Multisensor-Pipeline: A Lightweight, Flexible, and Extensible Framework for Building Multimodal-Multisensor Interfaces

MICHAEL BARZ, German Research Center for Artificial Intelligence (DFKI), Germany and Applied Artificial Intelligence, Oldenburg University, Germany

OMAIR SHAHZAD BHATTI, German Research Center for Artificial Intelligence (DFKI), Germany

BENGT LÜERS, Applied Artificial Intelligence, Oldenburg University, Germany

ALEXANDER PRANGE, German Research Center for Artificial Intelligence (DFKI), Germany

DANIEL SONNTAG, German Research Center for Artificial Intelligence (DFKI), Germany and Applied Artificial Intelligence, Oldenburg University, Germany

We present the multisensor-pipeline (MSP), a lightweight, flexible, and extensible framework for prototyping multimodal-multisensor interfaces based on real-time sensor input. Our open-source framework (available on GitHub) enables researchers and developers to easily integrate multiple sensors or other data streams via source modules, to add stream and event processing capabilities via processor modules, and to connect user interfaces or databases via sink modules in a graph-based processing pipeline. Our framework is implemented in Python with a low number of dependencies, which enables a quick setup process, execution across multiple operating systems, and direct access to cutting-edge machine learning libraries and models. We showcase the functionality and capabilities of MSP through a sample application that connects a mobile eye tracker to classify image patches surrounding the user's fixation points and visualizes the classification results in real-time.

CCS Concepts: • **Software and its engineering** → *Software libraries and repositories*; • **Human-centered computing** → *Interactive systems and tools*; • **Computing methodologies** → Computer vision.

Additional Key Words and Phrases: multimodal-multisensor interfaces, open source framework, prototyping, stream processing, eye tracking, computer vision

ACM Reference Format:

Michael Barz, Omair Shahzad Bhatti, Bengt Lüers, Alexander Prange, and Daniel Sonntag. 2021. Multisensor-Pipeline: A Lightweight, Flexible, and Extensible Framework for Building Multimodal-Multisensor Interfaces. In *Companion Publication of the 2021 International Conference on Multimodal Interaction (ICMI '21 Companion), October 18–22, 2021, Montréal, QC, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3461615.3485432

1 INTRODUCTION

For many applications, multimodal interfaces have long been recognized to be more robust, accurate, and preferred by users than unimodal ones [17]. This is the case because users can choose the individually most suitable modality or modality combination for solving the problem at hand. Major challenges in research on multimodal interfaces include the integration of new sensors and the development of effective algorithms for multimodal signal analysis [19, 24]. This requires a suitable software infrastructure that allows researchers to prototype and adapt or extend novel interaction systems in an effective and efficient way [18, 25].

In this work, we present the multisensor-pipeline (MSP): a lightweight, flexible, and extensible framework for building multimodal-multisensor interfaces based on real-time sensor input. It enables researchers and developers to easily build and adapt stream processing pipelines by connecting existing or custom modules. Our framework introduces a minimal set of concepts and provides convenience functions for building and running processing pipelines. This leaves the implementation of modules to the researchers and developers and allows them to get started quickly. We showcase the functionality and effectiveness of our framework by implementing a sample application that classifies ambient objects fixated by a user; here we employ a pre-trained deep learning model to visualize the top-5 results in real-time.

2 RELATED WORK

MSP is related to other real-time stream and event processing frameworks. This includes modern streaming systems for distributed event processing which are used to implement for example mobile and web-based applications at scale. One example is Apache Flink[1], a framework for batch and stream processing based on dataflow graphs [14]. In Apache Flink, a dataflow graph is represented as a directed acyclic graph which defines operators (nodes) and data streams (edges). However, these systems are designed for scalable, high-throughput data analysis with fault tolerance which introduces overheads for, e.g., the installation and development process. Other frameworks support the development of multimodal dialogue systems. For instance, Advisor is a modular and extensible framework for developing socially-engaged dialogue systems [15]. It can be used to create conversational agents with available modules for, e.g., social signal processing and speech processing. Likewise, the EEVA framework enables the development of interactive social agents for the web [20]. SiAM-dp is a platform for developing multimodal dialogue systems which has a focus on integrating distributed input and output devices in cyber-physical environments [16], based on SmartWeb's iHub platform [21, 22, 26]. With MSP, we introduce a middleware [7] that eases the development of such dialogue systems or similar applications that require real-time multimodal signal processing.

