Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming

Ruofei Du
Google Research
San Francisco, CA, USA
me@duruofei.com

Na Li
Google Research
Mountain View, CA, USA
linazhao@google.com

Jing Jin
Google Research
Mountain View, CA, USA
jingjin@google.com

Michelle Carney
Google Research
Mountain View, CA, USA
michellecarney@google.com

Scott Miles
Google Research
Mountain View, CA, USA
sjmiles@google.com

Maria Kleiner
Google Research
Mountain View, CA, USA
mmandlis@google.com

Xiuxiu Yuan
Google Research
Mountain View, CA, USA
xiuxiuyuan@google.com

Yinda Zhang
Google Research
Mountain View, CA, USA
yindaz@google.com

Anuva Kulkarni
Google Research
Mountain View, CA, USA
anuvak@google.com

Xingyu "Bruce" Liu
Google Research
Mountain View, CA, USA
liubruce@google.com

Ahmed Sabie
Google Research
Mountain View, CA, USA
ahmedsabie@google.com

Sergio Orts Escolano
Google Research
San Francisco, CA, USA
sorts@google.com

Abhishek Kar
Google Research
Mountain View, CA, USA
sorts@google.com

Ping Yu
Google Research
Mountain View, CA, USA
piyu@google.com

Ram Iyengar
Google Research
Mountain View, CA, USA
ramiyengar@google.com

Adarsh Kowdle
Google Research
San Francisco, CA, USA
adarshkowdle@google.com

Alex Olwal
Google Research
Mountain View, CA, USA
olwal@acm.org

ABSTRACT

In recent years, there has been a proliferation of multimedia applications that leverage machine learning (ML) for interactive experiences. Prototyping ML-based applications is, however, still challenging, given complex workflows that are not ideal for design and experimentation. To better understand these challenges, we conducted a formative study with seven ML practitioners to gather insights about common ML evaluation workflows.

The study helped us derive six design goals, which informed Rapsai[1], a visual programming platform for rapid and iterative development of end-to-end ML-based multimedia applications. Rapsai features a node-graph editor to facilitate interactive characterization and visualization of ML model performance. Rapsai streamlines end-to-end prototyping with interactive data augmentation and model comparison capabilities in its no-coding environment. Our evaluation of Rapsai in four real-world case studies (N=15) suggests that practitioners can accelerate their workflow, make more informed decisions, analyze strengths and weaknesses, and holistically evaluate model behavior with real-world input.

CCS CONCEPTS

• **Computing methodologies** → Visual analytics; *Machine learning*; • **Software and its engineering** → **Visual languages**.

KEYWORDS

Visual Programming; Node-graph Editor; Deep Neural Networks; Data Augmentation; Deep Learning; Model Comparison; Visual Analytics

ACM Reference Format:
Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu "Bruce" Liu, Ahmed Sabie, Sergio Orts Escolano, Abhishek Kar, Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal. 2023. Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming. In *CHI '23: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, April 23 – 28, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 23 pages. https://doi.org/10.1145/3544548.3581338

1 INTRODUCTION

Recent advances in deep learning [26, 30–32, 40, 50, 52, 58, 74] have enabled the use of a plethora of on-device machine learning (ML)

[1]Rapsai is an abbreviation for *Rapid Application Prototyping System for AI*

CHI '23, April 23 – 28, 2023, Hamburg, Germany

ACM ISBN 978-1-4503-9421-5/23/04…$15.00
https://doi.org/10.1145/3544548.3581338

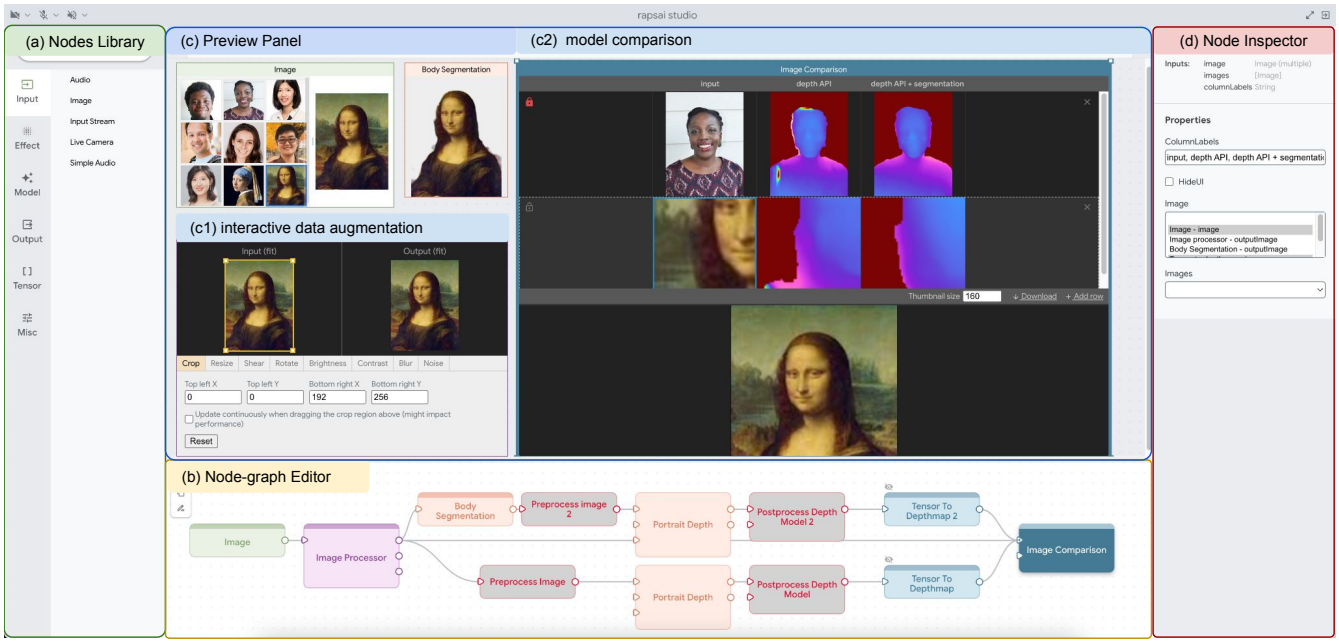


Figure 1: Rapsai empowers machine learning (ML) researchers and practitioners to rapidly build and iterate on real-time ML applications ingesting multimedia data with a visual programming interface. (a-b) users can build new multimedia pipelines by connecting input, effect, models, and output nodes within a node-graph editor; (c) users can interactively evaluate the generality of ML models with (c1) interactive data augmentation, and (c2) qualitative comparison to understand the differences and trade-offs between multiple models. (d) users can change settings of a specific node, *e.g.*, labels of image comparison.

models for real-time multimedia applications. Examples include people segmentation for film production and video conferences [29, 43], depth estimation for 3D reconstruction [22, 33, 85], hand and body tracking for interaction [5, 84], and audio denoising for remote communication [13, 48].

Rapid prototyping and evaluation of ML has gained recent attention in natural language research. Examples include prompt-based [35, 79, 80] and sketch-based [11] prototyping with large language models, visual analytic systems for comparing language classification [47], and image classification models [21, 62]. Practitioners are also increasingly leveraging commercial platforms [8, 78] to deploy and try out cloud-based ML models.

In contrast to language models, development of multimedia applications pose unique challenges, often requiring special-purpose models, with a variety of options to solve problems such as face, body, and hand pose estimation, as well as scene or depth understanding [41]. Perception models that are designed for low-latency, real-time multimedia applications tend to have large variations in input and output possibilities, which makes both visualization and prototyping difficult to generalize.

Given high complexity, the development of high-performance, real-time multimedia prototypes typically requires coordination of a cross-functional team of ML practitioners, such as ML researchers, engineers, designers, and UX researchers. Their collaboration is particularly critical when fine-tuning and evaluating model robustness, characterizing strengths and weaknesses, and inspecting performance in the context of the usage scenario. Iterative development is further complicated by the need to integrate updated models into applications before they can be evaluated.

To better understand this space, we conducted a formative study with seven computer vision researchers, audio ML researchers, and engineers. We gained key insights from their current workflow of model development, evaluation, and deployment: (1) quantitative analysis on training and testing datasets is not sufficient to compare a model's real-world performance to prior art; (2) model deployment is often hindered by corner cases reported by users in the testing phase; (3) building an efficient end-to-end GPU pipeline with other perception models is often beyond the skills of most ML researchers and engineers.

The formative study allowed us to derive six design goals, which informed the development of Rapsai. Rapsai is a visual programming platform (Fig. 1) that streamlines the iterative development of perception pipelines through a node-graph editor interface. It enables users to interactively gain insights into model behavior and assess trade-offs through data augmentation modules that accelerate the systematic comparison of different deep learning models.

We evaluate Rapsai with 15 ML practitioners in four real-world case studies — portrait depth, scene depth, portrait matting, and audio denoising. Our results show that Rapsai has the potential to improve how ML practitioners evaluate model effectiveness, assess strengths and weaknesses, holistically evaluate behavior, and create end-to-end multimedia applications.

Contributions

Our contributions are summarized as follows:

(1) Six design goals for iterative prototyping with perception models, derived from a formative user study (N=7).
(2) System design and implementation of Rapsai, a visual programming platform with interactive data augmentation, model comparison in a node-graph editor, cross-device input/output, and direct deployment of GPU-accelerated deep learning pipelines.
(3) Four case studies with 15 ML researchers and engineers that show the potential of Rapsai to improve rapid prototyping with on-device models.

2 RELATED WORK

Our work is inspired by prior art in visual programming and visual analytics for machine learning models, as well as existing systems for ML experimentation and evaluation.

2.1 Visual Analytics of ML Models

Tools for visual analytics are widely used in model evaluation and debugging. They help researchers gain insights faster, find directions to improve in model architecture, and facilitate comparison and selection of better models. We reviewed the most relevant systems for the model evaluation and debugging task and summarized their features in Appendix Table 1. We further discuss key examples and refer readers to the survey by Hohman *et al.* [28] for a comprehensive review.

Aggregated *vs.* Instance-based visualization. Aggregated visualization provides quantitative metrics over a batch of data, while instance-based visualization provides qualitative details for individual examples, thus uncovering nuances in model quality for corner cases among different models. Such tools play a vital role in many ML tasks, such as visual saliency [12], matting [49], superresolution [76], colorization [2], where subjective scoring is important and ground truth is often absent, or in tasks evaluating model robustness by input perturbation [18]. For example, Ribeiro *et al.* [54] designed LIME, a system that visualizes per-class masks of the classifier results overlaid on the input images. They showed that even users without ML experience can gain insights. Hohman *et al.* [28] emphasizes the importance of visual analytics to support model users (application developers, designers, etc.) who may have limited ML experience with selecting and experimenting with models. More work is needed in the instance-based visualization area so that we can see more successful cases like LIME. However, visualizing multimedia (image and sound) results is not an easy task. The most recent survey on visual analytics techniques for ML [83] highlighted that most research focuses on textual or tabular data, whereas few works study multimedia. We observe two main challenges. First, while LIME successfully showed one type of visualization, there can be many different types of visualizations for multimedia results. We therefore need a set of visualizations to cover different result types in image and sound research. Second, visualizing multimedia results requires domain expertise in graphics and/or audio. Past research [44, 51] point out that multimedia systems often require GPU-based rendering for the systems to be usable in near real-time, since CPU performance can be a bottleneck. We developed Rapsai to be flexible and extensible, allowing for the addition of various visualizations for different perception models and use cases. Additionally, Rapsai addresses the latency issue by providing real-time, GPU-based rendering capabilities.

Comparison of model quality. Model comparison provides insights into differences in model performance. Comparisons can be made at either the aggregated level or instance-based level. Comparison tools operating at the aggregated result level mainly support the comparison of different metrics. In the language domain, Murugesan *et al.* [47] proposed DeepCompare for interactive model comparison using aggregated visualizations like histograms, treemaps of results, and heatmaps of the neuron weights. Johnson *et al.*[36] designed NJM-Vis, a system comparing sentiment models by listing positive and negative sentences containing relevant keywords. Both model developers and model users can develop insights into which model performs better. In the image domain, Spinner *et al.* [62] designed ExplAIner, a plugin for TensorBoard for visualizing the saliency map besides the input image, with positive and negative examples laid out in a row. All these prior works rely on known ground truth. However, we highlight that instance-based visual comparison can be even more important for research areas without ground truth, such as generative modeling [55], because they rely on manual evaluation by human evaluators [7]. One such example is the VASS system by He *et al.* [27], which is specifically designed for semantic segmentation models and can visualize colored masks for different scene objects. Users found the ability to compare the visualized results *"the most insightful part of the tool"*.

