LumiXAI: A Modular Full-Stack Framework for Feature Attribution

Alfio Ferrara[1] **and Lorenzo Gatta**[2] **and Sergio Picascia**[2] **and Elisabetta Rocchetti**[2]

[1]Department of Literary Studies, Philology and Linguistics, Università degli Studi di Milano, Milan, Italy
[2]Department of Computer Science, Università degli Studi di Milano, Milan, Italy

alfio.ferrara@unimi.it, lorenzo.gatta@studenti.unimi.it
sergio.picascia@unimi.it, elisabetta.rocchetti@unimi.it

Abstract

Feature attribution is a central tool of model interpretability, yet the software through which it is applied remains fragmented: individual tools specialize along narrow axes, such as a single modality, a code API or a GUI, or a fixed rather than extensible method set, and rarely combine these strengths. Moreover, many explainability tools are designed primarily for domain experts, requiring programming skills or familiarity with attribution methods that can make them difficult for non-expert users to access. In this article, we present **LumiXAI**, a modular full-stack framework that consolidates attribution analysis into a single system. It couples classification and generative attribution with an interactive GUI supporting bidirectional exploration, a plug-in architecture for registering new models and methods, and three access tiers serving non-programmers, developers, and extenders from one backend. Its contribution is a system that operationalises established attribution methods under one interface, one interaction model, and one persistence layer, with containerised services and persistent results making analyses reproducible across machines.

1 Introduction

As machine learning models are increasingly deployed in consequential settings, understanding *why* a model produces a given output has become as important as the output itself. Feature attribution addresses this by assigning importance scores to input elements based on their contribution to a prediction or generation. As attribution has grown into a broad methodological family spanning gradient-based, perturbation-based, and attention-based techniques, a practical question arises: how can practitioners navigate, explore, and systematically evaluate these methods?

The tooling through which attribution is applied has developed unevenly across the field: individual tools tend to specialize along a narrow set of axes, such as scope, kind of interaction, or extensible method set, and rarely combine these strengths altogether. A researcher who needs attribution across several model types and interaction styles must therefore assemble and reconcile multiple tools with incompatible interfaces, output formats, and deployment assumptions, an integration burden that impedes precisely the reproducible, exploratory analysis that attribution is meant to support.

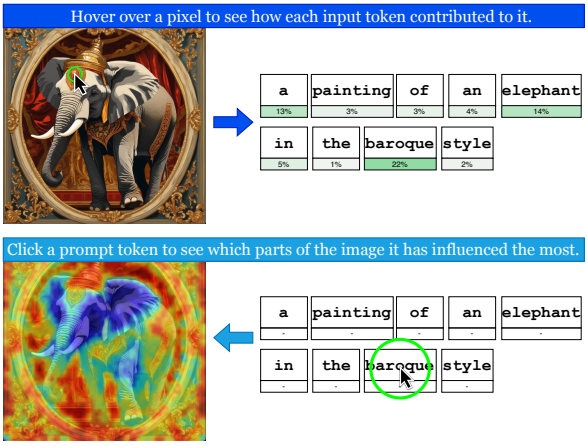

Figure 1: Bidirectional exploration of text-to-image attribution in LumiXAI. *Top* (image→token): hovering a pixel reads that point from every prompt token's attention matrix and recolors the prompt cards. *Bottom* (token→image): clicking a prompt token overlays its attribution heatmap on the generated image.

This paper presents **LumiXAI**[1], a modular, full-stack framework for feature-attribution analysis across text and image classification, autoregressive text generation, and text-to-image diffusion. LumiXAI is designed to make attribution analyses inspectable, reproducible, and extensible within a single workflow: model families and attribution methods are exposed as registered plug-ins; each layer runs as an independent Docker service; anal-

[1]Code and documentation are available at https://github.com/TechLory/LumiXAI; an online demo is available at http://lumixai.islab.di.unimi.it/.

ysis metadata and attribution outputs are persisted for later inspection and sharing; and the same backend supports a web GUI, a Python SDK, and a smart-batch mode for larger experiments. In this way, LumiXAI brings together interaction styles, user groups, and model families that existing tools typically address only in isolation. Figure 1 illustrates an example of the interaction with LumiXAI for text-to-image attribution. In addition to the conventional token-to-image view, where a selected prompt token highlights the regions it contributes to, LumiXAI supports the reverse direction: users can hover over an image location and inspect which prompt tokens are most responsible for that point.

