

TokenSmith: Streamlining Data Editing, Search, and Inspection for Large-Scale Language Model Training and Interpretability

Mohammad Aflah Khan^{1*}, Ameya Godbole^{2*}, Johnny Tian-Zheng Wei², Ryan Wang², James Flemings², Krishna Gummadi¹, Willie Neiswanger², Robin Jia²

¹Max Planck Institute for Software Systems, ²University of Southern California

Correspondence: afkhan@mpi-sws.org, ameyagod@usc.edu

Abstract

Understanding the relationship between training data and model behavior during pretraining is crucial, but existing workflows make this process cumbersome, fragmented, and often inaccessible to researchers. We present TokenSmith, an open-source library for interactive editing, inspection, and analysis of datasets used in Megatron-style pretraining frameworks such as GPT-NeoX, Megatron, and NVIDIA NeMo. TokenSmith supports a wide range of operations including searching, viewing, ingesting, exporting, inspecting, and sampling data, all accessible through a simple user interface and a modular backend. It also enables structured editing of pretraining data without requiring changes to training code, simplifying dataset debugging, validation, and experimentation. TokenSmith is designed as a plug and play addition to existing large language model pretraining workflows, thereby democratizing access to production-grade dataset tooling.

TokenSmith is hosted on GitHub¹, with accompanying documentation and tutorials². A demonstration video is also available on YouTube.³

1 Introduction

The barrier to pretraining large language models from scratch has been rapidly declining, driven by improved access to GPUs, a growing number of open-source frameworks, and the widespread sharing of technical knowledge. As a result, academic groups, open-source organizations and hobbyists are increasingly able to conduct meaningful pretraining research (Biderman et al., 2023; Azerbayev et al., 2023; Chi et al., 2023; Yin et al., 2023; Gupta et al., 2023; Horawalavithana et al., 2022; Ibrahim et al., 2024; Gao et al., 2025a,b; Zeng et al., 2024).

However, a persistent challenge across existing frameworks is the lack of robust tooling for inspecting and interacting with the training data. Tasks such as debugging loss spikes by tracing relevant datapoints, generating modified datasets for counterfactual experiments or decontamination, and even viewing specific batches or sequences remain cumbersome in open-source setups. For example, producing a counterfactual dataset typically requires manually identifying files, ensuring token alignment, and re-tokenizing the entire corpus (a process that can take over a day for large datasets).

We introduce TokenSmith, a toolkit designed to make this process seamless. TokenSmith addresses these gaps by providing intuitive abstractions for editing, inspecting, and managing datasets, thereby enabling faster iteration and deeper insight throughout the pretraining workflow. TokenSmith is built on top of Megatron-LM (Shoeybi et al., 2020), a widely adopted and scalable framework for large language model pretraining. Notably, popular libraries such as GPT-NeoX (Andonian et al., 2023) and NVIDIA NeMo⁴ are also based on Megatron-LM and follow a common dataset format and training pipeline structure. This shared foundation allows TokenSmith to natively support all three frameworks with minimal integration overhead. Additionally, TokenSmith is designed to be extensible, making it easy to add support for other frameworks.

2 Library Offerings

We build TokenSmith for practitioners and researchers working directly with large-scale language model pretraining. Our goal is to make it significantly easier to address a wide range of research questions and engineering challenges that arise when working with massive datasets and opaque training processes. Instead of offering a monolithic

*Equal contribution

¹<https://github.com/aflah02/tokensmith>

²<https://aflah02.github.io/tokensmith/>

³<https://www.youtube.com/watch?v=cD08VE9fZvU>

⁴<https://github.com/NVIDIA/NeMo>

interface, we emphasize modularity and extensibility, providing intuitive abstractions through a simple frontend and a well-documented backend that can be adapted to different workflows. Here we describe the key functionalities we support: Inspect, Sample, Edit, Export, Ingest, and Search.