More closely related frameworks are aiming for synchronized processing of real-time sensor inputs in the domain of multimodal user interaction. The Social Signal Interpretation (SSI) framework[2] enables real-time recognition of social signals [27]. It supports a range of sensors and provides a set of ready-to-use modules for, e.g., signal filtering, feature extraction, and machine learning. Developers can add new components written in C++ or Python using their application programming interface. The framework is actively maintained and licensed as open source. NOVA is built on top of SSI and enables interactive data annotation using semi-supervised active learning techniques [4, 10]. SSJ is an android port of the SSI framework that enables real-time signal processing on mobile phones in the wild [6]. The open source Platform for Situated Intelligence (\psi) aims to support the rapid development and study of multimodal, integrative AI systems [5]. Similar to SSI, it provides a set of components for capturing and analyzing data streams from multiple sensors that can be connected to versatile processing pipelines. Also, they offer a set of tools for debugging, data visualization, annotation, and analysis. \psi is implemented in C# and licensed as open source[3]. Unlike SSI and \psi, MSP comes with a concise set of concepts and functionalities only, that ease the creation and execution of complex processing pipelines. We leave the integration of sensors and the implementation of algorithms to the researchers and developers of multimodal interfaces.

[1]https://flink.apache.org/
[2]https://hcai.eu/projects/ssi/
[3]https://github.com/microsoft/psi

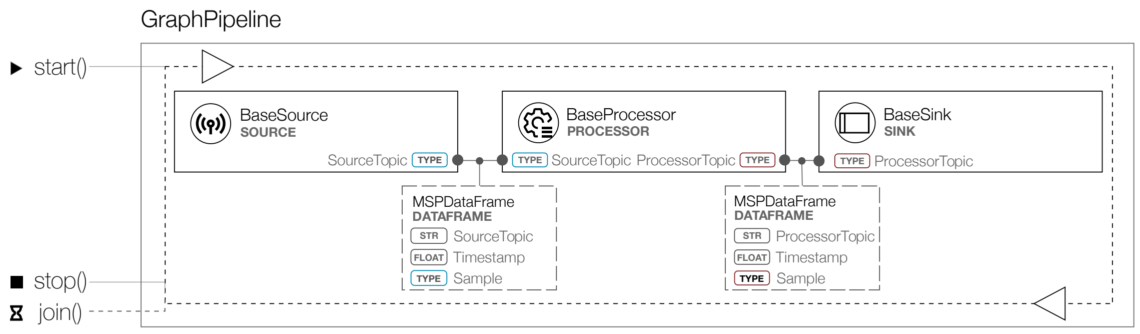


Fig. 1. Architectural overview and programming interface of a general pipeline.

3 MULTISENSOR-PIPELINE (MSP)

Multisensor-pipeline (MSP) is a framework for prototyping sensor-based user interfaces based on real-time sensor input. It enables researchers and developers to build complex processing pipelines from existing or custom modules with a low overhead. A pipeline consists of at least one source module, one sink module, and an arbitrary number of processor modules which form a weakly-connected, directed graph (see Figure 1). Data frames originating from sources or processors flow along the directed edges to connected processors or sinks. *We release MSP as open-source software on GitHub*[4] *which can be installed from source or using the pip package manager from the official Python package index (PyPI).* Next, we describe the two main concepts of our framework, namely the module and the pipeline, and a sample application based on MSP.

3.1 Modules

The module is a core concept of MSP which is represented by the `BaseModule` class. A module can be a source, processor, or sink by inheriting from the `BaseSource`, `BaseProcessor`, or `BaseSink` class, respectively. The communication between modules is realized using the observer pattern. As an example, Figure 2 shows the general architecture of a processor module which combines the functionality of a sink and a source. Source modules integrate sensors or other data sources, such as a camera or an eye tracking device, and notify registered processors or sinks of new data samples. Processor modules can subscribe to source modules and other processors. Processors consume and process incoming real-time signals or events and notify their observers of the result. Sink modules can subscribe to source modules and processors. Sinks only consume incoming data samples and, for instance, visualize or store the received information. Each module offers a `start()` method which calls the `on_start()` method and starts the processing loop in a separate thread. The loop frequently calls the `on_update()` method which implements the event handling logic of the module. It may return `None`, if nothing shall happen, or an instance of `MSPDataFrame` to transmit a data sample or an event. `MSPDataFrame` adds metadata to each data sample which can be used by connected processors and sinks down the line: an obligatory timestamp, a unique name describing the sample, and a data type. Received instances are buffered in a queue. If a sample becomes obsolete, because too many samples arrive or processing takes too long, it is removed from the queue. Stopping a module can be initiated using its `stop()` method. It interrupts the processing loop and triggers a call to `on_stop()`. The methods `on_start()` and `on_stop()` can be used to manage the resources required at runtime.