Existing visual analytics tools demonstrated that instance-based visualization can provide insights for ML practitioners, but few works explored how it helped users for models without ground truth. We study researchers' practice of how they gain insights from comparing generative model results. Our study includes four different models for images and sounds to be representative of diverse application domains.

2.2 Pipeline *vs.* model interpretability

Existing visual ML analytics tools often focus solely on the model in isolation, with less attention to end-to-end evaluation and debugging. This approach can be a major hurdle when integrating models into applications, particularly in multimedia applications where end-to-end pipelines may include rendering operations (*e.g.*, crop, resize, slice, etc.) and more than one model. Focusing on the model in isolation may also not provide adequate insights into how it will perform with other models, as errors in pre- and post-processing operations and upstream models can impact the final output.

In Rapsai, we examine how model developers and model users gain insights from ML pipelines. We review some popular systems for this purpose. MediaPipe [44] is an efficient framework for building real-time perception pipelines, where each computing unit (*e.g.*, an ML model, tensor operations, or rendering pass) is represented as a node in a MediaPipe graph. Colab [23] is a web-based interactive computing system, adapted from Jupyter Notebook [39] and integrated with cloud-based computation resources. HuggingFace Spaces [78] is a model playground that facilitates experimentation and interaction. It enables researchers to write preprocessing logic

in Python, including the user interface, and to run models using a Colab-like infrastructure. It allows for the creation, sharing, and execution of ML pipelines.

In this work, we study how the existing multimedia systems reviewed in this subsection are used. Previous research on explainable AI [19] has largely focused on *model interpretability*, but we argue that the field should also consider *pipeline interpretability*. The insights from our study informed the Rapsai tool's design to address pipeline interpretability challenges. Our Rapsai evaluation demonstrates its effectiveness in helping model developers and model users gain insights into how models work in end-to-end pipelines.

2.3 Visual Programming of Machine Learning and Graphics Applications

Rapsai leverages visual programming to support complex pipeline authoring. The concept of visual programming and node-graph editors dates back to 1960s, where it was referred to as "a pictorial program" and "man-machine communications" [64]. It has proved to be useful in helping domain-specific experts complete specific difficult programming tasks, such as process control program for oil refinement or circuit design [37, 53]. Even today, these concepts are widely applied in computer-aided design (Blender [6], Maya [4], 3DS Max [3]), game engines (Unity [72]), AR/VR applications (Snap Lens Studio [60], TikTok Effect House [70]), programming learning platforms (Scratch [57]), and data pipeline systems [1].

In the machine learning domain, emerging visual programming systems have significantly empowered application development on many fronts. In natural language research, several works have explored interactive prompt-based prototyping [35, 79, 80] to help model users design with large language models. Each prompt step is visualized as a node and prompts are chained as a connected graph of nodes to represent a custom tailored application of a language model. Wu *et al.* [79] showed how PromptChainer can help model users build language pipelines. Further, Wu *et al.* [80] showed that by explicitly visualizing steps and their flow as oppose to treating the model as a black box, users found the model to be more transparent, debuggable and designable. In multimedia research, tools like Fiebrink's Wekinator [17] help composers and performers use ML in creative practice, and Katan *et al.* [65] use interactive machine learning to engage people with disabilities in musical interface development. Diaz *et al.* [14] use interactive machine learning for game development. Carney *et al.* [9] show how the web-based Teachable Machine can help educators and students learn machine learning classification and train models with intuitive visual tools. Users in 201 countries have created over 125,000 classification models with Teachable Machine.

Few academic works address image or audio pipeline prototyping with ML models. There are commercial graphics platforms that provide some convenient features to build image pipelines, especially for the non-ML components. Shader Graph in Unity [71] provides an intuitive way to fine-tune shader programs, whereas xNode [73] offers developers a general-purpose node-graph framework, and Snap Lens Studio [60] provides augmented reality (AR) developers a node-based scripting system for AR effects.

In contrast to prior art, Rapsai is uniquely designed for supporting image and audio pipeline prototyping for ML practitioners, including both *model developers* and *model users*, like engineers, UX researchers, and prototype creators. Built on top of prior state-of-the-art work and informed by our findings of how practitioners use pipelines for experimentation and prototyping, Rapsai was designed as a holistic solution for prototyping end-to-end image and audio applications. Through Rapsai, we study whether visual programming tools can help practitioners build pipelines and gather insights into new ideas to improve the ML models in their application contexts.

3 FORMATIVE STUDY AND FINDINGS

To unveil use cases and challenges for a rapid prototyping system for multimedia ML applications, we conducted a formative study using semi-structured interviews with a preliminary mock-up (Fig. 2) of the system. The key process and findings of this study are presented here, with detailed protocol available in the appendix.

3.1 Formative Study with Semi-structured Interviews and Mock-up

We recruited seven deep learning practitioners (31–41 years old, $\bar{x} = 34$, $SD = 3.9$) via group email invitations at our institution. The participants, labeled I1–I7, were not familiar with Rapsai and were not involved in this project previously. The individual semi-structured interviews took place remotely via Google Meet and lasted 45-60 minutes. Each interview consisted of three stages: a background survey, an ML process interview, and a discussion of our mock-up.

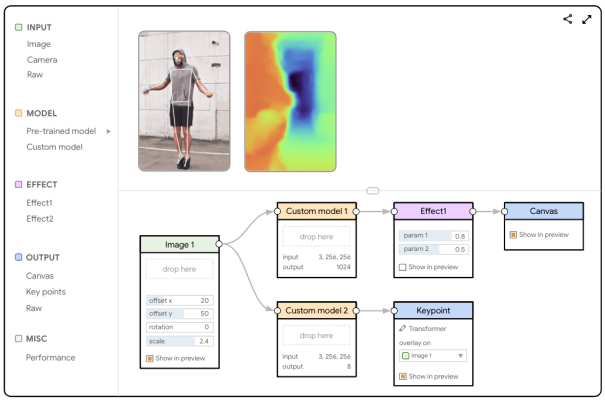


Figure 2: Mock-up interface used in the formative study.

Background Interview. In the first phase (10 minutes), the interviewer asked demographic questions, followed by questions about the participants' professional experiences of training and evaluating neural networks, as well as building multimedia applications. All of our participants reported at least six years of ML experiences (6–12 years, $\bar{x} = 9$, $SD = 2.2$) with image and/or audio data. All of the participants are ML researchers or engineers who self-reported a familiarity with deep learning ranging from $6 - 7$ on a 7-point Likert scale [42].

ML process interview. In the second phase (20 minutes), the interviewer asked the participants about the general procedure of their ML *debugging* and *prototyping* process, the challenges they have encountered when tuning deep learning models and shipping models to products, and the tools they have used to accelerate their evaluation of neural networks. The participants were also asked about what they liked and disliked of their current tools.

Discussion of an Envisioned System. We next presented a mock-up figure (Fig. 2) of our envisioned system to the participants. We asked about their first impressions, their preferred features in the mock-up, what features they would like to add, and any suggestions to improve the system design.

3.2 Tasks and Challenges

Three researchers organized participants' responses with the affinity diagram approach. Across application areas and model types, the participants described three main tasks (**T1**–**T3**) and six challenges (**C1**–**C6**) in their ML workflow when iterating from model development to multimedia applications.

T1. Examine models' robustness and understand error patterns. Robustness is a vital criteria for model launch in production. A model needs to perform well not just on training and testing data, but also on real data in the target application. During the training phase, ML researchers usually perform extensive data augmentation on the training sets to improve robustness. Data augmentation can generate new and diverse examples by translating, cropping, rotating, shearing of images, or adding noise, adjusting pitch or sampling rate of audio. However, during the model evaluation phase, the actual data collected from users and in-the-wild sources often reveal unforeseen issues. Researchers need to examine these issues, get insights into which direction to improve the model, then retrain the model. We summarize two real-world challenges that researchers encountered in this model evaluation process:

- **C1. Lack of data processing for input in-the-wild**. Collecting and preparing data in-the-wild is an essential step in the model evaluation phase. However, it is not trivial to process data for input in-the-wild due to (i) irregular formats, (ii) additional coding, and (iii) requirement of external devices. Firstly, inputs from the wild seldom match the requirement of an ML model. For example, a video input from the webcam may have a different aspect ratio or resolution than the required tensor shape of a model (I1–I4); a microphone recording needs to be converted to a fixed sampling rate as a tensor input (I5–I6). The complexity of processing all kinds of input formats yield longer cycles between finding failure cases and fine-tuning new models. I4 requested that a *"new tool should minimize efforts in coding"*. Thirdly, participants need to find data from external devices that can reflect the real use cases, such as video or image taken from a front-facing camera on mobile devices (I5–I6), or audio taken from a new microphone (I7).
- **C2. Lack of interactive data and model tuning**. Being able to change data and model, as well as interact with the results during the model evaluation phase, is crucial in helping researchers identify directions to improve their models. However, it is not easy to achieve this with the current tools, and some researchers resorted to developing their own tools for this purpose. For example, I2, I4, and I6 rely on custom-programmed pipelines to adjust brightness and contrast, and for adding noise when evaluating models on real data: *"models can be affected by over-fitting, so need to test with a large variety of image augmentations"* (I2). Many participants commented on the limitation of existing tools: *"TensorBoard is limited, <you> can't change the scale of the color map or interact with the point cloud."* (I7), *"3D data is hard to visualize... metrics may not have great instructional meaning for depth data". "I need to verify if <my model> works under different lighting conditions"* (I4).

T2. Qualitatively compare model performance. Quantitative metrics play crucial roles in some domains like image and sound classification. In other fields, such as depth estimation, image generation, and sound denoising, ML practitioners rely on qualitative examples to gain insights and thus desire direct comparison by examples to evaluate model performance.

- **C3. Loss of application context.** Metrics of a sole model lacks the application context from the larger pipeline and are not always reliable for specific domains such as depth estimation. For example, two depth predictors with very similar IMAE[2] or IRMSE[3] scores may result in different behaviors on data in-the-wild: *"Metric doesn't help <in my depth models>, it's always good for all the models, so it's no use. They need human eyes to evaluate."* (I6) *"Loss cannot give you the insights you want, need human reviewers to do quality checks."* (I3) In other examples, two depth estimation models with different metrics may result in comparable results for end-user applications, such as interacting with a 3D model, since animating a mesh does not require perfect depth maps. Hence, stakeholders may select the faster model for deployment.
- **C4. Lack of direct comparison and sharing.** For image-to-image and sound-to-sound models, our participants often want to isolate bad examples of a specific error pattern to discuss with stakeholders, or share an extended challenge set for human annotators to label. They desired tools to allow side-by-side model comparison in varying conditions based on different sources of input, data augmentation, and visualization techniques. In addition, our participants wanted to be able to easily share the challenge set and results with collaborators.

T3. Frequently integrate models into applications. Integrating models into applications is often a long and tedious process. ML practitioners need to adapt their models to the application pipelines (*e.g.*, adjusting input/output resolution, reducing model size, and adding training sets), trade off between overall accuracy and latency, and collect real-world user feedback. They face two major challenges: slow iterations and insufficient controllability of a built pipeline.

- **C5. Slow iterations.** Integrating models into applications often requires a different skill set than an ML researcher has, including front-end programming (*e.g.*, using Java, Swift, or JavaScript) and graphics interloping (*e.g.*, using OpenGL or

[2]IMAE: the mean absolute error of the inverse depth
[3]IRMSE: root mean squared error of the inverse depth

WebGL). I6 explained why a prototyping platform is preferred: *"Test in production app is difficult and time-consuming. Last time, it took months to integrate into Google Photos."*. I2 commented: *"A lot of time goes into visualization of challenge sets, benchmarking, and metrics. Usually takes weeks. "* Worse still, different ML pipelines are built from scratch for different models though they share similar input, processing, and output: *"There's no general solution <so far>, there's different input and output for different models, <I> spent a lot of time in building demos."* (I2). Adapting an unfamiliar model into an existing pipeline is also a hard task: *"You have to build pipeline, and the question is how you adapt those models into your pipeline."* (I7). Nevertheless, the participants needed to re-compile the end-user application with new models and deploy their model on device (*e.g.*, Android phones), which can take 10–30 minutes every time the model changes (I4).