2.1 Inspecting and Sampling Datasets

Understanding the relationship between training data and model behavior is a recurring challenge in large-scale pretraining. Practitioners often need to trace issues back to specific sequences or isolate dataset subsets for hypothesis testing. Typical questions include:

- *How can we trace and identify sequences that correlate with sudden spikes in training loss?*
- *Between two model checkpoints, what new data did the model encounter, and how might it explain improvements or regressions in performance?*
- *Are there tokenization issues or formatting inconsistencies that may have gone unnoticed?*
- *What happens if the model is trained only on a specific subset, such as domain-specific documents or repeated early-stage data?*
- *Can we sample sequences based on properties such as length, content patterns, or document metadata?*

TokenSmith provides a unified set of tools to support both deep inspection and flexible sampling:

- **Precise inspection utilities** allow users to locate and analyze data at the level of individual sequences, batches, or training steps using indices or global step numbers.
- **Sampling utilities** support extracting subsets of the data based on custom policies, enabling rapid prototyping, ablation studies, and behaviorally targeted dataset construction.
- **Modular integration** through a backend API allows seamless incorporation into training pipelines, analysis scripts, or interactive UIs.

By enabling structured, reproducible interrogation of the training dataset, TokenSmith empowers practitioners to move from anecdotal debugging to systematic, data-driven understanding of model behavior.

2.2 Editing Datasets

As training progresses or evaluation findings emerge, practitioners frequently encounter the need to modify the dataset. These changes are often motivated by new insights or requirements, such as:

- *Removing specific batches that correlate with spikes in training loss*
- *Filtering out examples that may result in test set leakage or data contamination*
- *Creating counterfactual variants of the dataset for controlled experiments, such as ablation studies or robustness analysis*

To support such use cases, TokenSmith offers a flexible editing interface that allows for **targeted edits**, enabling users to directly specify and modify individual sequences.

These capabilities allow researchers to iterate on dataset versions without re-engineering the training pipeline. Edits can be performed programmatically and are fully compatible with the inspection and sampling modules of the library. This makes it possible to run sophisticated data-centric experiments, such as testing the effect of subtle perturbations, while maintaining full control over what the model sees during training.

2.3 Exporting Datasets

Beyond sampling, reproducibility and interoperability are essential in dataset-centric research. Some recurring challenges are:

- *How can we verify and share a specific version of the dataset used for a paper, in a reproducible format compatible with popular libraries like HuggingFace Datasets?*
- *Can we export only specific batches/sequences of a dataset that show interesting trends to avoid sharing large binaries?*

To this end, TokenSmith includes **export tools** for converting datasets (entirely or in parts) into formats such as JSONL and CSV which are also HuggingFace compatible. These tools enable seamless sharing, integration with external pipelines, and long-term reproducibility of experimental results.

2.4 Ingesting Datasets

Curating datasets for large-scale pretraining often begins with converting diverse data sources into a format compatible with Megatron-style frameworks. However, this step is frequently under-documented and error-prone. A common challenge is: *How can we ingest and tokenize new datasets into the Megatron binary format without writing custom conversion pipelines?*

TokenSmith addresses this through streamlined **ingestion utilities** that support converting standard formats such as JSONL and CSV into the required .bin/.idx representation. These tools reduce the overhead of dataset preparation and ensure seamless compatibility with Megatron-based pretraining workflows.

2.5 Searching Datasets

As pretraining datasets grow in size and complexity, being able to efficiently search and retrieve relevant content becomes essential for both debugging and targeted experimentation. Practitioners and researchers often encounter challenges such as:

- *How can we locate all occurrences of a specific phrase, token, or n-gram to inspect or remove sensitive or duplicate content?*
- *Can we curate the likely continuations given a naive n-gram model to contrast the generation likelihoods of our LLMs?*
- *Is it possible to trace model behaviors to specific textual patterns or domains within the training set?*
- *How do we efficiently support search at scale without loading the entire dataset into memory?*

To address these challenges, TokenSmith builds abstractions over Tokengram⁵, an efficient n-gram indexing and search tool optimized for large-scale corpora. This allows users to perform fast searches over pre-tokenized corpora. Integrating Tokengram in TokenSmith provides support for end-to-end data interventions. For example, the search results can be processed with the Inspect and Export utilities for easy sharing with your collaborators or downstream post-processing. The matched documents from the search results can be processed with the

⁵<https://github.com/EleutherAI/tokengrams>

Edit utilities. This might be useful to mask out toxic text, anonymize documents in place, etc.

By making dataset search fast and programmatically accessible, TokenSmith empowers users to build more informed training sets, track down model behaviors to specific training signals, and conduct controlled data-centric research at scale.

3 Practical Case Studies

In this section, we provide examples of how TokenSmith could simplify pipelines for NLP research.