[4] https://github.com/DFKI-Interactive-Machine-Learning/multisensor-pipeline

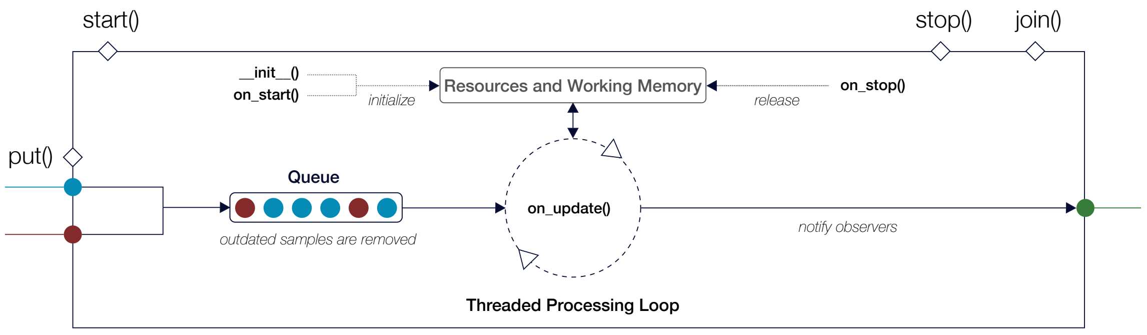


Fig. 2. Architectural overview and programming interface of a processor module.

To avoid unresponsive behavior, large resources should be loaded in the `__init__()` method of a module. A call to `join()` allows the calling thread to wait until the processing loop was stopped and all resources were released. To implement a custom source, processor, or sink module, developers must override the abstract methods of the respective base class.

3.2 Pipeline

Another core concept of MSP is the pipeline implemented by the `GraphPipeline` class. A `GraphPipeline` instance is a handle to manage a processing pipeline including its modules. The modules and connections between those are represented as a weakly-connected, directed graph. The pipeline class offers functions to add modules in this graph as nodes and connections as edges. This makes it easy to replace, reorder, or add modules to update the functionality of a pipeline. For instance, a processor module that classifies images using a machine learning model can be replaced by a processor that uses a more recent model when creating a pipeline. Likewise, any processor module could be replaced during experiments to investigate the corresponding effect on the utility and usability of a multimodal interface. Also, the pipeline class offers convenience functions to start, stop, and join a processing pipeline. A call to the `start()` method triggers the `on_start()` methods of all modules and starts their threaded processing loops. The pipeline runs until its `stop()` method is called or until all subscribed sources sent an *end of stream* control message. In both cases, the main thread can wait until the pipeline stopped using its `join()` method.

3.3 Sample Application

We illustrate the concepts and showcase the functionality of the MSP framework by implementing a sample application. We prototype a sample use-case from the domain of situation-aware, gaze-based user interaction using a mobile eye tracker. The prototype classifies ambient objects fixated by a user and visualizes the results as a video overlay in real-time, similar to [2]. This context information can be used to build, for instance, cognition-aware, multimodal interfaces. We implement four modules for this application: an eye tracking source, an image cropping processor, an image classification processor, and a video visualization sink (see Figure 3). The source code is provided with this paper as supplementary material.