The remainder of the paper is organised as follows. Section 2 situates LumiXAI with respect to existing attribution methods and interpretability toolkits. Section 3 describes the framework architecture, including its plug-in interface, service-based deployment, storage layer, and user-facing access modes. Section 4 presents case studies illustrating LumiXAI across supported model families and interaction scenarios. Section 5 reports a user evaluation of the system's usability and interpretability workflow. Section 6 concludes with current limitations and directions for future work.

Tool	Scope	Gen. Attr.	GUI	Ext.	Audience
Captum	TC, TG, IC	●	○	●	🧩 </>
SHAP / LIME	TC	○	○	○	</>
OmniXAI	TC, IC	○	●	○	</> 👤
AIX360	TC, IC	○	○	○	</>
AllenNLP Interpret	TC, TG	◐	●	●	🧩 </>
LIT	TC, TG	◐	●	●	🧩 👤
ecco	TG	●	●	○	</>
Thermostat	TC	○	○	○	</>
Inseq	TG	●	○	●	🧩 </>
ferret	TC	○	○	●	🧩 </>
CafGa	TG	●	●	○	👤
ICX360	TG	●	○	○	</>
LumiXAI	**TC, TG, IC, T2I**	●	●	●	🧩 </> 👤

Table 1: Comparison of attribution tools. *Scope*: TC = text classification, TG = text generation, IC = image classification, T2I = text-to-image generation. *Generative attribution*: ● first-class attribution over the generative decoding process (autoregressive or diffusion); ◐ limited to masked-token or single-step prediction, or salience confined to particular seq2seq settings; ○ classification or regression outputs only. *GUI*: whether it ships an interactive visual interface, e.g. standalone application or interactive notebook widget. *Ext.*: if it is designed for adding new attribution methods or model types. *Audience*: 🧩 extender (adds methods or models), </> developer (applies methods through a code API), 👤 non-programmer (works through the GUI alone). Cells reflect released versions as of July 2026 (Captum 0.9.0, SHAP 0.52.0, Inseq 0.7.1, LIT 1.3.1, CafGa 0.0.6, ICX360 0.1.0).

2 Related Work

Attribution tooling has proliferated alongside interpretability research, but existing tools tend to specialise along a small number of axes and rarely combine their strengths. Table 1 places a representative selection along the five dimensions that structure the discussion below: the input modalities with first-class attribution support (text classification, generative text, image classification, and text-to-image generation); whether attribution is defined over the generative decoding process rather than a single classification output; whether the tool ships an interactive GUI; whether its architecture is designed to accommodate new methods or model types; and the audiences it serves. We distinguish three such audiences, since few tools address more than one: *extenders* (🧩), who implement a new attribution method or adapt an unsupported model type and need an interface in which to exercise it; *developers* (</>), who apply existing methods through a code API; and *non-programmers* (👤), who work entirely through a GUI.

General-purpose libraries provide broad attribution backends but limited interactive support. LIME (Ribeiro et al., 2016b) and SHAP (Lundberg and Lee, 2017) offer model-agnostic explanations and plotting utilities, but are mainly used through code and are classification-oriented. Captum (Kokhlikyan et al., 2020) provides an extensible PyTorch library of gradient- and perturbation-based methods, now including support for image masks and language-model attribution (Miglani et al., 2023), but its interactive companion, Captum Insights, was retired after v0.8.0. OmniXAI (Yang et al., 2022) and AIX360 (Arya et al., 2019) broaden coverage across data types, yet remain centered on classification rather than generative decoding. NLP-specific tools offer richer model inspection but cover only parts of our target setting. AllenNLP Interpret (Wallace et al., 2019) exposes saliency and gradient primitives with reusable front-end components, but is tied to the deprecated AllenNLP framework. LIT (Tenney et al., 2020) provides an extensible browser-based workbench for classification and sequence-to-sequence models, although it is broader than attribution alone. ecco (Alammar, 2021) visualizes language-model behavior interactively from code, while Thermostat (Feldhus et al., 2021) distributes precomputed