3.1 Training Dynamics of Memorization

Several research groups have attempted to study memorization of natural (Huang et al., 2024; Jagielski et al., 2023) or counterfactually curated (Chang et al., 2025; Wei et al., 2024) data during LM pre-training. In order to study training dynamics, these projects relied on one of two approaches:

1. **Re-tokenizing the corpus:** Wei et al. (2024) studied the memorization of watermarks in pre-training by injecting randomized strings in groups of documents. They re-tokenize the entire corpus along with different sets of watermarked documents. Note that the number of modified documents is a small fraction of the full corpus; thus, they spend considerable time re-tokenizing unchanged data. Moreover, their approach cannot control the order of unchanged documents in different training runs.
2. **Modifying the training library:** Huang et al. (2024) and Chang et al. (2025) study verbatim memorization of documents in the early, middle, and late stages of pre-training by injecting curated/synthetically generated documents in specific training sequences. They achieve this by modifying the data loader and training loop of the pre-training library (Andonian et al., 2023; Team OLMo et al., 2024). This engineering-intensive approach requires a deep understanding of the underlying pre-training libraries. Moreover, this may decrease the training efficiency of the library.

In contrast, TokenSmith allows you to directly edit the tokenized dataset (§ 2.2). This allows you to perform the same experiments (1) without having to re-tokenize documents that haven’t changed between training runs, and (2) without modifying the pre-training libraries.

3.2 Identifying Causes of Instability

Team OLMo et al. (2024) highlight that sudden spikes in training loss can lead to instability further along in pre-training and worse final model performance. In order to debug the cause of the instability, they inspect the batches of data that caused the spikes and identify that the batches contain training sequences with repeated n-grams. The Inspect and Export tools in TokenSmith provide a straightforward interface to extract batches based on the step number (where the loss spike occurred).

4 Library Design

TokenSmith is designed to support two complementary modes of interaction: a Pythonic API for seamless integration into existing training or analysis pipelines, and a visual UI for interactive exploration and inspection.

4.1 Overview: Megatron Dataset Format

Megatron-LM’s indexed format uses two files per data split. The .bin file contains raw token sequences (packed back-to-back) as a flat array of integers. The .idx file contains metadata and pointers into the .bin. Specifically, the index begins with a header (version, dtype, number of sequences, number of documents) and then lists, for each sequence, its length (number of tokens) and its byte offset in the .bin. It also records, for each document, which sequences belong to it. In effect, .idx lets the dataset class reconstruct which slice of the big token array corresponds to each example.

The consistency of this format across libraries allows TokenSmith to support them out-of-the-box with minimal adaptations, enabling seamless interoperability and inspection without requiring separate data handling logic for each framework.

4.2 Pythonic API

TokenSmith exposes a modular, object-oriented API that allows users to programmatically ingest, edit, search, sample, inspect, and export datasets. This API is well-suited for integration into training scripts, research notebooks, or batch processing workflows. Figures 1, 2, 3, 4 and 5 illustrate the core API patterns:

- **Figure 1** illustrates how users can configure and execute token-level search queries using a Tokengram-backed index over the dataset.

- **Figures 2 and 3** showcase the interfaces for inspecting specific sequences and sampling data according to user-defined policies.
- **Figure 4** presents the editing interface, which supports fine-grained, targeted modifications to existing sequences.
- **Figure 5** demonstrates the dataset ingestion and export utilities, which allow users to import new corpora and export subsets or modified datasets in standard formats.

The library is designed to latch onto your existing pretraining environments with minimal configuration (instructions outlined clearly in the README).⁶

```
from tokensmith.manager import DatasetManager
dataset_manager = DatasetManager()

dataset_manager.setup_search(
    bin_file_path="tokenized_data.bin",
    search_index_save_path="search_index_tokenized_data.idx",
    vocab=2**16,
)

tokenized_string = [67, 45, 99] # List of tokens

count = dataset_manager.search.count(test_sample)
isPresent = dataset_manager.search.contains(test_sample)
positions = dataset_manager.search.positions(test_sample)
```

Figure 1: Search API

```
from tokensmith.manager import DatasetManager
dataset_manager = DatasetManager()

dataset_manager.setup_edit_inspect_sample_export(
    dataset_prefix='data_tokenized_text_document',
    batch_info_save_prefix='batch_info',
    train_iters=100, train_batch_size=16, train_seq_len=2048, seed=42,
)

sample_0 = dataset_manager.inspect.inspect_sample_by_id(sample_id=0)

batch_0 = dataset_manager.inspect.inspect_sample_by_batch(
    batch_id=0, batch_size=4
)
```