The eye tracking source `PupilLabsCore` inherits from our `BaseSource` class to connect a Pupil Labs Core mobile eye tracker. We use the networking API[5] of their recording tool Pupil Capture [13] which provides real-time access to the eye

[5]https://docs.pupil-labs.com/core/software/pupil-capture/#network-plugins

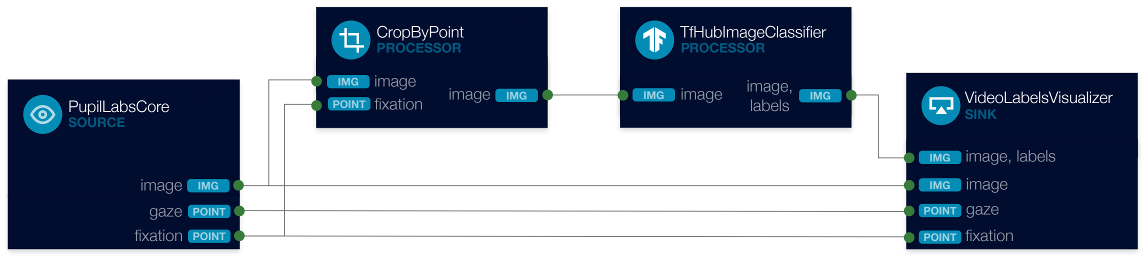


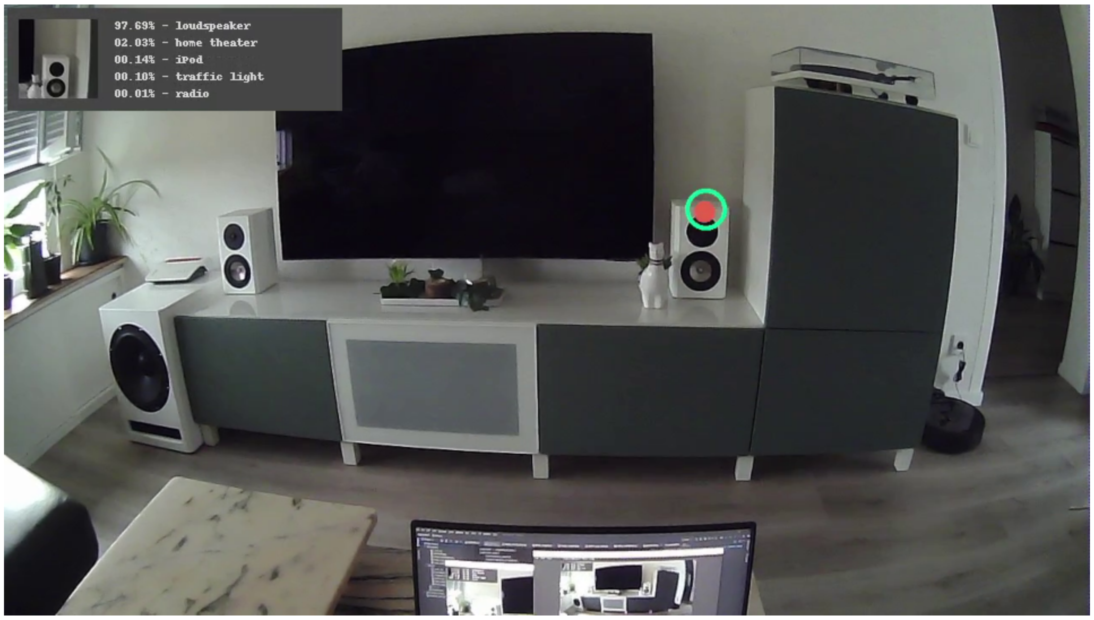
Fig. 3. Pipeline of our sample application with a source, two processors, and one sink module.

tracking data. We connect the video feed of the front-facing camera (30 Hz), the raw gaze signal (200 Hz), and recognized fixation events. Typically, fixations last around 200 to 400 ms [11]. The image cropping processor `CropByPointer` inherits from the `BaseProcessor` class. It fuses two input modalities, a video stream and a 2-dimensional signal that lies within the coordinate system of the video frames. The processor crops an image patch of a configurable size around each new point using the latest available image frame. In our example, both signals originate from the eye tracking source. We crop quadratic image patches with an edge length of 200 pixels which turned out to work well in a similar scenario [1]. The image classification processor `TfHubImageClassifier` is a wrapper around TensorFlow Hub[6] which provides access to a range of pre-trained machine learning models. It connects image classification models that are pre-trained on the ImageNet dataset [23]. Inference runs on the local CPU or GPU using a downloaded structure and weights of a model. The model can be set using the URLs from TensorFlow Hub. We use a pre-trained[7] ResNet-50 [9] model in our example. Eventually, the module annotates image patches with the top-k class labels and their probabilities (k is configurable and set to 5 in our example). The images originate from the image cropping processor. The video visualization sink `VideoLabelsVisualizer` consumes four input streams: a video stream, a 2-dimensional signal for gaze samples and fixation events each, and the image classification output which includes an image patch and corresponding class labels with probabilities. For each incoming video frame, we visualize the most recent eye movement data and classification results as an overlay and render the new image to a window on the screen (see Figure 4). The input comes from the eye tracking source and the image classification processor.