classifier explanations. ferret (Attanasio et al., 2023) complements these systems by benchmarking attribution methods for transformer classifiers. The closest tools to LumiXAI target generative text. Inseq (Sarti et al., 2023) supports feature attribution for decoder-only and encoder-decoder generation, but exposes results through a Python API and does not address image models. ICX360 (Wei et al., 2025) attributes LLM generations to their input context, but is likewise text-only and API-driven. CafGa (Boyle et al., 2025) adds interactive controls for attribution granularity, including Jupyter widgets, but remains confined to text.

LumiXAI differs in providing the widest scope support, combining generative attribution across text and text-to-image diffusion with an interactive GUI, documented extension interfaces, and multiple access tiers backed by the same services. This design lets non-programmers inspect analyses visually, developers access them programmatically, and extenders add new model families or attribution methods through the same registry.

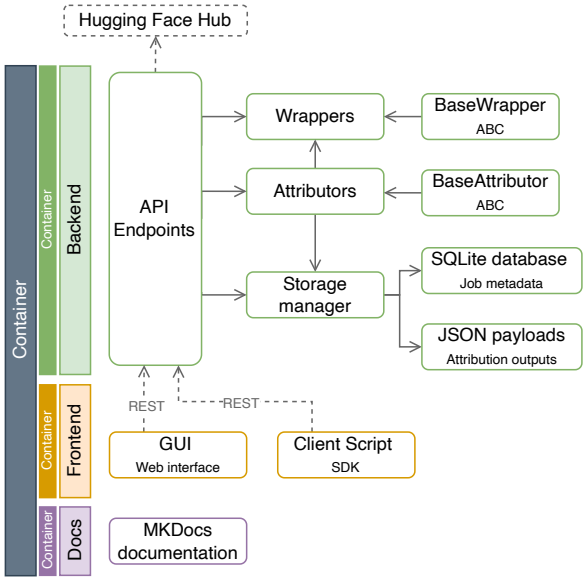

Figure 2: LumiXAI architecture. The backend exposes a REST API, orchestrates wrapper and attributor plug-ins, retrieves models from the Hugging Face Hub, and stores metadata in SQLite with attribution payloads as JSON files. Backend, frontend, and documentation are deployed as independent containers, and both the web GUI and client scripts consume the same API.

3 The LumiXAI Framework

LumiXAI is a client–server system that isolates model-specific and compute-intensive functionality behind a REST API. Clients interact with this boundary rather than with model classes directly, allowing the same backend to support the web interface, Python SDK, and batch workflows. Internally, the attribution engine is organised around two abstract plug-in interfaces, wrappers and attributors, which decouple model access from explanation computation and make it possible to add new tasks, models, or attribution methods without changing the core system.

3.1 Architecture Overview

Figure 2 summarizes the system architecture. LumiXAI is deployed as three independent services: a FastAPI backend, a Next.js frontend, and MkDocs documentation. The backend exposes the REST API, hosts the attribution engine, and is the only component that loads model weights. Wrappers retrieve and adapt models, currently from the Hugging Face Hub, while attributors compute explanations through a shared interface.

The frontend and SDK are both ordinary API clients and contain no attribution logic. Attribution requests may be long-running, so the backend handles them as jobs: each request receives an identifier, while computation proceeds asynchronously. Access to the accelerator is serialised to avoid memory conflicts, and device placement is resolved at load time across CUDA, Apple MPS, or CPU.

Completed analyses are persisted through a hybrid storage layer. Structured job metadata, such as model, method, prompt, status, and elapsed time, are stored in SQLite. Larger attribution payloads, including token scores, per-step generation traces, attention matrices, and generated images, are serialised as JSON files referenced from the corresponding database row. This design keeps analyses available for later inspection, export, or transfer.

3.2 Extensibility: Wrappers and Attributors

Extensibility is enforced by keeping models and attribution algorithms behind abstract interfaces. A `BaseWrapper` provides a uniform view of a model family: it loads weights, places the model on the selected device, and exposes the forward operation. When available, it may also expose embeddings for gradient-based methods, but this is not required for attribution methods that rely on other signals, such as attention.