Figure 2: Inspect API

4.3 Interactive UI

To enable visual exploration and rapid debugging, TokenSmith provides an intuitive user interface built with Streamlit.⁷ The UI serves both as a reference implementation and a customizable layer for users to extend based on their workflows. It supports interactive search, batch and sequence level inspection, and document viewing. Figures 6a and

⁶<https://github.com/aflah02/tokensmith?tab=readme-ov-file#-quick-start>

⁷<https://streamlit.io/>

```

from tokensmith.manager import DatasetManager

dataset_manager = DatasetManager()

dataset_manager.setup_edit_inspect_sample_export(
    dataset_prefix='data_tokenized_text_document',
    batch_info_save_prefix='batch_info',
    train_iters=100, train_batch_size=16, train_seq_len=2048, seed=42,
)

def sparse_sample_policy(start_index, num_samples, step_size=10):
    return [start_index + i * step_size for i in range(num_samples)]

def fibonacci_batch_policy(max_batch_id):
    fib = [1, 1]
    while fib[-1] < max_batch_id:
        fib.append(fib[-1] + fib[-2])
    return [f for f in fib if f < max_batch_id]

sparse_samples = dataset_manager.sample.get_samples_by_policy(
    policy_fn=sparse_sample_policy, start_index=50,
    num_samples=4, step_size=25
)

fib_batches = dataset_manager.sample.get_batches_by_policy(
    policy_fn=fibonacci_batch_policy, batch_size=2, max_batch_id=25,
)

```

Figure 3: Sample API

```

from tokensmith.manager import DatasetManager

dataset_manager = DatasetManager()

dataset_manager.setup_edit_inspect_sample_export(
    dataset_prefix='data_tokenized_text_document',
    batch_info_save_prefix='batch_info',
    train_iters=100, train_batch_size=16, train_seq_len=2048, seed=42,
)

dataset_manager.edit.inject_and_preview(
    text='This is a test sentence', tokenizer=tokenizer,
    injection_loc=20, injection_type='seq_shuffle', add_eos_token=True,
    dry_run=True # Set to False to not only preview but also perform
)

```

Figure 4: Edit API

```

from tokensmith.manager import DatasetManager

dataset_manager = DatasetManager()

dataset_manager.ingest.ingest_from_jsonl(
    input_jsonl_path='data.jsonl', output_prefix='data_tokenized',
    vocab_path='tokenizer.json', neox_dir='gpt-neox',
    workers=8, append_eod=True, dataset_impl='mmap',
    tokenizer_type='HFTokenizer',
)

dataset_manager.setup_edit_inspect_sample_export(
    dataset_prefix='data_tokenized_text_document',
    batch_info_save_prefix='artifacts/batch_info',
    train_iters=100, train_batch_size=16, train_seq_len=2048, seed=42,
)

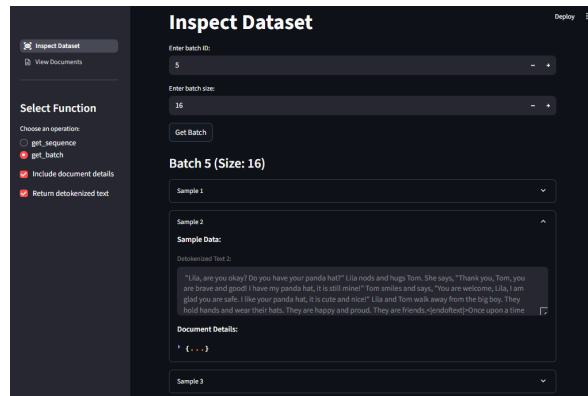
manager.export.export_sequences(
    sequence_indices=[100, 200, 300],
    output_path='exports/sequences.csv', format_type='csv',
)

manager.export.export_batch_range(
    start_batch=5, end_batch=10, # Exports batches 5-9
    batch_size=32, output_path='batch_range.csv', format_type='csv',
    return_detokenized=True, tokenizer=tokenizer, flatten_batches=True
)

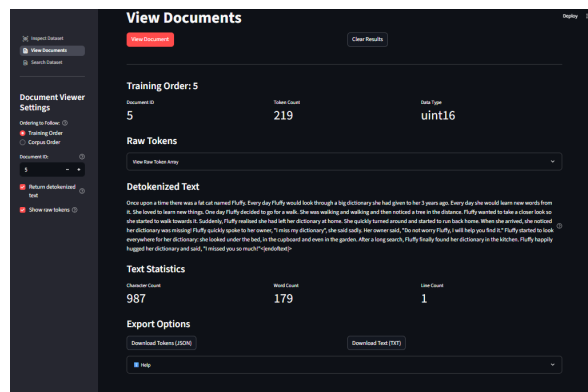
```