C6. Insufficient controllability. Our participants wanted to examine individual components and intermediate results in a pipeline to understand where the errors come from and the bottleneck is: *"We need to integrate the model with other modules — (to evaluate) can we improve the higher-level model?"* (I7) For researchers who do not build applications by themselves, it often requires detailed communication between teams to figure out where the bug occurred: *"There is often miscommunication between the production team and ours."* (I1)

3.3 Discussion of the Envisioned System

The idea of a visual programming interface for multimedia applications with ML models captivated all participants. I4 commented: *"Good! It's flexible. You have the option to modify the graph live."* I1 remarked: *"It's useful to exchange results across researchers and developers."* Meanwhile, participants posed forward-looking questions: *"How easy will it be to customize the pipeline for our models?"* (I5) *"How can you make sure the inputs and outputs are valid for a node?"* (I7) *"Can we output several models with different quantization and compare?"* (I4)

Finally, participants provided various enhancement suggestions for the proposed system: *"Would be nice to have a video sequence as input and quickly spot low-performant frames."* (I1) *"For the visualizations, flexibility would be good, like hiding some images, annotating images."* (I5) *"It will be helpful to have available data to play with."* (I7) *"It would be nice to magnify pixels and examine at the pixel level."* (I2). We incorporate the relevant suggestions into design goals in the following section.

4 DESIGN GOALS

Informed by the formative studies, we conducted three brainstorming sessions with 12 participants, including researchers, engineers, and designers who have worked on ML-related projects, four of which participated in the formative study. Our intention was to elicit design goals for Rapsai through an understanding of their pain points:

G1. Provide a visual programming platform for rapidly building ML prototypes (C3, C5). Most ML researchers do not have the bandwidth nor capabilities (most researchers train models in Python) to write end-to-end prototypes, which typically requires a wide range of knowledge in the application domain, *e.g.*, Java for Android, Swift for iOS, and JavaScript for the web. They often prefer to refrain from additional coding jobs when building end-to-end prototypes or pipelines. Therefore, it might be beneficial to provide users with a visual programming tool so that they can rapidly build ML applications.

G2. Support real-time multimedia user input in-the-wild (C1, C5). Getting early-stage user adoption and feedback plays a crucial role between model development and end-user applications. However, users' webcam input, microphone recordings and uploaded photos often have different data formats than that required by the model, which requires common operations such as normalization, expanding dimensions and conversion to GPU tensors. Providing an abstraction for these preprocessing steps could provide users with more instant connection between raw input and a generic perception model.

G3. Provide interactive data augmentation (C2, C6). We need to provide ways to support interactive data augmentation and model tuning to quickly gain insights into model performance. This is needed not only by ML researchers, but also for product teams to be able to gain insights and make evaluations on the fly.

G4. Compare model outputs and render results directly side-by-side (C3, C4). Being able to evaluate and compare models on the fly is crucial to day-to-day ML development. We need to support ways to apply different models to the same data for comparison, as well as provide visualizations for comparing different data types.

G5. Share visualization with minimum efforts (C4, C5). A recent survey in model trustworthiness pointed out that model evaluation is a collaborative process, and agreement by domain experts/colleagues is a critical step [10]. However, sharing challenge sets, results, and findings is cumbersome using current tools. We aim to provide easy ways of sharing so that people do not need to install software, or do extra manual work.

G6. Provide off-the-shelf models and datasets (C5, C6). For model comparison, researchers often want to compare with the existing state-of-the-art models. It is time-consuming to find those models, set them up to run in the same pipeline and use the same input data. Hence, we aim to provide ready-to-use pretrained models for popular ML applications.

5 RAPSAI: SYSTEM ARCHITECTURE

We designed Rapsai iteratively over a year with weekly feedback from three teams of ML practitioners from the formative study. Its final design consists of four coordinated panels: (a) Nodes Library, (b) Node-graph Editor, (c) Preview Panel, and (d) Node Inspector. The system is mainly written in JavaScript and leverages TensorFlow.js [59] for ML capabilities, Arcs.js[4] for reactive pipeline, and three.js [69] for graphics rendering. We synchronize the pipeline modification with a remote server powered by Firebase [24]. In

[4]ArcsJs: https://github.com/project-oak/arcsjs-core

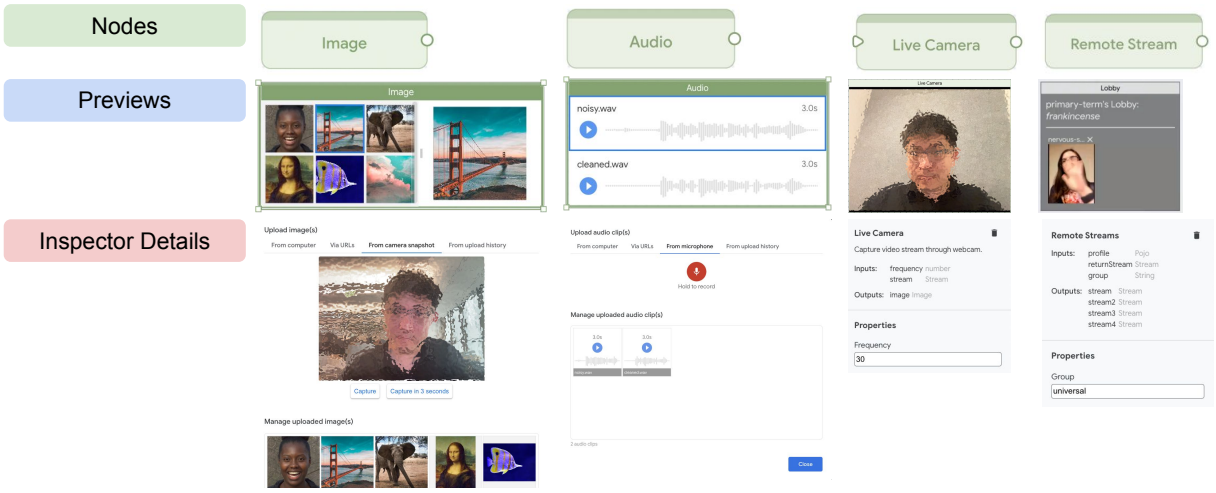


Figure 3: Input nodes allow users to upload local photos, fetch URLs, stream live webcam, record audios, or fetch video streams from a remote device's camera.

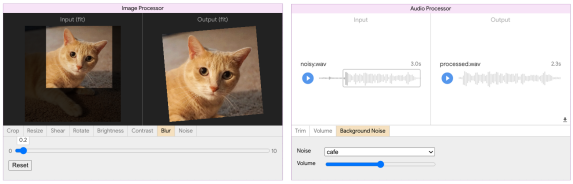
this section, we first detail each component of the interface, and then dive deep into the underlying architecture and discuss our strategies to maximize the performance.

5.1 Nodes Library

In the Nodes Library panel, we provide a comprehensive list of 38 nodes to satisfy the requirements from the four case studies. Here, we define a minimum set of primitive nodes to build the majority of ML pipelines for multimedia applications. We refer readers to Appendix D for the full list of nodes.

Input nodes contain six types of input: *image node*, *video node*, *audio node*, *live camera node*, *live audio node*, *remote stream node* as shown in Fig. 3. To meet our goal **G2**, these nodes support collecting single or continuous image and sound data from various sources, such as cameras and microphones from user devices, uploads from local file systems, and online resources. This design supports data collection in different scenarios. For example, the image node allows users to capture a photo from their webcam, upload from their hard drive, or fetch from a list of remote URLs. The video node allows users to record a video with their external webcam or upload a video from disk or YouTube. The audio node allows users to record sounds from their microphone, or upload audio files from their disk or Internet. The live camera node allows users to use their live camera stream, similar for the live audio node. The remote stream node allows users to stream input from another device (*e.g.*, mobile phone) via WebRTC, by opening a URL of the page with a lobby node, which is detailed in Appendix B.3).

Effect nodes contain interactive nodes for data augmentation and graphical processing nodes to apply shaders to tensors **(G3)**. For example, as shown in Fig. 4, computer vision practitioners can crop and translate a region of interest in the input to verify an image model's invariance to translation. Similarly, they can rotate, shear and resize an image to examine potential biasing issues in the training sets, or apply blur and noise to test a model's robustness. For audio practitioners, we provide augmentation tools such as trimming the audio, changing volume, and adding background noise from a collection of 17 presets. Please refer to Appendix D for the complete set of options.



(a) image processor (b) audio processor

Figure 4: Effect nodes for interactive data augmentation. Rapsai empowers deep learning practitioners to interactively apply common data augmentation methods to data nodes and examine the robustness of their models.

For graphical processing nodes, we offer shader processing and image mixer nodes for building end-to-end vision-graphics pipelines. The shader processing node allows creators, animators, and engineers to write custom fragment shaders to process or create new images on the GPU. To facilitate a large community, we also leverage the ShaderToy API and allow users to bridge a complicated shader into the ML pipeline. This unblocks novel use cases such as creative AR filters for virtual conferences and advanced data augmentation such as distortion and pixelation. The image mixer node is inspired by the layer mixing approach in commercial image processing software, hence allowing users to blend two images in 26 modes[5].

Model nodes contain a list of state-of-the-art pre-trained models, covering a wide range of tasks in image and audio research (**G6**). We also allow users to load their own models with a Custom

[5]Canvas blending modes: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/globalCompositeOperation

Model Runner node. Currently, the model runner supports TensorFlow.js, TFLite, and TFLite Micro models. We also provide some high level task oriented model APIs, such as body segmentation and image classification, from TensorFlow.js models API [6]. In this paper, we focus on discussing use cases with image-to-image and audio-to-audio models only.

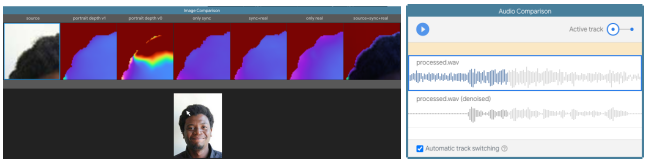


Figure 5: Comparison nodes: (a) Image comparison allows users to hover over an image and examine the zoom-in details across different models. (b) Audio comparison plays audio inputs one-by-one with automatic track switching.

Output nodes provide different ways for visualizing model results. Basic visualizers such as image viewer and audio player allow practitioners to plug them into any node from the pipeline to probe the output, therefore they can easily debug intermediate results. The comparison visualizer offers practitioners an intuitive and interactive way to qualitatively analyze results **(G4)**.

Tensor nodes contain a set of helper utilities for pre- and post-processing, such as converting data from tensor to an image, or applying normalization to the raw tensor output. For example, the "preprocess image" node converts an input image to a 4D tensor — an input format that is required by most image models. The "tensor picker" node allows users to select a tensor from an array of output tensors, and a "tensor postprocess" node allows users to convert a tensor to an image and apply normalization calculators.

The above set of primitive nodes enable perception pipelines of image-to-image and sound-to-sound models. For advanced users, we offer a set of prebuilt nodes to visualize output from different classes of models. For example, a bar viewer to visualize classification results, a webpage viewer to embed online surveys for remote user studies, and WebRTC nodes for getting real-time camera input from mobile devices. We refer readers to the appendix for more details.

5.2 Node-graph Editor

To provide a visual programming platform **(G1)**, we developed a node-graph editor (Fig. 6) with a modular architecture and open sourced its core as ArcsJs[7].

In the node-graph editor, users can drag a node from the nodes library, connect nodes by dragging a line between their connectors, duplicate a node, and remove a node. As shown in 7(a-b), users can explore what node they can connect to by dragging out an edge from the node as a list of candidate nodes will pop up to choose from. This interaction allows for free exploration while ensuring the connection is valid.

[6]TensorFlow.js model APIs: https://github.com/tensorflow/tfjs-models
[7]ArcsJs: https://github.com/project-oak/arcsjs-core

5.3 Preview Panel

The preview panel contains the visualization of each visible node, as well as their interactive controls. For example, in the data augmentation node, users can drag sliders to change an image's brightness, contrast, and blurriness in real time and get instant feedback in the image viewer node. Users can also hover over an image comparison node to observe the magnified views of results side-by-side (Fig. 5). When a node is selected in the node-graph editor, its corresponding visualization is highlighted in the preview panel, and vice versa. Users can double click on a node to rename it.