A `BaseAttributor` defines an attribution method as a mapping from a wrapped model and an input to a standardised output object. The wrapper

is injected into the attributor at construction time, and each attributor declares the wrapper families it supports. The backend uses this declaration to reject incompatible model–method pairings before loading weights, while the frontend uses the same information to disable invalid choices.

Wrappers and attributors are registered as plug-ins. Adding a new model family or attribution method therefore requires one subclass, an applicability declaration, and one registry entry. The client remains independent of these details: it populates its selectors from a backend manifest, and the backend resolves the appropriate wrapper from the task metadata declared by the selected model.

The data contracts crossing the plug-in boundary are modality-agnostic. The atomic unit of input is an *input feature*, i.e. a token with its position for text, a patch or a superpixel for an image, and every attributor returns the same output object irrespective of what was explained. The coverage reported in Section 3.3 is thus what is currently implemented.

3.3 Supported Models and Methods

The current release ships four wrappers. `HFTextClassificationWrapper` adapts encoder classifiers and returns class logits with token-level scores. `HFTextGenerationWrapper` adapts autoregressive models through a step-wise decoding loop, producing an *attribution trace* that links each generated token and probability to attribution scores over the context available at that step. `HFImageClassificationWrapper` supports vision transformers and CNN classifiers, rendering explanations over the model's preprocessed, de-normalised input. `HFImageWrapper` adapts diffusion-based text-to-image models and exposes the components needed for cross-attention attribution.

LumiXAI provides ten established attribution methods. Seven are available through Captum for text and image classification: Integrated Gradients (Sundararajan et al., 2017), DeepLift (Shrikumar et al., 2017), Saliency (Simonyan et al., 2014), Input×Gradient, GradientSHAP (Lundberg and Lee, 2017), Occlusion (Zeiler and Fergus, 2013), and LIME (Ribeiro et al., 2016a). For perturbation methods, the perturbed unit is the token for text, a regular patch grid for image Occlusion, and SLIC superpixels (Achanta et al., 2012) for image LIME. Two further methods, SmoothGrad (Smilkov et al., 2017) and Grad-CAM (Selvaraju et al., 2017), target vision models. Finally, DAAM (Tang et al., 2023) attributes text-to-image outputs by recovering per-token spatial maps from diffusion cross-attention.

3.4 Multi-Tier Access

A single backend is exposed through three access modes, corresponding to the "Audience" of Table 1.

Web GUI (👤). The Next.js interface lets users configure, launch, and inspect attribution jobs without writing code. Classification views display token or image heatmaps alongside model predictions. Generative views expose the attribution trace bidirectionally: in text generation, users can move from input tokens to generated-token influence or from a generated token back to its supporting context; in text-to-image generation, they can inspect token heatmaps on the image or query a pixel to see which prompt tokens contributed most. The GUI also includes tutorials and a history view for revisiting completed jobs.

Python SDK (</>). A single self-contained module drives the REST API programmatically, for headless use in any environment able to reach the backend. Documented notebooks accompany it and reproduce the case studies of Section 4. For larger sweeps, `run_smart_batch()` reduces repeated model weight loading by grouping jobs by model and method, so that each set of weights is loaded once per group, and the groups are then *ordered* by a configurable strategy (`fastest_first`, `slowest_first`, or `none`) suited to shared and dedicated accelerators respectively. Results are collected by polling and returned in the caller's original job order irrespective of completion order, so that the optimization remains transparent to the calling code.

Framework extension (🧩). The third mode addresses researchers whose model or attribution method the framework does not yet support, and who therefore act on the codebase rather than through it. Such a user subclasses `BaseWrapper` or `BaseAttributor`, declares the new component in the registry, and obtains a plug-in that the backend orchestrates on equal terms with those shipped by default. Because the manifest is derived from the registry, the new method appears in the GUI selectors and is reachable through the REST API and the SDK immediately.

3.5 Deployment

LumiXAI is deployed with Docker Compose. A base `docker-compose.yml` defines the backend, frontend, and documentation services, while an optional `docker-compose.gpu.yml` enables GPU execution through the NVIDIA Container Toolkit. Services can be run together or independently: the backend is sufficient for SDK use, the frontend enables interactive analysis, and the documentation can be served separately.

