical machine translation. In Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia (eds.), *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 12–58, Baltimore, Maryland, USA, Jun 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3302. URL https://aclanthology.org/W14-3302/.

Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. Neural architecture search for transformers: A survey. *IEEE Access*, 10:108374–108412, 2022. ISSN 2169-3536. doi: 10.1109/ACCESS.2022.3212767.

Sanghyuk Roy Choi and Minhyeok Lee. Transformer architecture and attention mechanisms in genome data analysis: A comprehensive review. *Biology*, 12(77):1033, Jul 2023. ISSN 2079-7737. doi: 10.3390/biology12071033.

William Falcon. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995. ISBN 0-201-63361-2.

Md. Asraful Haque and Shuai Li. Exploring chatgpt and its impact on society. *AI and Ethics*, Feb 2024. ISSN 2730-5961. doi: 10.1007/s43681-024-00435-4. URL https://doi.org/10.1007/s43681-024-00435-4.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Jian Jiang, Lu Ke, Long Chen, Bozheng Dou, Yueying Zhu, Jie Liu, Bengong Zhang, Tianshou Zhou, and Guo-Wei Wei. Transformer technology in molecular science. *WIREs Computational Molecular Science*, 14 (4):e1725, 2024. ISSN 1759-0884. doi: 10.1002/wcms.1725.

Zexiang Liu, Dong Li, Kaiyue Lu, Zhen Qin, Weixuan Sun, Jiacheng Xu, and Yiran Zhong. Neural architecture search on efficient transformers and beyond. (arXiv:2207.13955), Jul 2022. doi: 10.48550/arXiv.2207.13955. URL http://arxiv.org/abs/2207.13955. arXiv:2207.13955 [cs].

Abdelrahman Mohamed, Hung-yi Lee, Lasse Borgholt, Jakob D. Havtorn, Joakim Edin, Christian Igel, Katrin Kirchhoff, Shang-Wen Li, Karen Livescu, Lars Maaløe, Tara N. Sainath, and Shinji Watanabe. Self-supervised speech representation learning: A review. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1179–1210, Oct 2022. ISSN 1932-4553, 1941-0484. doi: 10.1109/JSTSP.2022.3207050. arXiv:2205.10643 [cs].

Subhash Nerella, Sabyasachi Bandyopadhyay, Jiaqing Zhang, Miguel Contreras, Scott Siegel, Aysegul Bumin, Brandon Silva, Jessica Sena, Benjamin Shickel, Azra Bihorac, Kia Khezeli, and Parisa Rashidi. Transformers in healthcare: A survey. *Artificial Intelligence in Medicine*, 154:102900, Aug 2024. ISSN 09333657. doi: 10.1016/j.artmed.2024.102900. arXiv:2307.00067 [cs].

John Ousterhout. *A Philosophy of Software Design*. 1st edition, 2018. ISBN 1-73210-220-1.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin (eds.), *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA, Jul 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.

Narendra Patwardhan, Stefano Marrone, and Carlo Sansone. Transformers in the real world: A survey on nlp applications. *Information*, 14(44):242, Apr 2023. ISSN 2078-2489. doi: 10.3390/info14040242.

Gracile Astlin Pereira and Muhammad Hussain. A review of transformer-based models for computer vision tasks: Capturing global context and spatial relationships. (arXiv:2408.15178), Aug 2024. doi: 10.48550/arXiv.2408.15178. URL http://arxiv.org/abs/2408.15178. arXiv:2408.15178 [cs].

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. (arXiv:2108.12409), Apr 2022. doi: 10.48550/arXiv.2108.12409. URL http://arxiv.org/abs/2108.12409. arXiv:2108.12409.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. (arXiv:1910.10683), Sep 2023. doi: 10.48550/arXiv.1910.10683. URL http://arxiv.org/abs/1910.10683. arXiv:1910.10683 [cs].

Sasha Rush, Austin Huang, Suraj Subramanian, Jonathan Sum, Khalid Almubarak, and Stella Biderman. The annotated transformer, 2022. URL https://nlp.seas.harvard.edu/annotated-transformer/.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. (arXiv:1803.02155), Apr 2018. doi: 10.48550/arXiv.1803.02155. URL http://arxiv.org/abs/1803.02155. arXiv:1803.02155 [cs].

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. (arXiv:2104.09864), Nov 2023. doi: 10.48550/arXiv.2104.09864. URL http://arxiv.org/abs/2104.09864. arXiv:2104.09864.