Figure 5: Ingest and Export API

6b show examples of the inspect and document viewing pages, while Figure 8 (Appendix C) illustrates different search modes. Users can browse individual sequences or batches to trace issues such as loss spikes, search for specific phrases, locate



(a) Inspect UI showing detailed information for a selected batch, including tokenized and detokenized sequences and document metadata.



(b) View Document UI for browsing individual documents and their tokenized representations.

Figure 6: TokenSmith inspection and viewing interfaces for exploring dataset contents at the batch and document levels.

them within documents, and explore next token distributions, all through simple point and click interactions.

Together, the API and UI provide a unified and flexible interface to dataset management, enabling both hands-on experimentation and automated workflows at scale.

4.4 Design Patterns

TokenSmith is built with a strong emphasis on clean software engineering to ensure ease of use, extensibility, and long-term maintainability. Its architecture follows established design patterns to provide a clear separation of concerns, enable safe experimentation, and support both research and production environments. These design choices also make it easier for contributors to extend and integrate the toolkit with custom workflows. A detailed breakdown of the patterns used, including handler-based modularity, a facade interface, and runtime configurability, is provided in Appendix B.

5 Benchmarking

While there are no established baselines for many of the operations supported by TokenSmith we compare against the de facto workflows that practitioners currently rely on to achieve similar outcomes. These comparisons highlight the complexity and overhead of existing approaches, and demonstrate how TokenSmith streamlines them.

- **Editing a dataset to produce a counterfactual version:** Consider the case where a researcher wants to generate a modified dataset that differs by only a few examples from an original corpus spanning hundreds of billions of tokens.
 - *Current workflow:* Manually identify and replace relevant files, ensure token alignment (if needed), and re-tokenize the entire dataset. This process is brittle and time-consuming; for example, tokenizing a 500B-token corpus can take over a day depending on the system configuration.
 - *With TokenSmith:* Users can programmatically perform targeted or randomized edits directly on the tokenized dataset using our editor interface. This removes the need to reason about token boundaries or initiate a full re-tokenization pass, significantly reducing iteration time and engineering overhead.
- **Sampling Sequences According to Custom Policies:**
 - *Without TokenSmith :* Practitioners must manually extract sequences from the binary files, align them with training indices, implement policy-based filtering logic, and reconstruct the final subset (often requiring non-trivial changes to the existing pipeline).
 - *With TokenSmith :* The sampling API abstracts away these complexities, allowing users to specify high-level parameters and a custom policy function to obtain the desired subset with minimal effort.

We also examine the scalability of our system by measuring the time taken for three representative operations as dataset size increases.

5.1 Dataset Setup

We measure the time required to execute the `setup_edit_inspect_sample_export` method, which initializes all necessary handlers based on

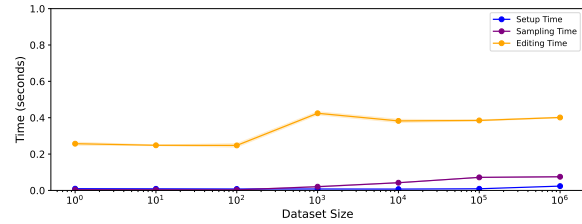


Figure 7: Execution time for setup, sampling, and editing operations across varying dataset sizes. For sampling and editing, the reported times correspond to 100 operations.

the user’s configuration. This step introduces negligible overhead, remaining under 0.03 seconds even for corpora with one million documents.

5.2 Sampling Performance

We randomly sample 100 sequence indices and measure the total retrieval time. This process is repeated five times, and we report the average and standard deviation. A single-item fetch is performed beforehand to warm up the system. Sampling remains stable across dataset sizes. For example, sampling 100 sequences from a 1M-document corpus takes under 0.1 seconds.