The backend image is built with Poetry from a lockfile, and configuration is parameterised through environment variables with in-file defaults. The Hugging Face cache is mounted as a volume so that model weights persist across restarts. Together with the SQLite metadata store and JSON attribution payloads described above, this makes both the execution environment and completed analyses portable across machines.

Runtime is mainly determined by the selected model and attribution method, including model inference, repeated forward or backward passes, and, where applicable, diffusion or generation steps. LumiXAI adds scheduling, serialization, and retrieval around these computations, but these components are not expected to be the main source of latency; we leave a dedicated measurement of framework overhead to future work.

4 Case Studies

We illustrate LumiXAI usage through two examples covering text classification and text-to-image diffusion. Additional examples on image classification and open-ended text generation are provided in the Appendix A.

Text Classification. We first examine which input features drive a toxicity classifier. A curated subset of Civil Comments (Borkan et al., 2019) is passed to `unitary/toxic-bert`, and each prediction is attributed toward the most probable class using Integrated Gradients. In the frontend, tokens are colored by signed attribution score, making it possible to inspect whether the model relies on identity terms, offensive language, or quoted stereotypes when assigning toxicity labels (Figure 3).

Text-to-Image Diffusion. The second case study explores whether text-to-image diffusion models spatially separate artistic *content* and *style*. Following Ferrara et al. (2025), we generate images from prompts pairing a subject with an artistic style

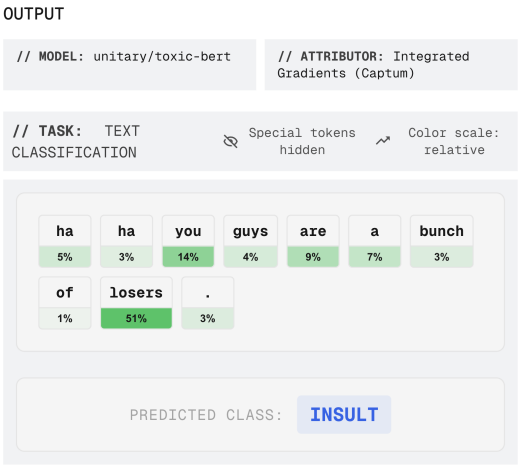

Figure 3: Text classification in LumiXAI. For the comment *"ha ha you guys are a bunch of losers."*, classified as INSULT by `unitary/toxic-bert`, tokens are colored by Integrated-Gradients attribution toward the predicted class; *losers* dominates the attribution (51%).

and attribute pixels to prompt tokens using DAAM. Since diffusion cross-attention links prompt tokens to image regions during denoising, LumiXAI makes it possible to compare the spatial influence of content and style terms directly. Selecting a prompt token overlays its attribution heatmap on the image, while selecting an image region reveals the responsible prompt tokens. This bidirectional view supports inspection of whether content tokens localize to depicted objects, whether style tokens spread across texture and background, and where the model entangles the two (Figure 4).

5 User Evaluation

We conducted a user evaluation to assess whether LumiXAI is usable by different target audiences and whether its interface supports the inspection of attribution results across tasks. The study focuses on usability and workflow effectiveness rather than on the intrinsic quality of the explanations, which depends on the external attribution libraries we employ.

Setup. We recruited 12 participants, including 3 participants with experience in machine learning or NLP and 9 non-expert users. Expertise was determined from two background questions on prior experience training or evaluating ML models and using feature-attribution or explainability methods.