The preview panel can also generate a unique, shareable URL for collaborative work on an ML pipeline **(G5)**. Anyone with the URL can open the webpage to interact with the ML pipeline, without requiring any software to be installed.

5.4 Node Inspector

Each time the user clicks on a node, the node inspector panel changes accordingly. Using the node inspector, users can change properties, for example, by uploading new test sets to the system in input nodes, altering the model's parameters, or changing the color maps of the visualization.

To allow easy sharing and temporary prototyping, we group properties into two categories, persistent properties and instantaneous properties. Persistent properties are parameters adjustable by the pipeline owner in the node inspector, and are synced with the cloud server. Instantaneous properties are those that can be changed in the preview panel. Those are mainly for interaction by anyone who opens the webpage, and are designed to be ephemeral.

5.5 GPU Pipeline

Rapsai is designed with the goal of real-time performance for frequently comparing models and adapting to real-time multimedia applications. Hence we leverage GPU computing in every stage of the pipeline.

(1) For model inference, Rapsai automatically creates an off-screen WebGL context and leverages TensorFlow.js for evaluating the uploaded Keras or Graphdef model.
(2) In data augmentation nodes (image processing), we leverage a hardware-accelerated HTML canvas to process the image or video in real time instead of using CPU arrays.
(3) In shader processing nodes, we create WebGL canvases for visualizing the results with fragment shaders. However, audio mixing and changing volume still runs on the CPU.

6 CASE STUDIES

We conducted four case studies to evaluate Rapsai's ability to help ML practitioners acquire qualitative insights into ML models in end-to-end pipelines. We worked closely with four computer vision and audio teams. We performed case studies with 15 participants working at Google, of which five were team leads who had offered continuous feedback to Rapsai during its development, whereas the remaining participants had no prior involvement. We interviewed team leads to delve deeply into how Rapsai could aid in their model development and address product-related issues, whereas novice users could reveal insights regarding usability and additional pain points from different perspectives.

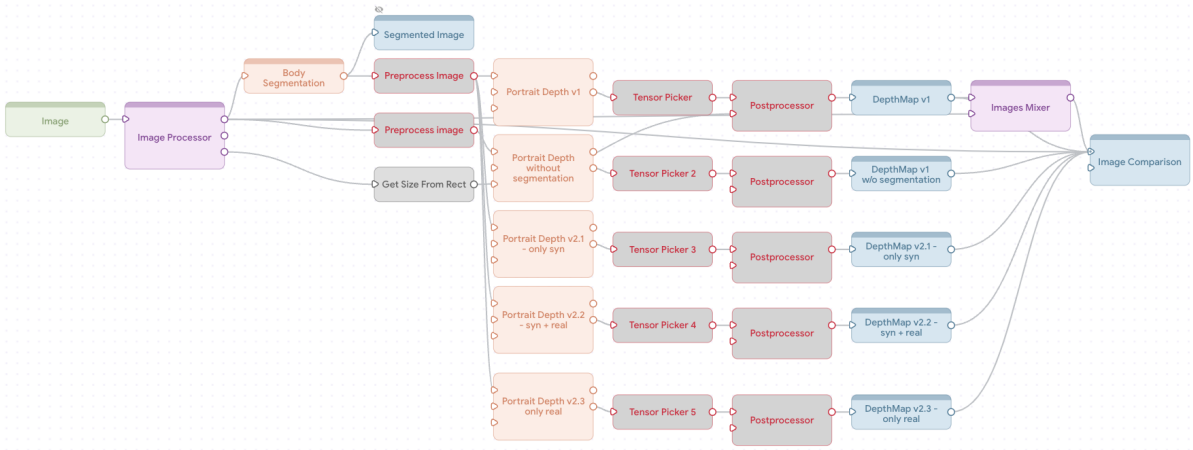
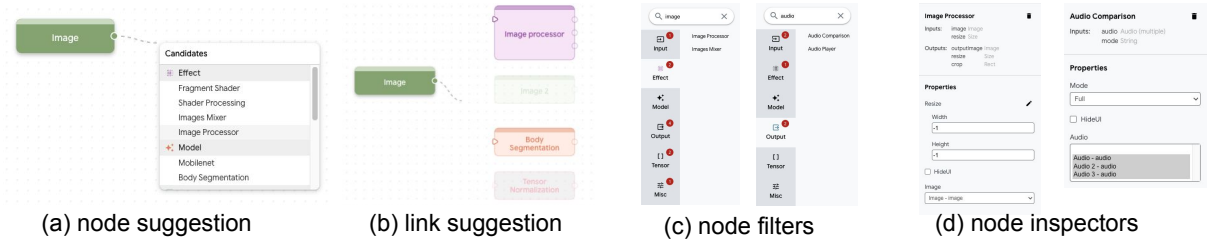


Figure 6: Node-graph editor in Rapsai allow users to efficiently build a perception pipeline with interaction data augmentation and model comparison. This figure shows the corresponding node-graph pipeline to render Fig. 5.



(a) node suggestion (b) link suggestion (c) node filters (d) node inspectors

Figure 7: Examples of Rapsai's node-graph editor and inspectors: (a) when the user drag-and-drops an edge from a node to blank area in the editor, Rapsai automatically suggests a list of compatible nodes; (b) invalid nodes are greyed out to prevent users from programming errors; (c) users can filter nodes with keywords in the node library; (d) users can change node properties on the rightmost node inspector panel.

6.1 Study Protocol and Demographics

We recruited 15 participants (28–56 years old, $\bar{x} = 37.7$, $SD = 7.0$) via email invitations within Google, labeled P1–P15. Eight studies took place in a quiet room with a 27-inch display and the participant's laptop, while the rest were conducted remotely with screen sharing in video conferences. Eight participants were ML researchers who routinely train models in their day-to-day work, and nine participants prototyped multimedia demos for models developed by themselves or their team. 13 participants reported longer than four years of experiences in ML. 12 participants ranked their familiarity with deep learning as five or more, on a seven-point Likert scale [42]. The session for each participant lasted 55–60 minutes and consisted of four stages; a background interview (6.1 ± 0.8 min), a video tutorial (4 min), a visual analytics procedure using Rapsai with semi-structured interviews (39.4 ± 4.6 min), and a discussion of Rapsai and future perception prototyping platforms (10.2 ± 2.0 min). Two interviewers hosted each session with one interviewer conducting the interview and the other taking notes. Onsite participants interacted with Rapsai alongside the interviewers, while remote participants shared their screen and followed instructions from the interviewer.

After the case studies, we kept Rapsai available to all Google employees and distributed an exit survey to all participants. They rated their experiences using Rapsai and Colab, a widely used experimentation platform by all the participants, along various dimensions. We detail the questions in the exit survey and its limitations in the appendix E.3.

The background interview employs a shorter list of questions from the formative study, in which participants reported a wider range of ML experiences (1–26 years, $\bar{x} = 6.8$, $SD = 6.0$). We first played a tutorial video to familiarize participants with Rapsai, then instructed them to perform three tasks: (1) build a new multimedia pipeline with Rapsai, (2) interact with the data augmentation node and identify advantages and disadvantages of the ML models, (3) compare two or more ML models, use examples to justify the preferred one, and discuss their findings. The pipeline used in the tasks varied depending on the areas of focus for the different team members.

6.2 CS1: Portrait Depth with Relighting Effects

The first case study investigates the portrait depth pipeline, which uses two publicly-available models from TensorFlow Hub: AR portrait depth [68] and Mediapipe segmentation [45], as well as three

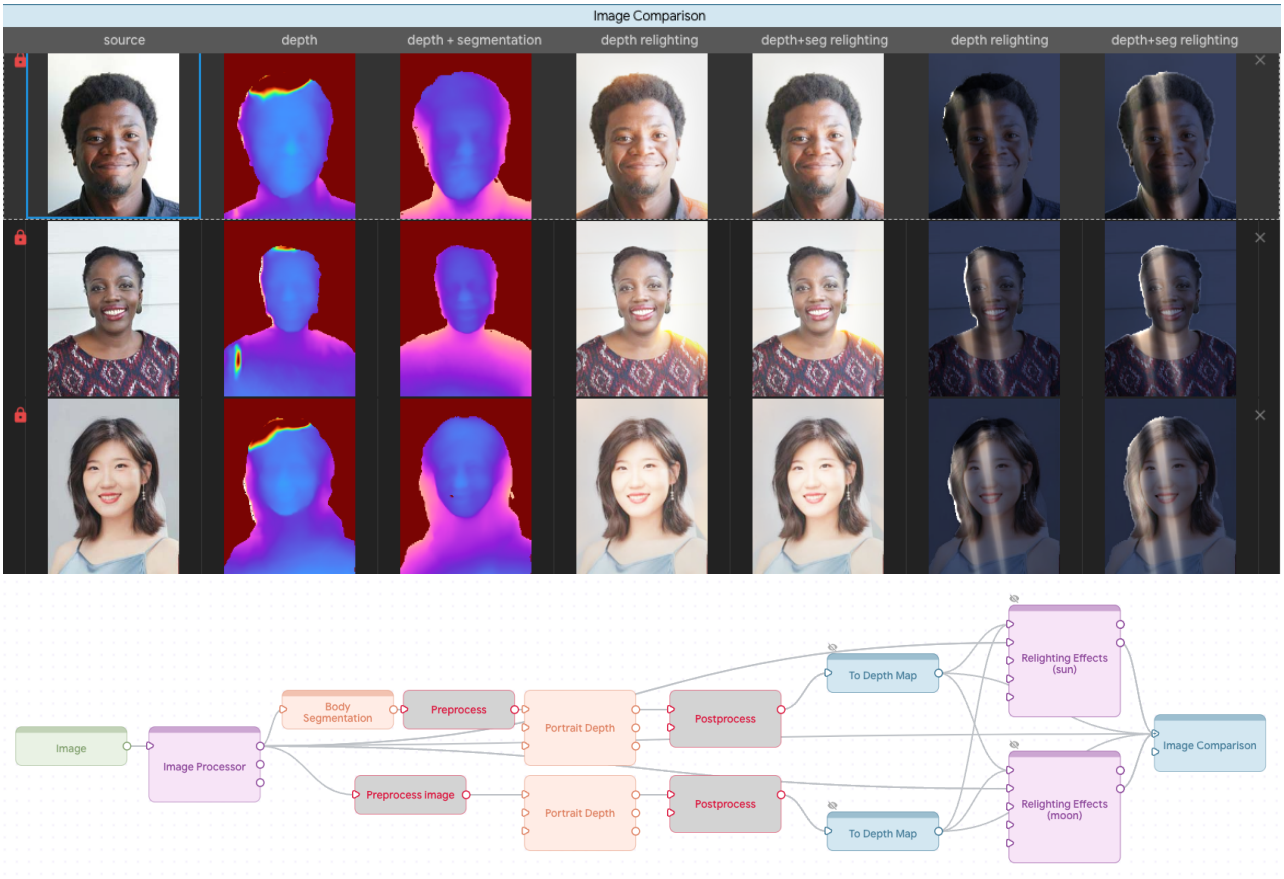


Figure 8: The Portrait Depth pipeline connects a body segmentation model, four portrait depth estimation models, two relighting shader effects for end-user applications and comparisons with respect to the source. For simplicity, we only show results of two models here. Please refer to the supplementary for more results.

portrait depth estimation models developed by the participants. The depth estimation takes a single color portrait image as the input and produces a depth map, which estimates per-pixel distance to the camera. We invited 4 participants (P1, P5, P7, P10) in CS1, where P5 and P10 were not involved in model development.

Specifically in CS1, we found that all portrait depth models perform much better when using a body segmentation model first, as evidenced in Fig. 8. Hence, we instruct all participants to compare the models with and without the segmentation node. However, in conventional ML pipeline development it typically requires hours of extra coding to bridge an ML model to the pipeline, which disincentives engineers from trying out new ML models. Participants also compared five ML portrait depth pipelines with different augmentation techniques, as shown in Fig. 5 and in the appendix.

Generating depth maps from RGB images is never the end of model development. Previously, we demonstrated that portrait depth can be used to render 3D photos [16] in real time while in Fig. 8, we compare different depth pipelines for relighting effects of portrait images that are not included in the training set.