5.3 Editing Time

To assess editing performance, we insert the sentence (This is a test sentence.) at 100 random positions across the dataset, following an initial warm-up edit. This operation is repeated five times, with average time and standard deviation reported. Edit latency remains under 0.5 seconds even for a 1M-document corpus, indicating minimal sensitivity to dataset size.

We showcase the benchmarking results in Figure 7. These results highlight that TokenSmith offers low-latency interactivity, making it well-suited for both rapid experimentation and large-scale dataset manipulation. For search-related benchmarks, we refer readers to Tokengrams’ evaluations,⁸ as TokenSmith directly integrates Tokengrams for all token-level search operations.

Before each benchmark measurement, we re-instantiate the `DatasetManager` and explicitly trigger garbage collection using `gc.collect()` from Python’s `gc` module.⁹ The benchmarking script, along with the corresponding results, is available

⁸<https://github.com/EleutherAI/tokengrams?tab=readme-ov-file#performance>

⁹<https://docs.python.org/3/library/gc.html>

in the repository.¹⁰

6 Conclusion

We present TokenSmith, a modular and extensible toolkit designed to streamline dataset-centric workflows in Megatron-style LLM pretraining. By offering intuitive abstractions for ingesting, editing, inspecting, sampling, searching, and exporting training data, TokenSmith fills a crucial gap in current open-source infrastructure. TokenSmith’s support for multiple backends, efficient operations at scale, and dual interface (Pythonic API and visual UI) makes it accessible to researchers, practitioners, and hobbyists alike. As the LLM ecosystem increasingly embraces open and reproducible research, we believe TokenSmith will serve as a practical foundation for understanding, debugging, and experimenting with the data that drives modern language models.

Acknowledgments

We thank the EleutherAI team for open-sourcing Tokengrams and GPT-NeoX, and for their helpful responses to our questions. We also acknowledge the contributions of NVIDIA’s Megatron and GPT-NeoX repositories, which serve as foundational components in our work. This work was supported in part by a NAIRR Pilot Award of NVIDIA DGX Cloud services, a gift from the USC-Amazon Center on Secure and Trusted Machine Learning, and the National Science Foundation under Grant No. IIS-2403436. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Finally, we note that large language models were used to assist in editing and refining the writing of this paper.

References

Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Jason Phang, Shivanshu Purohit, Hailey Schoelkopf, Dashiell Stander, Tri Songz, Curt Tigges, Benjamin Thérien, and 2 others. 2023. [GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch](#).

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. 2023. [Proofnet: Autoformalizing and formally proving undergraduate-level mathematics](#). *Preprint*, arXiv:2302.12433.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O’Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. *Pythia: a suite for analyzing large language models across training and scaling*. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org.

Hoyeon Chang, Jinho Park, Seonghyeon Ye, Sohee Yang, Youngkyung Seo, Du-Seong Chang, and Minjoon Seo. 2025. How do large language models acquire factual knowledge during pretraining? In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS ’24*, Red Hook, NY, USA. Curran Associates Inc.

Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. 2023. [Dissecting transformer length extrapolation via the lens of receptive field analysis](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13522–13537, Toronto, Canada. Association for Computational Linguistics.

E. Gamma. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Pearson Education.

Tianyu Gao, Alexander Wettig, Luxi He, Yihe Dong, Sadhika Malladi, and Danqi Chen. 2025a. [Metadata conditioning accelerates language model pre-training](#). *Preprint*, arXiv:2501.01956.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. 2025b. [How to train long-context language models \(effectively\)](#). *Preprint*, arXiv:2410.02660.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L. Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. 2023. [Continual pre-training of large language models: How to \(re\)warm your model?](#) *Preprint*, arXiv:2308.04014.

Sameera Horawalavithana, Ellyn Ayton, Shivam Sharma, Scott Howland, Megha Subramanian, Scott Vazquez, Robin Cosbey, Maria Glenski, and Svitlana Volkova. 2022. [Foundation models of scientific knowledge for chemistry: Opportunities, challenges and lessons learned](#). In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 160–172, virtual+Dublin. Association for Computational Linguistics.

Jing Huang, Diyi Yang, and Christopher Potts. 2024. [Demystifying verbatim memorization in large language models](#). *Preprint*, arXiv:2407.17817.

¹⁰<https://github.com/aflah02/TokenSmith/tree/main/benchmarking>

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L. Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. 2024. [Simple and scalable strategies to continually pre-train large language models](#). *Preprint*, arXiv:2403.08763.