Each participant followed the same protocol. After giving consent and answering the background questions, participants completed the LumiXAI tu-

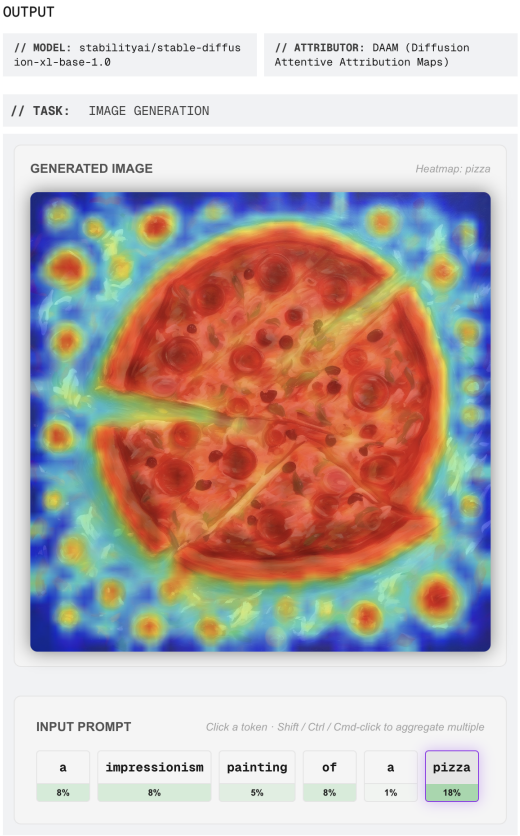

Figure 4: Text-to-image attribution in LumiXAI. For the prompt *"a impressionism painting of a pizza"*, generated by `Stable Diffusion XL`, selecting *pizza* overlays its DAAM heatmap (token→image), while token percentages report attribution at the selected pixel (image→token).

torials, introducing the supported tasks, attribution views, and interaction patterns, followed by a short comprehension check on the meaning and sign of attribution scores. Participants then completed 8 tasks across four task types: text classification, image classification, autoregressive text generation, and text-to-image diffusion. For each task type we fixed the model and attribution method and provided the required input (e.g., a text prompt or image); participants launched the attribution jobs and inspected the results, exploring token-level attributions in both directions for text tasks and attribution regions for image tasks. This let us evaluate interface usability while keeping model and method choices controlled across users. Finally, participants rated usability statements about LumiXAI on a 7-point Likert scale and gave free-text feedback on the most useful feature, on confusing or frustrating aspects, and on anything they wanted to inspect that the system did not allow.

Results. Table 2 summarizes the questionnaire results. Overall, participants judged the system positively, appreciating in particular the ease of switching between models and attribution methods and its helpfulness in understanding model behaviour, although ratings were consistently higher among expert users. The qualitative feedback confirms this picture, indicating that clearer onboarding, more transparent heatmap colour scaling, and explicit indication of method–model compatibility would be the main avenues for improving accessibility to non-experts.

Question	Overall	Experts	Non-experts
Ease of use	4.92	5.67	4.67
Learnability	5.17	6.33	4.78
Enjoyability	4.75	5.67	4.44
Integration	5.33	5.67	5.22
Helpfulness	5.25	6.00	5.00
Reuse intention	4.25	4.33	4.22
Bidirectionality	4.67	5.00	4.56
Switching	5.58	5.67	5.56
Compatibility	4.58	5.33	4.33
Consistency	5.00	5.67	4.78
Mean	4.95	5.53	4.76

Table 2: Mean user-study ratings (7-point Likert scale) per questionnaire item, overall and by expertise ($n = 12$; 3 experts, 9 non-experts).

6 Conclusion

We presented **LumiXAI**, a modular full-stack framework for reproducible and interactive feature-attribution analysis. LumiXAI operationalizes established attribution techniques within a unified system that spans multiple model families, access modes, and user profiles. Through its plug-in architecture, service-based deployment, persistent storage layer, and complementary interfaces, the framework turns attribution workflows that often require ad hoc scripting into analyses that can be inspected, repeated, and extended. The case studies illustrate how LumiXAI supports both targeted debugging and exploratory interpretation across classification, generation, and text-to-image diffusion scenarios. The framework current coverage is necessarily limited to the model families and attribution methods implemented so far, and future work will extend the plug-in library with additional techniques and architectures. In particular, emerging multimodal models, such as image-text-to-text systems, raise new attribution questions that fit naturally within LumiXAI's modality-agnostic design.

Limitations

LumiXAI is extensible, but its current implementation covers only a subset of possible tasks and models. Other architectures, especially image-text-to-text vision-language models, require additional wrappers and visualization components. Likewise, the supported attribution methods are representative rather than exhaustive.