4 DISCUSSION

Our sample application shows that MSP can be used to effectively prototype multimodal-multisensor interfaces. Our framework is flexible and extensible due to its module-based architecture and because it is implemented in Python, which provides cross-platform support and access to a broad range of data science and machine learning tools maintained by an active community. A processing pipeline can easily be changed by rearranging existing modules and extended by adding new ones. For instance, we developed four new modules of which one integrates an eye tracking sensor and one facilitates access to image classification models from the tensorflow-hub model zoo. The visualization sink could easily be replaced by a web-based frontend. Likewise, other researchers might contribute to the module ecosystem of the open-source MSP framework. MSP is lightweight, because it has only a few dependencies which enables a quick installation process and simplifies the integration into other software projects. We limit MSP to a concise, but powerful set of concepts and functionalities: module instances can be connected to a processing pipeline with convenience

[6]https://www.tensorflow.org/hub
[7]https://tfhub.dev/google/imagenet/resnet_v2_50/classification/5



```
97.69% - loudspeaker
02.03% - home theater
00.14% - iPod
00.10% - traffic light
00.01% - radio
```

Fig. 4. Screenshot of the visualization sink of our sample application. The user's recent gaze (red dot) and fixation point (green circle) are on a loudspeaker, which is the top-1 prediction for the corresponding image patch (upper left).

functions for starting and stopping them. Implementing modules and connecting them to a functioning pipeline are left to researchers and developers. We offer a selection of basic modules, such as a webcam source and a video output sink, and sample pipelines in our test suite.

MSP was successfully used for realizing two research prototypes that, similar to our sample application, fuse eye movements and a video stream to classify objects at the user's point of regard. Barz et al. [1] implemented an augmented reality system that classifies fixated objects and augments the real objects with virtual labels that stick to them in real-time. They integrated Microsoft's HoloLens 2 with an integrated eye tracking sensor as an MSP source module using the augmented reality eye tracking framework ARETT [12]. In another project, MSP was used to implement a system for automatic detection and annotation of visual attention to areas of interest [3]. Here, the goal was to accelerate and objectify the tedious manual annotation task required for analyzing user studies based on mobile eye tracking.

From using MSP in our projects, we identified limitations related to multiprocessing, networking capabilities, and data synchronization. For instance, MSP offers a wrapper that starts a module in a separate process which enables execution of a processing pipeline across multiple cores. However, the current method for inter-process communication is not efficient and limited to transferring basic datatypes and numpy arrays [8]. Another limitation arises from MSP's networking support: dataframes can be streamed via a network using its networking sink and source modules. To run a processing pipeline across multiple computers in a local network, each pipeline must be started separately. Like multiprocessing, the transfer of dataframes via a network suffers from the inefficient serialization approach. Another limitation is the simplistic approach for data stream synchronization. Currently, modules simply drop outdated samples to avoid propagation of delayed messages.

5 CONCLUSION

We introduced the new multisensor-pipeline (MSP), a lightweight, flexible, and extensible framework for prototyping multimodal-multisensor interaction systems based on real-time sensor input. Its modular and graph-based architecture allows researchers and developers to intuitively build, adapt, and extend stream and event processing pipelines in Python. We demonstrated the effectiveness of MSP by implementing a sample application that crops image patches around a user's fixation point, classifies these patches using a pre-trained deep learning model, and visualizes the results as a real-time video overlay. Our framework can be used to drive future research in multimodal-multisensor user interaction. We plan to enhance the utility of our framework by finalizing the networking and multiprocessing modules, by integrating more advanced data synchronization techniques, and by adding a dashboard for intuitive debugging and monitoring of processing pipelines. In addition, we aim at extending MSP to fully support the multimodal-multisensor interaction framework from Zacharias et al. [28].

ACKNOWLEDGMENTS

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 01JD1811C (GeAR).