6.3 CS2: Scene Depth for Visual Effects

The second study uses two internal scene depth estimation models developed by the team. Similar to the portrait depth model, the scene depth model estimates a depth map given a generic color image and can be further applied to a wide range of augmented reality applications such as occlusion-aware rendering, rain effects, fog effects [15]. In Fig. 9, we present one pipeline that generates real-time depth-aware fog effect [15] from an input RGB image. Four participants (P2, P4, P8, P9) were invited for CS2 and only P2 was involved in the model development.

In Rapsai, we provide a variety of different color maps[8] for visualizing grayscale depth maps, including Turbo, TurboPlus, Inferno, Magma, Plasma, Viridis, to help users to observe details, estimate quantitative values, and notice error patterns.

6.4 CS3: Alpha Matting for Virtual Conferences

Accurate segmentation, also known as alpha matting, is the vital key to a wide range of applications such as virtual backgrounds in

[8] Colormaps adopted in Rapsai: https://www.shadertoy.com/view/7slfRX

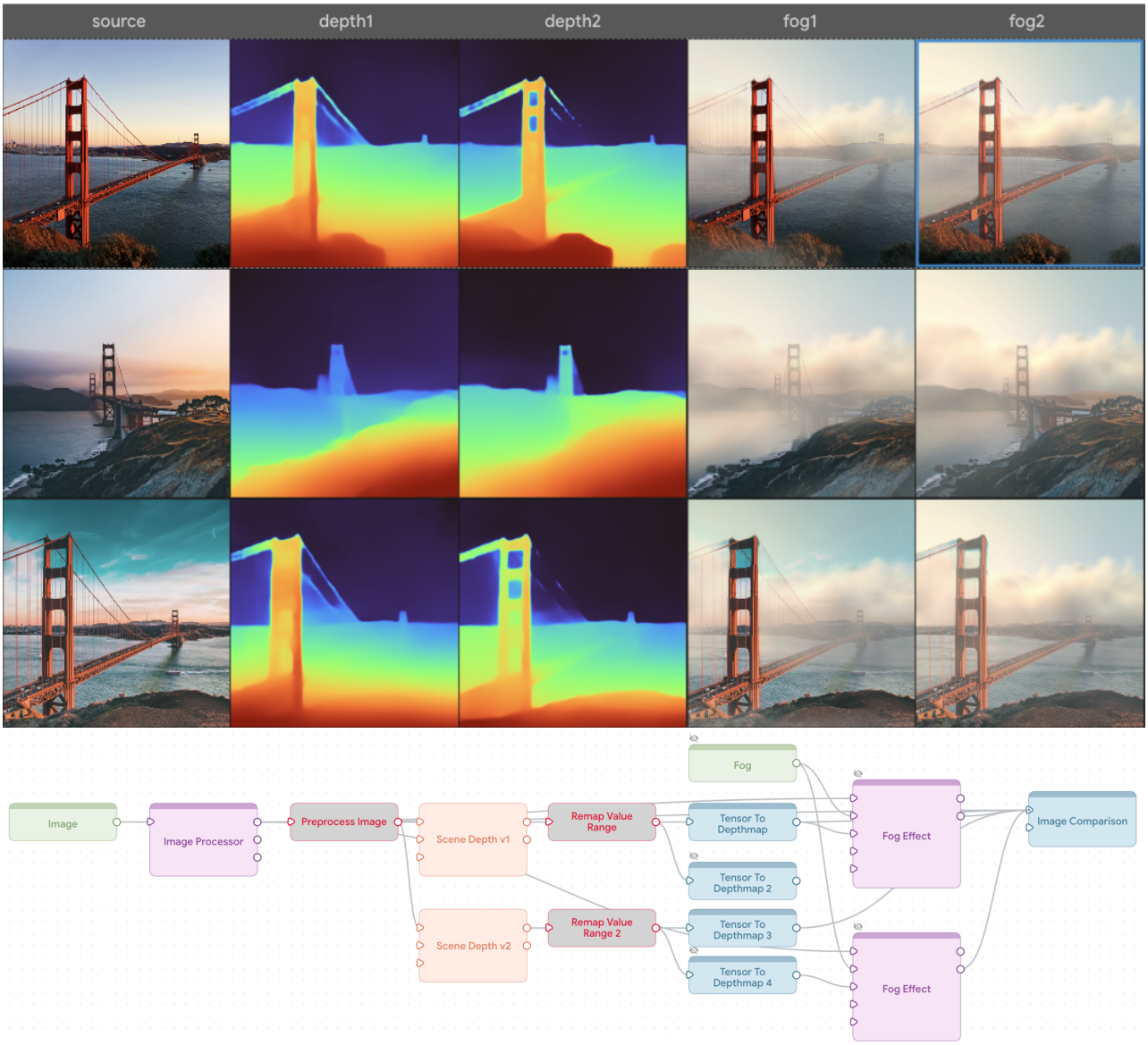


Figure 9: The Scene Depth pipeline leverages a data augmentation node, two scene depth estimation nodes, and a shader processing node for depth-based visual effects such as fog effects.

remote conferences and film production [81], photo relighting [49], and 3D face synthesis [66].

In this case study, we use the aforementioned publicly-available Mediapipe segmentation model and two (alpha) matting models [49] for improving the segmentation results in video conferences. The matting model requires two inputs of both the original image and a rough segmentation mask from a body segmentation model, then outputs a refined segmentation mask, making it complicated to examine the weaknesses in the end-to-end pipeline, as shown in Fig. 10. In addition, evaluating matting models require extensive feedback from users with different virtual backgrounds. Hence, we place a shader node in the end to mix input image, background image, and the matting results. Two participants (P3, P6) were invited for CS3 while only P3 was involved in the model development. We also invited P2 from CS2 for an additional round to try the Matting pipeline since P2 was interested in mixing the scene depth model with the matting model. They accidentally discovered that depth was predicted as a radial gradient with solid background.

6.5 CS4: Audio Denoising for Communication

In contrast to the three visual case studies, this fourth case study investigates two sound-to-sound models for an audio denoising task. Audio denoising, which aims to remove background noise,

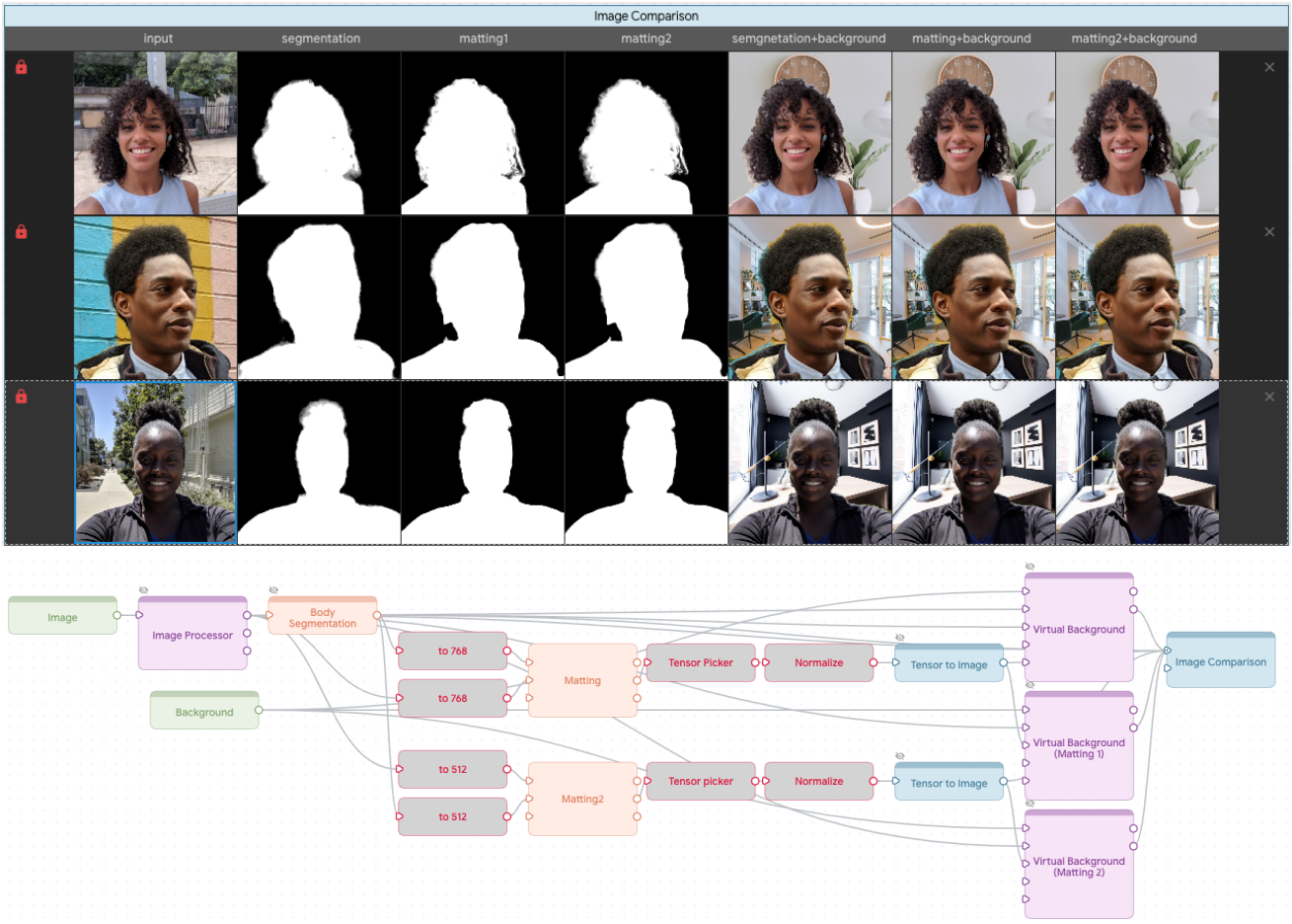


Figure 10: The matting pipeline connects a body segmentation model together with two matting models, and uses a shader processing node for virtual background replacement application.

has gained popularity in many real-time applications in recent years [61]. Five participants (P11–P15) were invited and instructed to build pipelines as shown in Fig. 11.

Specifically in CS4, we embedded a webpage node to collect qualitative feedback regarding model performance in audio denoising. Interestingly, not all participants agreed that the updated audio denoising version was better. With different background noising mixed into the user's recordings, some participants found that the updated model compressed their voice too much, or preferred to have more background context in their audio calls.

7 FINDINGS AND DISCUSSION

Through observing five ML experts exploring the performance of *their own* models, and ten other ML practitioners conducting perception prototyping and model analysis, we found that Rapsai helped facilitate rapid and deeper understanding of model benefits and trade-offs. Overall, all participants were able to make insights about the differences between the models and select the better model. All participants (13/15) who work on generative models (*e.g.*, image-to-image, sound-to-sound) find it useful in their workflow, while two researchers who worked on classification tasks find it less useful in daily development. Since Rapsai is not designed for classification tasks, we discuss their feedback and present how Rapsai could be extended for classification tasks in Appendix C.3. We next discuss our findings from the exit survey, as well as four major insights from the case studies.

7.1 Rapsai *vs.* Colab: Less Control but More Transparent and Collaborative

Fig. 12 depicts the quantitative results of our exit survey. We ran Mann-Whitney's U tests to evaluate the difference between Rapsai and Colab in the responses of our seven-point Likert scale questions. We found significant effects where users rated Rapsai more *transparent* about how it arrives at its final results (Rapsai 6.13 ± 0.88 *vs.* Colab 5.0 ± 0.88, $z = -3.09$, $p < .005$) and more *collaborative* with users to come up with the outputs (Rapsai 5.73 ± 1.23 *vs.* Colab 4.15 ± 1.43, $z = -2.86$, $p < .005$). Participants were satisfied with final results from both systems (Rapsai 6.00 ± 0.88 *vs.* Colab 5.46 ± 1.12, $z = -1.47$, $p = 0.15$).