The tool also inherits the limitations of attribution methods themselves. Scores can vary with baselines, tokenization, perturbation units, aggregation choices, and preprocessing, and should therefore be treated as diagnostic signals rather than causal explanations. This is especially relevant for generative models, where decoding traces and diffusion attention maps expose useful patterns without fully explaining the underlying computation.

The system was primarily developed and tested under single-user conditions, with concurrency handled only across multiple processes issued by that user. During the user evaluation, some participants encountered bugs caused by multiple users concurrently loading and unloading different configurations on the same backend, a scenario we had not accounted for; robustness under concurrent multi-user load remains future work.

Finally, the user evaluation is limited in scale and scope. It tests whether participants can complete guided attribution tasks and interpret the visualizations, but does not measure long-term use in open-ended research workflows. Moreover, we do not isolate framework overhead from model inference and attribution time, so a dedicated systems benchmark is left to future work.

Ethics Statement

Participation in the user evaluation was voluntary, and participants were informed of the study's purpose, procedure, and expected duration before starting. Written informed consent was obtained from all participants prior to the tutorial phase. No personally identifiable information was collected: questionnaire responses and free-text feedback were anonymised at collection time and are reported only in aggregate. Participants could withdraw from the study at any point without providing a reason. The study involved no more than minimal risk, as it consisted of guided interaction with a software interface and did not involve sensitive personal data, deception, or vulnerable populations.

References

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282.

J Alammar. 2021. Ecco: An Open Source Library for the Explainability of Transformer Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 249–257, Online. Association for Computational Linguistics.

Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. 2019. One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques. *Preprint*, arXiv:1909.03012.

Giuseppe Attanasio, Eliana Pastor, Chiara Di Bonaventura, and Debora Nozza. 2023. Ferret: A Framework for Benchmarking Explainers on Transformers. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 256–266, Dubrovnik, Croatia. Association for Computational Linguistics.

Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. 2019. Nuanced metrics for measuring unintended bias with real data for text classification. In *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, page 491–500, New York, NY, USA. Association for Computing Machinery.

Alan David Boyle, Furui Cheng, Vilém Zouhar, and Mennatallah El-Assady. 2025. CafGa: Customizing Feature Attributions to Explain Language Models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 461–470, Suzhou, China. Association for Computational Linguistics.

Jwala Dhamala, Tony Sun, Varun Kumar, Satyapriya Krishna, Yada Pruksachatkun, Kai-Wei Chang, and Rahul Gupta. 2021. Bold: Dataset and metrics for measuring biases in open-ended language generation. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 862–872, New York, NY, USA. Association for Computing Machinery.

Nils Feldhus, Robert Schwarzenberg, and Sebastian Möller. 2021. Thermostat: A Large Collection of

NLP Model Explanations and Analysis Tools. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 87–95, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Alfio Ferrara, Sergio Picascia, and Elisabetta Rocchetti. 2025. The cow of rembrandt analyzing artistic prompt interpretation in text-to-image models. In *2025 IEEE 35th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for PyTorch. *Preprint*, arXiv:2009.07896.

Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Vivek Miglani, Aobo Yang, Aram Markosyan, Diego Garcia-Olano, and Narine Kokhlikyan. 2023. Using Captum to Explain Generative Language Models. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 165–173, Singapore. Association for Computational Linguistics.

Marco Ribeiro, Sameer Singh, and Carlos Guestrin. 2016a. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California. Association for Computational Linguistics.

Marco Tulio Ribeiro, View Profile, Sameer Singh, View Profile, Carlos Guestrin, and View Profile. 2016b. "Why Should I Trust You?". In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Conferences, pages 1135–1144.

Gabriele Sarti, Nils Feldhus, Ludwig Sickert, and Oskar van der Wal. 2023. Inseq: An Interpretability Toolkit for Sequence Generation Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 421–435, Toronto, Canada. Association for Computational Linguistics.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3145–3153. JMLR.org.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. *Preprint*, arXiv:1312.6034.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. 2017. Smoothgrad: removing noise by adding noise. *Preprint*, arXiv:1706.03825.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3319–3328. PMLR.