Matthew Jagielski, Om Thakkar, Florian Tramer, Daphne Ippolito, Katherine Lee, Nicholas Carlini, Eric Wallace, Shuang Song, Abhradeep Guha Thakurta, Nicolas Papernot, and Chiyuan Zhang. 2023. [Measuring forgetting of memorized training examples](#). In *The Eleventh International Conference on Learning Representations*.

R.C. Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Alan Apt series. Pearson Education.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. [Megatron-Lm: Training multi-billion parameter language models using model parallelism](#). *Preprint*, arXiv:1909.08053.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Taffjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, and 21 others. 2024. [2 OLMo 2 Furious](#). *Preprint*, arXiv:2501.00656.

Johnny Wei, Ryan Wang, and Robin Jia. 2024. [Proving membership in LLM pretraining data via data watermarks](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13306–13320, Bangkok, Thailand. Association for Computational Linguistics.

Junqi Yin, Sajal Dash, Feiyi Wang, and Mallikarjun Shankar. 2023. [Forge: Pre-training open foundation models for science](#). In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '23*, New York, NY, USA. Association for Computing Machinery.

Zhiyuan Zeng, Qipeng Guo, Zhaoye Fei, Zhangyue Yin, Yunhua Zhou, Linyang Li, Tianxiang Sun, Hang Yan, Dahua Lin, and Xipeng Qiu. 2024. [Turn waste into worth: Rectifying top-k router of moe](#). *Preprint*, arXiv:2402.12399.

A Licensing

TokenSmith is released under the Apache 2.0 license. This permissive license allows for both academic and commercial use, as well as modification and redistribution, making it suitable for a wide range of research and production workflows.

B Design Patterns Employed in the Library

TokenSmith is structured around well-established software design principles that promote modular-

ity, extensibility, and maintainability. The internal architecture is intentionally clean and componentized to accommodate both research prototyping and production-scale workflows. Below, we describe the primary design patterns used in the codebase.

Handler Pattern (Command or Service Object) (Gamma, 1995) Each major functional area (editing, inspecting, sampling, exporting, and searching) is encapsulated within its own handler class. For instance, `EditHandler`, `InspectHandler`, `SampleHandler`, `ExportHandler`, and `SearchHandler` each manage their domain-specific logic while exposing a consistent interface. This clear separation of concerns makes the system easier to extend, test, and reason about.

Facade Pattern (Gamma, 1995) The `DatasetManager` class serves as a unified entry point to the system. It orchestrates the initialization of handlers and provides a high-level API to the end user. This shields users from internal complexities and reduces the cognitive load involved in accessing multiple capabilities.

Dependency Injection Rather than relying on tight coupling or global state, handlers receive references to the `DatasetManager` or its specific components during initialization. This inversion of control enhances testability and supports future decoupling and modular reuse.

Type Hinting and Forward References To avoid circular dependencies while retaining strong type safety, the library uses `TYPE_CHECKING` blocks and string-based type annotations (e.g., `'DatasetManager'`). This allows static analyzers and IDEs to provide full support while maintaining clear dependency boundaries.¹¹

Strategy Pattern (Configurable Behavior) (Gamma, 1995) Handlers expose methods whose behavior can be configured at runtime via parameters. For example, the `EditHandler` supports multiple injection strategies. This makes the system adaptable for experimentation without requiring internal changes to core logic.

Template Method Pattern (Gamma, 1995) Export operations follow a template structure in which base methods (such as `export`) provide a standard workflow but allow subclasses or extensions to

¹¹<https://peps.python.org/pep-0484/>

override certain steps. This approach encourages consistent behavior while allowing flexibility for future extensions or format support.

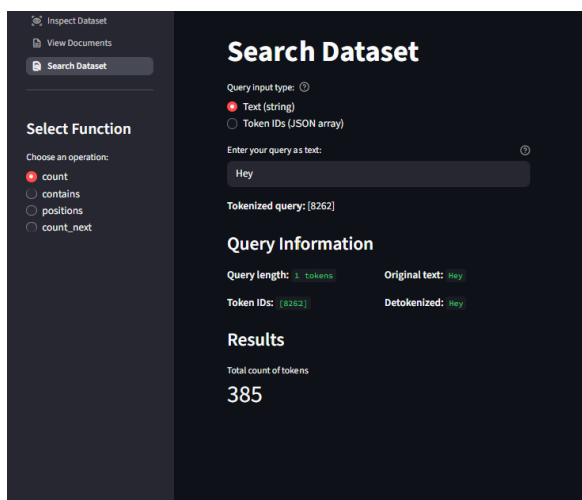
Validation and Defensive Programming Robust input validation and error handling are systematically applied across the codebase. Although not a formal design pattern, this practice contributes significantly to the reliability and maintainability of the library, especially in high-scale or adversarial settings.