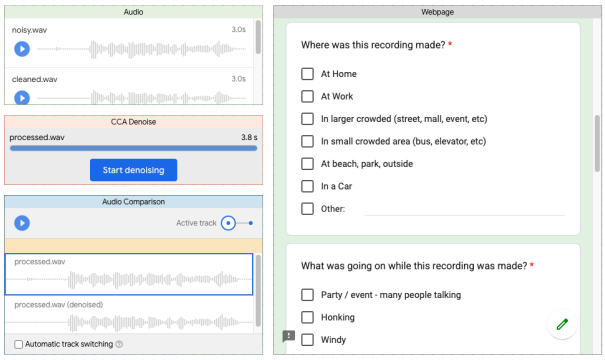


Figure 11: The audio denoising pipeline connects an audio input node, an audio processing node for data augmentation, two denoising models, an audio comparison node, and a webpage node for collecting user feedback. With a prebuilt pipeline, users were able to record their voice at home, compare the performance of two models and submit qualitative feedback directly to the model developers via webpages such as Google Forms.

However, most participants regarded Colab as a superior tool for assisting them in thinking deeper into how to complete the task (Rapsai 5.60 ± 1.72 *vs.* Colab 6.38 ± 0.97, $z = -0.97$, $p = 0.35$) and gave them greater control over the system (Rapsai 5.93 ± 0.91 *vs.* Colab 6.38 ± 1.09, $z = -0.91$, $p = 0.36$). This is largely due to all the participants being expert users and familiar with testing models by programming: "*Colab is more familiar and gives higher control than Rapsai's graphical interface. But Rapsai is quicker to use.*" (P12). "*Rapsai is great for quick visualizations of intermediate outputs in a pipelined ML system where the individual modules are fairly standard and have predictable inputs/outputs. Colab allows for arbitrarily complex Python code and control over the model/input/outputs which can be helpful for complex applications but sometimes tedious for simple pipelines.*" (P2).

Participants' preference for Rapsai or Colab varied based on the usage scenario. P15 commented: *"If I needed to set up training of a standard model or small variant using pre-existing data, Rapsai would be ideal. If I wanted to try some novel architecture (e.g., custom loss function), I would have to resort to Colab or similar."*. P3 remarked on different target user groups: *"Raspsai is more visual and intuitive, Colab is more flexible but also requires having programming skills, so it's not for everyone. Rapsai allows for faster prototyping and results visualization."*. P9 commented on using Rapsai and Colab for different phases in model development: *"I prefer to use Colab at the beginning stage for debugging. When the architecture of the model is almost fixed and <I> need to compare the models which are trained or built from different configurations, I will consider using Rapsai."*

7.2 Rapsai Accelerates Creating and Mixing Multimedia Pipelines With ML Models

Rapsai facilitates development of machine learning-based multimedia applications. We discovered that building a real-time perception pipeline is often beyond the expertise of ML researchers. Their current workflow shows great pain points, which often involves substantial communication and arduous hand-holding across teams. After watching a 4-minute tutorial video on Rapsai, all participants were able to build a custom pipeline from scratch within 15 minutes (avg=10.72, std=2.14). Participants usually spent less than five minutes getting the initial results (avg=3.98, std=1.95), then were trying out different input and output for the pipeline. Participants appreciate its intuitive interaction and visualization: *"It's great that I don't need to do much to have a visual pipeline. I have to do a lot of the demo myself."* (P4). *"Using a video <as input> helps me get a cross-time feel of how the model performance varies, which is hard to capture with metrics."* (P10). Moreover, participants find it useful for comprehending models' distinctions: *"It's a convenient tool for us to understand the difference between various models and speeds up the testing cycle with a very friendly-to-use interface."* (P1).

In the exit survey, all the participants commented that it will take them much shorter time to build pipelines in Rapsai than Colab. P6 commented: *"I spent about half a minute to create an image classification pipeline, and I spent 2–3 minutes to build a depth estimation pipeline from scratch, since it took some time to figure out how to preprocess the input and visualize the output... while Colab is more flexible for different tasks, I guess it could range from 1 hour to a day or two."*. P13 commented: *"In my case, I started from an existing template but overall it was quite fast, I'd say less than 5 min."*.

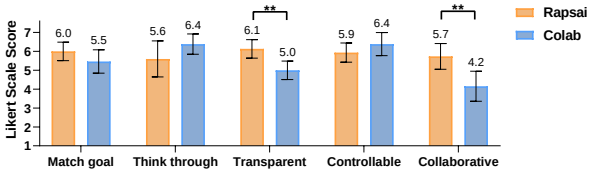


Figure 12: Participants' ratings on a seven-point Likert scale (details in Appendix E.3), accompanied by error bars representing standard deviations. Participants regarded Rapsai to be more *transparent* and *collaborative* than Colab, whereas Colab assisted users in *thinking through* the task and controlling the pipeline more effectively through programming. We found significant effects on the *transparent* and *collaborative* dimensions ($p \leq 0.01$) via Mann-Whitney's U tests.

Rapsai fosters innovative applications of ML models by mixing models and graphics in node-graph editors. With readily available models and example pipelines, ML practitioners are more inclined to experiment with novel ideas. For instance, P2 tried to connect a body segmentation model with a scene depth model and was surprised to find the radial effects in depth prediction. P8 mixed scene depth API, shader effects, with the matting models to create a pipeline of people segmentation with foggy background.

7.3 Rapsai Can Assist in Identifying Issues with ML Models and Training Sets

Rapsai's interactive data augmentation node can help ML researchers identify unforeseen architectural issues. For instance, in the data augmentation stage of CS1, P1 was surprised to find that adding noise had a huge impact on the performance of

the portrait depth model. P1 related this to the small kernel size (3×3), therefore recommended the team to increase the kernel size or applying denoising to the application pipeline. In CS2, P10 found a significant performance issue with the contrast slider and derived that the issue could be the receptive field: *"It can help me understand how I should change the model architecture and what training examples to add".*

Rapsai can further identify biased augmentation of the training sets. ML practitioners also find Rapsai helpful for detecting training process flaws. In CS3, P3 discovered that one model is more sensitive to blurriness than the other: *"It gives me an intuition about which data augmentation operations that my model is more sensitive, then I can go back to my training pipeline, maybe increase the amount of data augmentation for those specific steps that are making my model more sensitive."* In CS2, P8 commented after completing the data augmentation task: *"Comparing various noise parameters in the input to a model is useful to identify augmentation bias. "*

Rapsai helps determining whether more training data is required. Researchers also found Rapsai useful to streamline their workflow for augmenting data in-the-wild. In CS2, P2 appreciated Rapsai's simplicity: *"You can quickly explore many other hypotheses — is this going to mess up my model? In Colab, I need to have many hypotheses in mind, then code for that specific use case. I think the best part is that this is composable."* (P2). In CS4, P15 favored the "background noise mixing" mode: *"It's a fair amount of work to add some background noise, I have a script, but then every time I have to find that script and modify it. I've always done this in a one-off way. It's simple but also very time consuming. This is very convenient.".*

Rapsai can help identify edge cases using inputs in the wild. The majority of the participants found Rapsai useful in identifying edge cases using inputs in the wild. By uploading photos in the wild, ML researchers were able to identify challenging edge cases with transparent objects and long-range depth in CS2: *"the bottle is a good example to show the difference... here the mountain is much farther away from the thing, and the sky should have values of 0, I think <but it didn't>"* (P4). Using the data augmentation node, participants were able to identify edge cases of rotation, brightness, and inconsistency over cropped regions: *"I can manipulate the brightness to see when the model fails."* (P2). Despite that a model generally outperforms another model, researchers found contradictory cases using Rapsai with the data augmentation node: *"Interesting... one model should perform better than the other but performed worse when the audio volume is <low>."* (P13 in CS4).

7.4 Rapsai Helps Model Selection, Learning From Pipelines, and Study deployment

Rapsai provides qualitative evidence to select the optimal model. With the comparison nodes, Rapsai supports ML researchers in model selection with comparable metrics, as metrics in the domain of depth or matting are not necessarily reliable: *"Metrics can be the same even though somehow there might be errors. "* (P1). When two models have similar performance, stakeholders may not always choose the model with the superior metrics, but rather the model that best matches their use case: *"There are some times when you don't want all noise cancellation. People sometimes prefer audio with less noise cancellation because you want some context."* (P11) .

Rapsai assists ML practitioners in visually summarizing the capabilities of ML models with positive and negative examples. For instance, P6 found differences in a model's capabilities: *"The side face is not correct for one model, and good for the other.".* P10 found a good range of aspect ratios with the cropping tool: *"Cropping helps me decide how to crop to best use the model.".*

Sharing qualitative findings in presentations and publications. Eight of the 15 ML practitioners we interviewed spent over four hours per week analyzing individual cases and relied largely on qualitative results to compare model performance: *"I spend hours per week examining visual results by zooming in and zooming out."* (P9). They share comparative results with coworkers using external tools such as Microsoft PowerPoint, Adobe Illustrator, or Google Slides. Fortunately, Rapsai streamlines this process: *"It helps presentation. It takes days to prepare the best examples."* (P5). *"This is helpful to visualize the current capabilities of our model. We are working with potential partners and it is useful for them to be able to try out the model."* (P10) In addition, researchers find it useful to gather feedback: *"It's useful for getting quick feedback. Cutting time to need to run Colab or run a Mediapipe graph."* (P14)

Visual exploration of end-to-end pipeline for novice prototyping practitioners. Rapsai provides senior ML practitioners a convenient tool to demonstrate an end-to-end multimedia pipeline: *"I can use this to demonstrate the envisioned pipeline to <a junior engineer> and it clarifies the ambiguities."* (P6). Meanwhile, it helps junior ML practitioners to quickly build a multimedia pipeline without diving deep into the production codebase for integration: *"It's much faster than building prototypes before, previous experience is fragmented."* (P7).

Enable quick remote field study in the wild. Rapsai can also be used to rapidly deploy a remote in-the-wild field study. Before deploying an ML model (*e.g.*, denoising) to production, it often requires extensive qualitative user feedback with real-world user data. However, existing survey tools such as Google Forms and Amazon Mechanical Turk cannot easily record live audio from users and then run ML models within the page. In Rapsai, users can easily set up a model comparison node as well as a webpage node with a Google Form embedded, which addresses these needs: *"Recording from microphone is very good, it will be very fast turn around if I want to try it in real life. I need it to share with the <product managers> and researchers for user study."* (P14).

8 LIMITATIONS AND FUTURE WORK

In this section, we discuss the limitations of our study and current system, and provide future plans to address them where applicable.

Limitations of case study. Our case study and surveys could have been conducted more effectively by comparing Rapsai to a "sandbox" system, whose UI resembles Rapsai's but lacks the node-graph editor feature. Note that it is hard to attribute the success of Rapsai to individual features in it. The sample size (N=15) of our study with ML practitioners within the same organization may not be representative of the AI+HCI community as a whole. We envision that by providing a few templates with tutorials, Rapsai can further lower the barrier for entry-level users to create multimedia

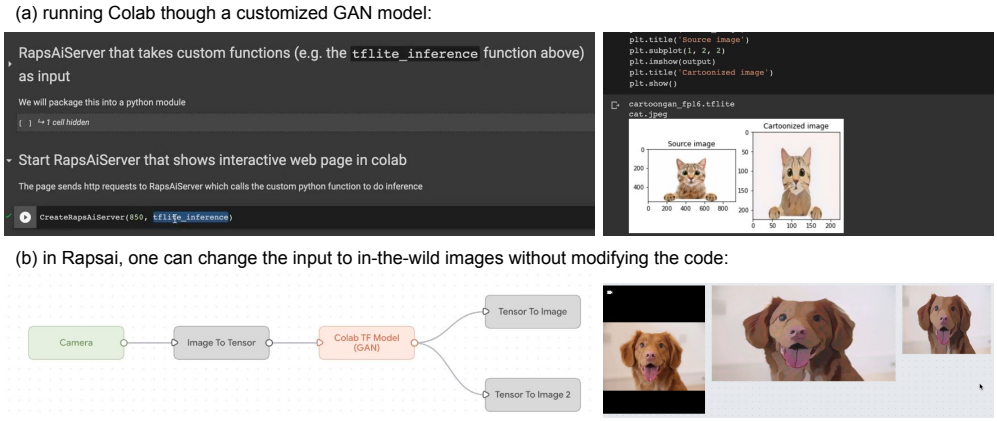


Figure 13: Experimental colab node that supports cloud-based inference of custom Python models.

pipelines. We expect that a future long-term deployment study with both novice and expert users will capture the merits and drawbacks of Rapsai in greater detail.