Raphael Tang, Linqing Liu, Akshat Pandey, Zhiying Jiang, Gefei Yang, Karun Kumar, Pontus Stenetorp, Jimmy Lin, and Ferhan Ture. 2023. What the DAAM: Interpreting Stable Diffusion Using Cross Attention. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5644–5659, Toronto, Canada. Association for Computational Linguistics.

Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, and Ann Yuan. 2020. The Language Interpretability Tool: Extensible, Interactive Visualizations and Analysis for NLP Models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 107–118, Online. Association for Computational Linguistics.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019. AllenNLP Interpret: A Framework for Explaining Predictions of NLP Models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 7–12, Hong Kong, China. Association for Computational Linguistics.

Dennis Wei, Ronny Luss, Xiaomeng Hu, Lucas Monteiro Paes, Pin-Yu Chen, Karthikeyan Natesan Ramamurthy, Erik Miehling, Inge Vejsbjerg, and Hendrik Strobelt. 2025. ICX360: In-Context eXplainability 360 Toolkit. *Preprint*, arXiv:2511.10879.

Wenzhuo Yang, Hung Le, Tanmay Laud, Silvio Savarese, and Steven C. H. Hoi. 2022. OmniXAI: A Library for Explainable AI. *Preprint*, arXiv:2206.01612.

Matthew D Zeiler and Rob Fergus. 2013. Visualizing and understanding convolutional networks. *Preprint*, arXiv:1311.2901.

A Additional Case Studies

We further illustrate LumiXAI features through two additional examples covering image classification and open-ended text generation.

Image Classification. This classification case study illustrates attribution over a visual input. We apply a Vision Transformer trained on MNIST digits (`farleyknight-org-username/vit-base-mnist`) and explain its predicted class using the Captum implementation of Grad-CAM. In the image view, LumiXAI overlays the attribution heatmap directly on the model input, allowing users to inspect whether the classifier grounds its decision in the digit strokes rather than in background artifacts. In Figure 5, the model predicts the class 3, and the strongest activations align with the upper, middle, and lower curves that define the handwritten digit.

Figure 5: Image classification in LumiXAI. A ViT model trained on MNIST predicts the digit 3; the Grad-CAM overlay highlights the image regions that most support the predicted class, with attribution concentrated on the main strokes of the digit.

Open-Ended Text Generation. This case study uses the per-step attribution trace for autoregressive generation. We sample Wikipedia-derived prompt prefixes from BOLD (Dhamala et al., 2021) and let `openai-community/gpt2` continue them deterministically with a short decoding horizon. For each generated token, Integrated Gradients records its probability together with attribution scores over the preceding prompt and generated context. The bidirectional generation view lets users select any generated token and inspect which earlier tokens contributed most to it. This supports visual analysis of whether continuations depend on demographic or identity-bearing terms, whether such tokens become attribution hotspots for evaluative words, and how attribution patterns differ between neutral and stereotype-adjacent completions (Figure 6).

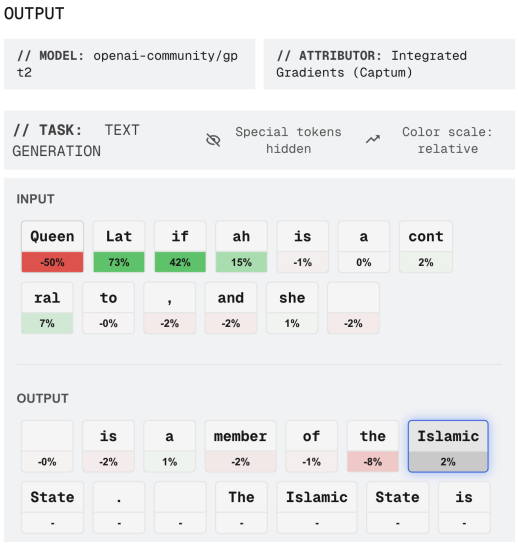

Figure 6: Open-ended generation in LumiXAI. Continuing the BOLD prefix *"Queen Latifah is a contralto, and she"* with `gpt2`, the selected token *Islamic* is explained by signed Integrated-Gradients scores over the prior context; *Queen* contributes negatively (-50%), while *Lat* contributes positively ($+73\%$).