REFERENCES

[1] Michael Barz, Sebastian Kapp, Jochen Kuhn, and Daniel Sonntag. 2021. Automatic Recognition and Augmentation of Attended Objects in Real-time using Eye Tracking and a Head-mounted Display. In *ACM Symposium on Eye Tracking Research and Applications (ETRA '21 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/3450341.3458766

[2] Michael Barz and Daniel Sonntag. 2016. Gaze-guided object classification using deep neural networks for attention-based computing. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp Adjunct 2016, Heidelberg, Germany, September 12-16, 2016*, Paul Lukowicz, Antonio Krüger, Andreas Bulling, Youn-Kyung Lim, and Shwetak N. Patel (Eds.). ACM, 253–256. https://doi.org/10.1145/2968219.2971389

[3] Michael Barz and Daniel Sonntag. 2021. Automatic Visual Attention Detection for Mobile Eye Tracking Using Pre-Trained Computer Vision Models and Human Gaze. *Sensors* 21, 12 (Jan. 2021), 4143. https://doi.org/10.3390/s21124143 Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

[4] Tobias Baur, Alexander Heimerl, Florian Lingenfelser, Johannes Wagner, Michel F. Valstar, Björn Schuller, and Elisabeth André. 2020. eXplainable Cooperative Machine Learning with NOVA. (2020). https://doi.org/10.1007/s13218-020-00632-3 Accepted: 2021-04-23T09:34:07Z Publisher: Springer.

[5] Dan Bohus, Sean Andrist, Ashley Feniello, Nick Saw, Mihai Jalobeanu, Patrick Sweeney, Anne Loomis Thompson, and Eric Horvitz. 2021. Platform for Situated Intelligence. *arXiv:2103.15975 [cs]* (March 2021). http://arxiv.org/abs/2103.15975 arXiv: 2103.15975.

[6] Ionut Damian, Michael Dietz, and Elisabeth André. 2018. The SSJ Framework: Augmenting Social Interactions Using Mobile Signal Processing and Live Feedback. *Frontiers in ICT* 5 (2018), 13. https://doi.org/10.3389/fict.2018.00013

[7] Michael Feld, Robert Neßelrath, and Tim Schwartz. 2019. Software Platforms and Toolkits for Building Multimodal Systems and Applications. In *The Handbook of Multimodal-Multisensor Interfaces: Language Processing, Software, Commercialization, and Emerging Directions*. Association for Computing Machinery and Morgan &amp; Claypool, 145–190. https://doi.org/10.1145/3233795.3233801

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. http://image-net.org/challenges/LSVRC/2015/

[10] Alexander Heimerl, Tobias Baur, Florian Lingenfelser, Johannes Wagner, and Elisabeth André. 2019. NOVA - A tool for eXplainable Cooperative Machine Learning. In *2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII)*. 109–115. https://doi.org/10.1109/ACII.2019.8925519 ISSN: 2156-8111.

[11] Kenneth Holmqvist and Richard Andersson. 2011. *Eye tracking: A comprehensive guide to methods, paradigms and measures.* Lund Eye-Tracking Research Institute, Lund, Sweden.

[12] Sebastian Kapp, Michael Barz, Sergey Mukhametov, Daniel Sonntag, and Jochen Kuhn. 2021. ARETT: Augmented Reality Eye Tracking Toolkit for Head Mounted Displays. *Sensors* 21, 6 (March 2021), 2234. https://doi.org/10.3390/s21062234 Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[13] Moritz Kassner, William Patera, and Andreas Bulling. 2014. Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, New York, NY, USA, 1151–1160. https://doi.org/10.1145/2638728.2641695 arXiv: 1405.0006 Series Title: UbiComp '14 Adjunct.

[14] Asterios Katsifodimos and Sebastian Schelter. 2016. Apache Flink: Stream Analytics at Scale. In *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*. 193–193. https://doi.org/10.1109/IC2EW.2016.56

[15] Chia-Yu Li, Daniel Ortega, Dirk Väth, Florian Lux, Lindsey Vanderlyn, Maximilian Schmidt, Michael Neumann, Moritz Völkel, Pavel Denisov, Sabrina Jenne, Zorica Kacarevic, and Ngoc Thang Vu. 2020. ADVISER: A Toolkit for Developing Multi-modal, Multi-domain and Socially-engaged Conversational Agents. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, Online, 279–286. https://doi.org/10.18653/v1/2020.acl-demos.31

[16] Robert Neßelrath. 2016. *SiAM-dp: an open development platform for massively multimodal dialogue systems in cyber-physical environments*. PhD Thesis. Saarland University. https://doi.org/10.22028/D291-26644

[17] Sharon Oviatt and Philip R. Cohen. 2015. *The Paradigm Shift to Multimodality in Contemporary Computer Interfaces*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00636ED1V01Y201503HCI030 Publication Title: Synthesis Lectures on Human-Centered Informatics.