Lack of support for PyTorch and cloud-hosted models. As a first step in a visual programming platform that bridges model development and multimedia applications, Rapsai leverages TensorFlow.js and offers utilities to convert TensorFlow models. It does not support native execution of PyTorch models. For cloud-hosted models that require a long inference time, Rapsai may not be the ideal solution as interactivity may not be of interest with regard to the execution latency. As a workaround shown in Fig. 13, we experimented with a Colab node that supports running custom Python scripts with Rapsai inputs, and outputting results back to Rapsai. However, the extra 1–3 second latency prevents Rapsai to perform interactive data augmentation for Colab-hosted models. In the future, we aim to support PyTorch and cloud-hosted models via community-contributed nodes, whereas a node could be linked to a remote server and fetch results by executing a block in the Colab notebook.

Generalizability to text and 3D data types. In this work, we focused primarily on multimedia data including images, video, and audio. More work would need to be done to better integrate text and 3D data types into Rapsai. Users cannot add new custom tensor logic without contributing a new code in Rapsai. Recent advances in neural rendering [46] also require novel and efficient rendering pipelines on the web, which is out of scope for Rapsai in its current stage. As an initial step, we present a preliminary example of bridging large language models and graphical pipelines in Appendix C.1.

Closer integration with the training pipeline and aggregated metrics. Systems like UMLAUT [56] have shown that interactive visualizations during training can provide early insights during model development. Rapsai is primarily designed for the testing and prototyping phases. Currently, it does not provide a visual summary of the model's performance over the entire dataset, but rather complements existing pipelines such as TensorBoard and eliminates the need to write code substantially in Colab. In the future, we hope to enable model researchers to being able to interactively analyze their models in Rapsai during the training phase to identify data augmentation or model issues sooner. Also by integrating into the training pipeline, Rapsai will be able to provide aggregated metrics.

Extended visualization libraries from different ML teams. Rapsai implemented the most commonly used nodes based on feedback from four multimedia ML teams, thus supporting image viewer, depth map visualization, shader visualization, audio player, and comparison nodes. However, this is still far from a complete set of data formats in the wild. For example, P1 and P5 would like to have point cloud visualization for their 3D models. P2 and P4 wanted to adjust hue in the interactive augmentation node. Although that can be achieved through a custom shader, P2 and P4 lack shader programming experience. In the future, we would like to open source Rapsai's infrastructure to allow deep learning practitioners to advance visual programming and analytics of end-to-end perception pipelines.

9 CONCLUSION

In this paper, we introduced Rapsai, a visual programming platform for interactive data augmentation, model comparison, and prototyping of multimedia ML applications. The design of Rapsai was informed by interviews and brainstorming sessions with perception ML practitioners, and was iteratively improved with experts. We evaluated Rapsai in four case studies with 15 deep learning researchers who built multimedia pipelines from scratch and compared different models. Analysts were able to holistically select the best model and further explain their strengths and weaknesses through examples. In addition, the unique data augmentation and qualitative comparison nodes were shown to enable new approaches to test robustness and share results in presentations and publications. In future work, we plan to extend the framework to support text and 3D data and integrate more closely with the training pipeline and cloud-hosted models.

ACKNOWLEDGMENTS

We would like to extend our thanks to Jun Zhang and Satya Amarapalli for a few early-stage prototypes, and Sarah Heimlich for serving as a 20% program manager, Eric Turner and Shahram Izadi for

reviewing the manuscript, Sean Fanello, Danhang Tang, Stephanie Debats, Walter Korman, and Anne Menini for providing initial feedback for the manuscript. We would also like to thank our reviewers for their insightful feedback.

REFERENCES

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, et al. 2015. The Dataflow Model: a Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. (2015). https://doi.org/10.14778/2824032.2824076

[2] Saeed Anwar, Muhammad Tahir, Chongyi Li, Ajmal Mian, Fahad Shahbaz Khan, and Abdul Wahab Muzaffar. 2020. Image Colorization: a Survey and Dataset. *ArXiv Preprint ArXiv:2008.10774* (2020). https://arxiv.org/pdf/2008.10774

[3] Autodesk. 2022. 3DS Max. https://www.autodesk.com/products/3ds-max

[4] Autodesk. 2022. Maya. https://www.autodesk.com/products/maya/overview

[5] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. 2020. Blazepose: On-Device Real-Time Body Pose Tracking. *ArXiv Preprint ArXiv:2006.10204* (2020). https://arxiv.org/pdf/2006.10204

[6] Blender. 2022. Blender. https://www.blender.org/

[7] Ali Borji. 2021. Pros and Cons of GAN Evaluation Measures: New Developments. https://doi.org/10.48550/arXiv.2103.09396

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901. https://doi.org/10.48550/arXiv.2005.14165

[9] Michelle Carney, Barron Webster, Irene Alvarado, Kyle Phillips, Noura Howell, Jordan Griffith, Jonas Jongejan, Amit Pitaru, and Alexander Chen. 2020. Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3334480.3382839

[10] Angelos Chatzimparmpas, Rafael Messias Martins, Ilir Jusufi, Kostiantyn Kucher, Fabrice Rossi, and Andreas Kerren. 2020. The State of the Art in Enhancing Trust in Machine Learning Models With the Use of Visualizations. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, Wiley Online Library, 713–756. https://doi.org/10.1111/cgf.14034

[11] John Joon Young Chung, Wooseok Kim, Kang Min Yoo, Hwaran Lee, Eytan Adar, and Minsuk Chang. 2022. TaleBrush: Sketching Stories With Generative Pretrained Language Models. In *CHI Conference on Human Factors in Computing Systems*. 1–19. https://doi.org/10.1145/3491102.3501819

[12] Runmin Cong, Jianjun Lei, Huazhu Fu, Ming-Ming Cheng, Weisi Lin, and Qingming Huang. 2018. Review of Visual Saliency Detection With Comprehensive Information. *IEEE Transactions on Circuits and Systems for Video Technology* 29, 10 (2018), 2941–2959. https://doi.org/10.1109/TCSVT.2018.2870832

[13] Alexandre Defossez, Gabriel Synnaeve, and Yossi Adi. 2020. Real Time Speech Enhancement in the Waveform Domain. *ArXiv Preprint ArXiv:2006.12847* (2020). https://doi.org/10.48550/arXiv.2006.12847

[14] Carlos Gonzalez Diaz, Phoenix Perry, and Rebecca Fiebrink. 2019. Interactive Machine Learning for More Expressive Game Interactions. In *2019 IEEE Conference on Games (CoG)*. IEEE, IEEE, 1–2. https://doi.org/10.1109/CIG.2019.8848007

[15] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, Shahram Izadi, Adarsh Kowdle, Konstantine Tsotsos, and David Kim. 2020. DepthLab: Real-Time 3D Interaction With Depth Maps for Mobile Augmented Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 829–843. https://doi.org/10.1145/3379337.3415881

[16] Ruofei Du, Yinda Zhang, Ahmed Sabie, and Jason Mayes. 2022. Portrait Depth API: Turning a Single Image into a 3D Photo with TensorFlow.js. https://blog.tensorflow.org/2022/05/portrait-depth-api-turning-single-image.html

[17] Rebecca Fiebrink, Perry R Cook, and Dan Trueman. 2011. Human Model Evaluation in Interactive Supervised Learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 147–156. https://doi.org/10.1145/1978942.1978965

[18] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. 2019. Understanding Deep Networks Via Extremal Perturbations and Smooth Masks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2950–2958. https://doi.org/10.1109/ICCV.2019.00304

[19] Krishna Gade, Sahin Cem Geyik, Krishnaram Kenthapadi, Varun Mithal, and Ankur Taly. 2019. Explainable AI in Industry. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 3203–3204. https://doi.org/10.1145/3292500.3332281

[20] Leo Gao, John Schulman, and Jacob Hilton. 2022. Scaling Laws for Reward Model Overoptimization. *arXiv preprint arXiv:2210.10760* (2022).

[21] Michael Gleicher, Aditya Barve, Xinyi Yu, and Florian Heimerl. 2020. Boxer: Interactive Comparison of Classifier Results. *Computer Graphics Forum* (Jun. 2020). https://doi.org/10.1111/cgf.13972

[22] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. 2019. Digging Into Self-Supervised Monocular Depth Estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3828–3838. https://doi.org/10.1109/ICCV.2019.00393

[23] Google. 2022. Colab. https://colab.sandbox.google.com/

[24] Google. 2022. Firebase. https://firebase.google.com

[25] Google. 2022. TensorBoard. https://www.tensorflow.org/tensorboard

[26] 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*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[27] Wenbin He, Lincan Zou, Arvind Kumar Shekar, Liang Gou, and Liu Ren. 2021. Where Can We Help? A Visual Analytics Approach to Diagnosing and Improving Semantic Segmentation of Movable Objects. *IEEE Transactions on Visualization and Computer Graphics* 28, 1 (2021), 1040–1050. https://doi.org/10.1109/TVCG.2021.3114855

[28] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. 2018. Visual Analytics in Deep Learning: an Interrogative Survey for the Next Frontiers. *IEEE Transactions on Visualization and Computer Graphics* 25, 8 (2018), 2674–2693. https://doi.org/10.1109/TVCG.2018.2843369

[29] Andrew Howard and Suyog Gupta. 2020. Background Features in Google Meet, Powered by Web ML. https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html

[30] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1314–1324. https://doi.org/10.1007/978-1-4842-6168-_11

[31] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv Preprint ArXiv:1704.04861* (2017). https://arxiv.org/pdf/1704.04861.

[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708. https://doi.org/10.1109/CVPR.2017.243

[33] Varun Jampani, Huiwen Chang, Kyle Sargent, Abhishek Kar, Richard Tucker, Michael Krainin, Dominik Kaeser, William T Freeman, David Salesin, Brian Curless, et al. 2021. SLIDE: Single Image 3D Photography With Soft Layering and Depth-Aware Inpainting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 12518–12527. https://doi.org/10.1109/ICCV48922.2021.01229

[34] Katarzyna Janocha and Wojciech Marian Czarnecki. 2017. On Loss Functions for Deep Neural Networks in Classification. https://doi.org/10.48550/ARXIV.1702.05659

[35] Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie Cai. 2022. PromptMaker: Prompt-Based Prototyping With Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. ACM. https://doi.org/10.1145/3491101.3503564

[36] David Johnson, Giuseppe Carenini, and Gabriel Murray. 2020. NJM-Vis: Interpreting Neural Joint Models in NLP. In *Proceedings of the 25th International Conference on Intelligent User Interfaces* (Cagliari, Italy) *(IUI '20)*. Association for Computing Machinery, New York, NY, USA, 286–296. https://doi.org/10.1145/3377325.3377513

[37] Gábor Karsai. 1995. A Configurable Visual Programming Environment: a Tool for Domain-Specific Programming. *Computer* 28, 3 (1995), 36–44. https://doi.org/10.1109/2.366147

[38] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. 2019. Real-Time Facial Surface Geometry From Monocular Video on Mobile GPUs. *ArXiv Preprint ArXiv:1907.06724* (2019). https://arxiv.org/pdf/1907.06724

[39] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 6. Jupyter Notebooks - a Publishing Format for Reproducible Computational Workflows.

[40] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444. https://doi.org/10.1109/ACCESS.2019.2912200

[41] Na Li, Jason Mayes, and Ping Yu. 2021. *ML Tools for the Web: a Way for Rapid Prototyping and HCI Research*. 315–343. https://doi.org/10.1007/978-3-030-82681-_10

[42] Rensis Likert. 1932. A Technique for the Measurement of Attitudes. *Archives of Psychology* (1932).