Christina V. Theodoris, Ling Xiao, Anant Chopra, Mark D. Chaffin, Zeina R. Al Sayed, Matthew C. Hill, Helene Mantineo, Elizabeth M. Brydon, Zexian Zeng, X. Shirley Liu, and Patrick T. Ellinor. Transfer learning enables predictions in network biology. *Nature*, 618(7965):616–624, Jun 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06139-9.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. (arXiv:1706.03762), Aug 2023. doi: 10.48550/arXiv.1706.03762. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762.

Oliver Vogel, Ingo Arnold, Arif Chughtai, and Timo Kehrer. *Software Architecture: A Comprehensive Framework and Guide for Practitioners*. Springer Publishing Company, Incorporated, 2011. ISBN 3-642-19735-3.

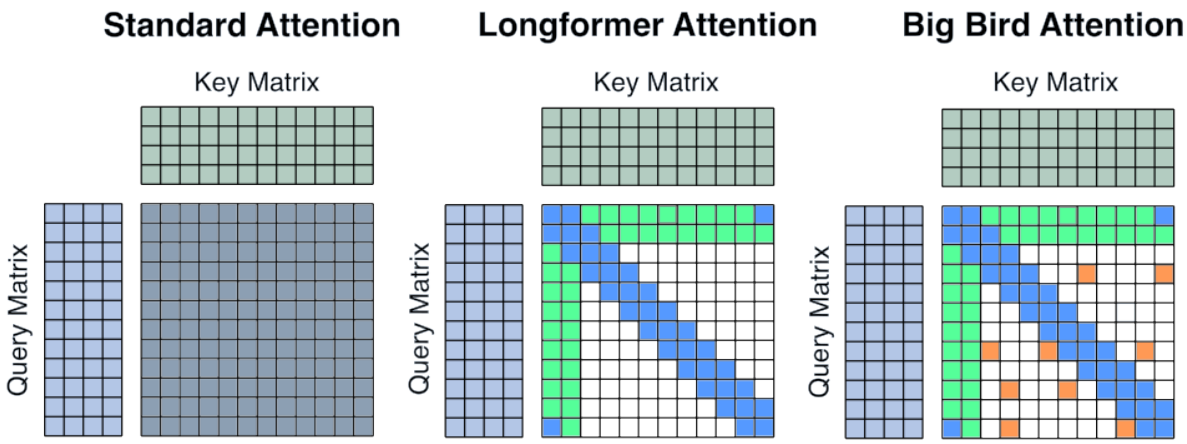
Yu-An Wang and Yun-Nung Chen. What do position embeddings learn? an empirical study of pre-trained language model positional encoding. (arXiv:2010.04903), Oct 2020. doi: 10.48550/arXiv.2010.04903. URL http://arxiv.org/abs/2010.04903. arXiv:2010.04903 [cs].

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. (arXiv:1910.03771), Jul 2020. doi: 10.48550/arXiv.1910.03771. URL http://arxiv.org/abs/1910.03771. arXiv:1910.03771 [cs].

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. (arXiv:2007.14062), Jan 2021. doi: 10.48550/arXiv.2007.14062. URL http://arxiv.org/abs/2007.14062. arXiv:2007.14062.

A Appendix

A.1 Supplementary Figures



Supplementary Figure 1: **Extendable attention mechanism implementation of AttentionSmithy.** While no alternates are yet implemented, the code is designed to allow alternate attention mechanisms that address various shortcomings of the original method. Two examples for future implementation include the Longformer attention method and Big Bird attention method. Both are designed to extend the allowable context window, a major bottleneck in transformer-based models. This is done by creating a sparse attention matrix only utilizing global and local tokens (both methods) as well as randomly selected tokens (Big Bird only).

A.2 Supplementary Code

The included supplementary `.zip` file (`machine-translation.zip`) contains example code demonstrating the implementation and execution of a machine translation transformer model using AttentionSmithy. The included scripts facilitate data downloading, model training, and evaluation.

A.2.1 Contents

- `data_download.py` – Downloads and preprocesses the WMT-14 German-English dataset.

- `data_import.py` – Handles dataset loading and processing for training.
- `main.py` – Runs the model training and evaluation pipeline.
- `model_import.py` – Defines the translation model and its components.
- `README.md` – Provides setup instructions, expected outputs, and additional notes.