Modular Package Structure The code is divided into clearly defined submodules (edit, inspect, sample, export, search), with each exposing a single handler class through its `__init__.py`. This supports the Single Responsibility Principle ([Martin, 2003](#)) and allows contributors to quickly locate, understand, and extend functionality.

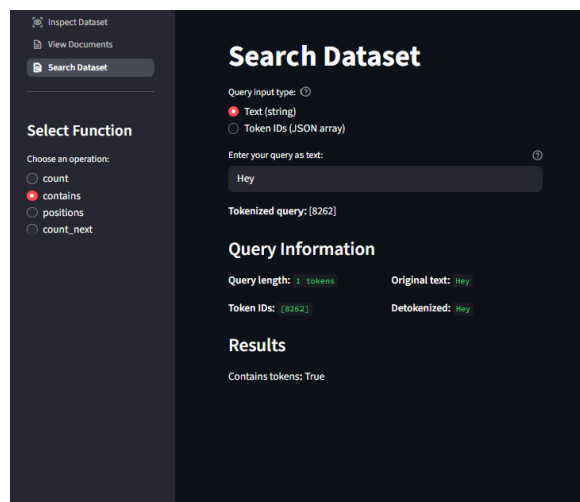
These design decisions collectively ensure that TokenSmith remains extensible and maintainable as it grows to support additional backends, workflows, and research use cases.

C User Interface

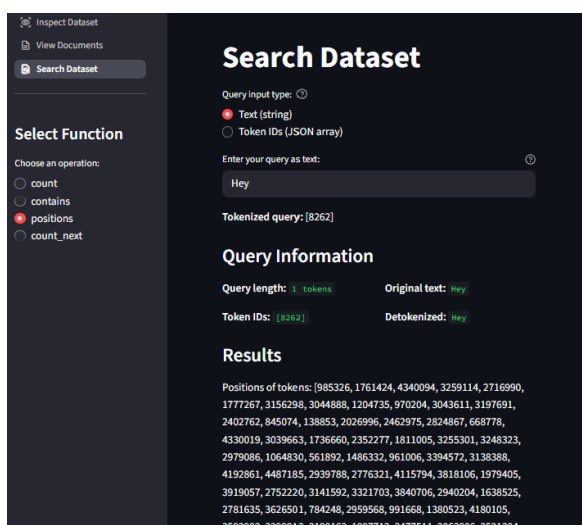
Figure 8 showcases different components of the TokenSmith search interface, illustrating how users can query token counts, presence, positions of occurrence, and likely next tokens using an n-gram model—all through intuitive visual tools.



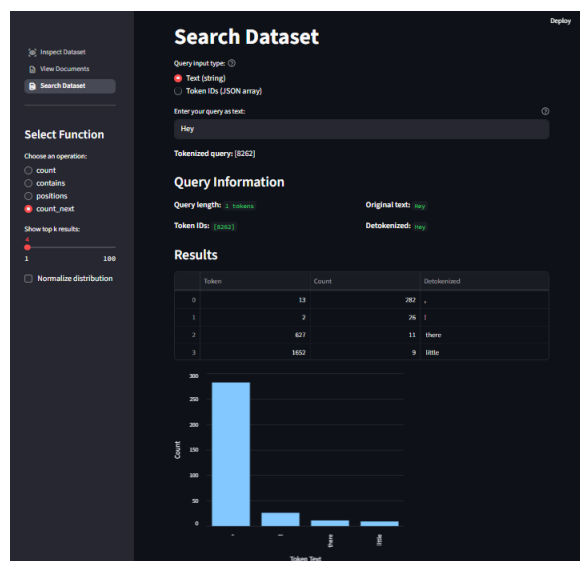
(a) Search for count



(b) Search for presence



(c) Search for positions of occurrence



(d) Search for likely next tokens using an n-gram model

Figure 8: TokenSmith search UI showing different search strategies: filtering by count, containment, positional match, and context-based continuation.