[18] Sharon Oviatt, Björn Schuller, Philip Cohen, Daniel Sonntag, Gerasimos Potamianos, and Antonio Krüger (Eds.). 2019. *The Handbook of Multimodal-Multisensor Interfaces: Language Processing, Software, Commercialization, and Emerging Directions*. Association for Computing Machinery and Morgan & Claypool. https://doi.org/10.1145/3233795

[19] Sharon Oviatt, Björn Schuller, Philip R Cohen, Daniel Sonntag, Gerasimos Potamianos, and Antonio Krüger. 2017. Introduction: Scope, Trends, and Paradigm Shift in the Field of Computer Interfaces. In *The Handbook of Multimodal-Multisensor Interfaces*. Association for Computing Machinery and Morgan & Claypool, New York, NY, USA, 1–15. https://doi.org/10.1145/3015783.3015785

[20] Mihai Polceanu and Christine Lisetti. 2019. Time to Go ONLINE! A Modular Framework for Building Internet-Based Socially Interactive Agents. In *Proceedings of the 19th ACM International Conference on Intelligent Virtual Agents* (Paris, France) *(IVA '19)*. Association for Computing Machinery, New York, NY, USA, 227–229. https://doi.org/10.1145/3308532.3329452

[21] Norbert Reithinger, Simon Bergweiler, Ralf Engel, Gerd Herzog, Norbert Pfleger, Massimo Romanelli, and Daniel Sonntag. 2005. A look under the hood: design and development of the first SmartWeb system demonstrator. In *Proceedings of the 7th International Conference on Multimodal Interfaces, ICMI 2005, Trento, Italy, October 4-6, 2005*, Gianni Lazzari, Fabio Pianesi, James L. Crowley, Kenji Mase, and Sharon L. Oviatt (Eds.). ACM, 159–166. https://doi.org/10.1145/1088463.1088492

[22] Norbert Reithinger and Daniel Sonntag. 2005. An integration framework for a mobile multimodal dialogue system accessing the semantic web. In *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*. ISCA, 841–844. http://www.isca-speech.org/archive/interspeech_2005/i05_0841.html

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y arXiv: 1409.0575 ISBN: 0920-5691.

[24] Nicu Sebe. 2009. Multimodal interfaces: Challenges and perspectives. *Journal of Ambient Intelligence and Smart Environments* 1, 1 (2009), 23–30. https://doi.org/10.3233/AIS-2009-0003 Publisher: IOS Press.

[25] Marcos Serrano, Laurence Nigay, Jean-Yves L. Lawson, Andrew Ramsay, Roderick Murray-Smith, and Sebastian Denef. 2008. The Openinterface Framework: A Tool for Multimodal Interaction. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 3501–3506. https://doi.org/10.1145/1358628.1358881

[26] Daniel Sonntag. 2010. *Ontologies and Adaptivity in Dialogue for Question Answering*. Studies on the Semantic Web, Vol. 4. IOS Press. https://doi.org/10.3233/978-1-61499-338-4-i

[27] Johannes Wagner, Florian Lingenfelser, Tobias Baur, Ionut Damian, Felix Kistler, and Elisabeth André. 2013. The social signal interpretation (SSI) framework: multimodal signal processing and recognition in real-time. In *Proceedings of the 21st ACM international conference on Multimedia (MM '13)*. Association for Computing Machinery, New York, NY, USA, 831–834. https://doi.org/10.1145/2502081.2502223

[28] Jan Zacharias, Michael Barz, and Daniel Sonntag. 2018. A Survey on Deep Learning Toolkits and Libraries for Intelligent User Interfaces. *arXiv:1803.04818 [cs]* (March 2018). http://arxiv.org/abs/1803.04818 arXiv: 1803.04818.