[43] Shanchuan Lin, Andrey Ryabtsev, Soumyadip Sengupta, Brian L Curless, Steven M Seitz, and Ira Kemelmacher-Shlizerman. 2021. Real-Time High-Resolution Background Matting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and*

Pattern Recognition. 8762–8771. https://doi.org/10.1109/CVPR46437.2021.00865

[44] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. 2019. MediaPipe: a Framework for Building Perception Pipelines. https://doi.org/10.48550/arXiv.1906.08172

[45] MediaPipe. 2022. MediaPipe Selfie Segmentation. https://tfhub.dev/mediapipe/tfjs-model/selfie_segmentation/general/1

[46] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

[47] Sugeerth Murugesan, Sana Malik, Fan Du, Eunyee Koh, and Tuan Manh Lai. 2019. DeepCompare: Visual and Interactive Comparison of Deep Learning Model Performance. (Sep. 2019). https://doi.org/10.1109/MCG.2019.2919033

[48] Ahmed Omran, Neil Zeghidour, Zalán Borsos, Félix de Chaumont Quitry, Malcolm Slaney, and Marco Tagliasacchi. 2022. Disentangling Speech From Surroundings in a Neural Audio Codec. *ArXiv Preprint ArXiv:2203.15578* (2022). https://doi.org/10.48550/arXiv.2203.15578

[49] Rohit Pandey, Sergio Escolano, Chloe Legendre, Christian HÃ¤ne, Sofien Bouaziz, Christoph Rhemann, Paul Debevec, and Sean Fanello. 2021. Total Relighting. *ACM Transactions on Graphics* (Aug. 2021). https://doi.org/10.1145/3450626.3459872

[50] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image Transformer. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 4055–4064. https://doi.org/10.1109/CVPR46437.2021.01212

[51] Donghao Ren, Bongshin Lee, and Tobias Höllerer. 2017. Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering. *Computer Graphics Forum* (June 2017), 61–70. https://doi.org/10.1111/cgf.13178

[52] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection With Region Proposal Networks. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://doi.org/10.5555/2969239.2969250

[53] Alex Repenning. 1993. Agentsheets: A Tool for Building Domain-Oriented Visual Programming Environments. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (Amsterdam, The Netherlands) *(CHI '93)*. Association for Computing Machinery, New York, NY, USA, 142–143. https://doi.org/10.1145/169059.169119

[54] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. https://doi.org/10.1145/2939672.2939778

[55] Lars Ruthotto and Eldad Haber. [n.d.]. An Introduction to Deep Generative Modeling. *GAMM-Mitteilungen* 44, 2 ([n. d.]), e202100008. https://doi.org/10.1002/gamm.202100008

[56] Eldon Schoop, Forrest Huang, and Bjoern Hartmann. 2021. Umlaut: Debugging Deep Learning Programs Using Program Structure and Model Behavior. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16. https://doi.org/10.1145/3411764.3445538

[57] Scratch. 2022. Scratch. https://scratch.mit.edu/

[58] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M. Summers. 2016. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging* 35, 5 (2016), 1285–1298. https://doi.org/10.1109/TMI.2016.2528162

[59] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, Stan Bileschi, Michael Terry, Charles Nicholson, Sandeep N. Gupta, Sarah Sirajuddin, D. Sculley, Rajat Monga, Greg Corrado, Fernanda B. Viégas, and Martin Wattenberg. 2019. TensorFlow.js: Machine Learning for the Web and Beyond. https://doi.org/10.48550/arXiv.1901.05350

[60] Snap. 2022. Lens Studio. https://ar.snap.com/en-US/lens-studio

[61] Samuel Sonning, Christian Schüldt, Hakan Erdogan, and Scott Wisdom. 2020. Performance Study of a Convolutional Time-Domain Audio Separation Network for Real-Time Speech Denoising. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 831–835. https://doi.org/10.1109/ICASSP40776.2020.9053846

[62] Thilo Spinner, Udo Schlegel, Hanna Schafer, and Mennatallah El-Assady. 2019. ExplAIner: a Visual Analytics Framework for Interactive and Explainable Machine Learning. *IEEE Transactions on Visualization and Computer Graphics* (2019). https://doi.org/10.1109/TVCG.2019.2934629

[63] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to Summarize with Human Feedback. *Advances in Neural Information Processing Systems* 33 (2020), 3008–3021.

[64] WR Sutherland. 1966. On-Line Graphical Specification of Procedures. *SJCC, Boston, Mass* (1966).

[65] Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S. Tan. 2009. EnsembleMatrix: Interactive Visualization to Support Machine Learning With Multiple Classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1283–1292. https://doi.org/10.1145/1518701.1518895

[66] Feitong Tan, Sean Fanello, Abhimitra Meka, Sergio Orts-Escolano, Danhang Tang, Rohit Pandey, Jonathan Taylor, Ping Tan, and Yinda Zhang. 2022. VoLux-GAN: a Generative Model for 3D Face Synthesis With HDRI Relighting. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*. ACM. https://doi.org/10.1145/3528233.3530751

[67] 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. https://doi.org/10.48550/ARXIV.2008.05122

[68] TensorFlow. 2022. AR Portrait Depth API. https://tfhub.dev/tensorflow/tfjs-model/ar_portrait_depth

[69] three.js authors. 2022. Three.js. https://threejs.org

[70] TikTok. 2022. Effect House. https://effecthouse.tiktok.com/

[71] Unity. 2022. Shade Graph. https://unity.com/features/shader-graph

[72] Unity. 2022. Unity. https://unity.com/products/unity-platform

[73] Unity. 2022. XNode. https://assetstore.unity.com/packages/tools/visual-scripting/xnode-104276

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *Advances in Neural Information Processing Systems* 30 (2017). https://doi.org/10.5555/3295222.3295349

[75] Xintao Wang, Yu Li, Honglun Zhang, and Ying Shan. 2021. Towards Real-World Blind Face Restoration With Generative Facial Prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9168–9178. https://doi.org/10.1109/CVPR46437.2021.00905

[76] Zhihao Wang, Jian Chen, and Steven CH Hoi. 2020. Deep Learning for Image Super-Resolution: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 10 (2020), 3365–3387. https://doi.org/10.1145/3485132

[77] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viégas, and Jimbo Wilson. 2019. The What-If Tool: Interactive Probing of Machine Learning Models. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (2019), 56–65. https://doi.org/10.1109/TVCG.2019.2934619

[78] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's Transformers: State-of-the-Art Natural Language Processing. *ArXiv Preprint ArXiv:1910.03771* (2019). https://arxiv.org/pdf/1910.03771

[79] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie Cai. 2022. PromptChainer: Chaining Large Language Model Prompts Through Visual Programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. ACM. https://doi.org/10.1145/3491101.3519729

[80] Tongshuang Wu, Michael Terry, and Carrie Cai. 2022. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. In *CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3491102.3517582

[81] Ning Xu, Brian Price, Scott Cohen, and Thomas Huang. 2017. Deep Image Matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2970–2979. https://doi.org/10.1109/CVPR.2017.41

[82] Xiwei Xuan, Xiaoyu Zhang, Oh-Hyun Kwon, and Kwan-Liu Ma. 2022. VAC-CNN: a Visual Analytics System for Comparative Studies of Deep Convolutional Neural Networks. *IEEE Transactions on Visualization and Computer Graphics* 28, 6 (jun 2022), 2326–2337. https://doi.org/10.1109/TVCG.2022.3165347

[83] Jun Yuan, Changjian Chen, Weikai Yang, Mengchen Liu, Jiazhi Xia, and Shixia Liu. 2021. A Survey of Visual Analytics Techniques for Machine Learning. *Computational Visual Media* 7, 1 (2021), 3–36. https://doi.org/10.1109/ACCESS.2019.2958551

[84] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. 2020. Mediapipe Hands: On-Device Real-Time Hand Tracking. *ArXiv Preprint ArXiv:2006.10214* (2020). https://arxiv.org/pdf/2006.10214

[85] Yinda Zhang, Neal Wadhwa, Sergio Orts-Escolano, Christian Häne, Sean Fanello, and Rahul Garg. 2020. Du^{2}Net: Learning Depth Estimation From Dual-Cameras and Dual-Pixels. In *European Conference on Computer Vision*. Springer, Springer, 582–598. https://doi.org/10.1007/978-3-030-58452-8_34

APPENDIX

We elaborate on the survey of visual analytics systems, supported nodes in Rapsai, more example applications, case study protocols, and exit surveys.

A SURVEY OF VISUAL ANALYTICS SYSTEMS

In addition to instance-based visualization reviewed in related work, for completeness, we add aggregated visualization literature in this appendix. There are different techniques to visualize aggregated metrics. For classification models, researchers often use the confusion matrix. Talbot *et al.* [65] presented an interactive visualization system called EnsembleMatrix for exploring the space of ensemble models and demonstrated that the metric visualization helped users with various ML experiences be able to combine multiple classifiers in order to build an improved model. For regression models, researchers typically use L1 or L2 norm loss [34]. Popular training tools such as TensorBoard [25] visualize loss plot by default in its result page. We list the visual analytics systems in Table 1, which summarizes the main dimensions we compared against. It clearly shows that Rapsai addresses all multimedia formats, supports generative models, which few systems support. In addition, Rapsai provides interactive data augmentation and no-code pipeline authoring, which is unique for visual ML analytics systems.

B EXTENDED EXAMPLES OF SUPPORTED NODES IN RAPSAI

Data Augmentation in the Image Processor Node

In Fig. 14, we present more examples of interactive data augmentations using the **Image Processor Node** of the **Effect Nodes** category. In the Image Processor Node, the **crop** tab allows users to select a region of the image or video as output, move the selection, and instantly see the impact on model performance; the **resize** tab allows users to change the aspect ratio of the image and verify if the model is adaptive to various resolutions; the **shear** and **rotate** tabs allow users to displace the image by an amount proportional to the y coordinates and rotate the image; the **brightness, contrast, blur, noise** tabs allow users to change the brightness levels, contrast levels, blurriness, and noisiness of the image with GPU acceleration.

Users can further connect the intermediate processed images with ML models, then output the results in **Image Comparison** nodes to obtain side-by-side comparison (Fig. 15) without coding efforts.

B.1 Data Augmentation in the Audio Processor node

In the **Audio Processor Node** of the **Effect Nodes** category, we provided researchers a variety of background noise, including *park, bus, cafe, construction, downtown, group meeting, party, crowd, street, rain, restaurant, shopping mall, supermarket,* and *train*. Users can change the volume of the background noise and see how their model performs under different noise levels.

B.2 More Examples of the Shader Effects Node

In the **Shader Library** node, we provide a list of pre-written shader effects for virtual conferences including reflections, sketching, pixelation, and 3D environments, as shown in the supplementary video and Fig. 16. Though Rapsai offers a no-coding environment for most ML practitioners, it also provides advanced users with **Shader Effects Nodes** to write their own fragment shader for visual effects in GLSL. This enables custom depth-based effects such as relighting and fog effects, shown in the Fig. 9, and also enables rapidly tuning graphics parameters such as kernel sizes of Gaussian blur for virtual background with foreground segmentation. We further present five example shader effects in Fig. 16, including television effects with distortion, halftone effects, tone-transfer effects, vignette effects, and a 3D effect by projecting the input image onto a virtual screen.

B.3 Lobby Node for Streaming Cross-device Videos

The **Lobby Node** allows users to stream up to four camera videos from other devices (*e.g.*, a mobile phone) as input. User can join an URL provided by the Lobby Node with a specific Group ID to start streaming. The Lobby Node can access video streams from all the joined clients and output stream through an ML or a graphics pipeline on a more powerful device (*e.g.*, a laptop) via WebRTC. This is useful when an ML model is trained for on-device use cases such as a mobile phone and ML practitioners need to perform testing or user studies with on-device in-the-wild input.

C ADDITIONAL APPLICATIONS

In addition to the four case studies presented in the main paper, we discuss additional applications in this section.

C.1 Bridging Large Language Models and Computer Graphics Applications

Since our case study, we have further extended Rapsai's node library with text input and large language models such as ChatGPT. In Fig. 18, we prompt ChatGPT [20, 63], a large language model by OpenAI with "write a shadertoy shader to show edges in an image". Then the second output from the ChatGPT node extracts the code snippets from the responses using regular expression matching. The fragment shader code from ChatGPT is applied to the input camera node and outputs a video stream with highlighted edges. Please refer to the supplementary video for more results. Please note that this is only a preliminary exploration into bridging language prompts and graphical applications and we hope future work would further extend this direction.

C.2 Augmented Reality Stickers with Real-time Facial Landmarks

With Mediapipe FaceMesh models[9] [38], ML prototypers can use Rapsai to quickly build virtual try-on applications with real-time facial landmarks (Fig. 19).

C.3 Compare Image Classification Results

Though Rapsai is not designed for image classification tasks, we also support comparing image classification results with simple

[9] MediaPipe FaceMesh: https://google.github.io/mediapipe/solutions/face_mesh.html

system	aggregated visualization	instance-based visualization	domain	supported model types	non-developer users	model comparison	interactive data augmentation	pipeline authoring
EnsembleMatrix [65]	✓	-	image	classification	✓	-	-	-
TensorBoard [25]	✓	-	all	all	-	-	-	-
LIME [54]	-	✓	image	classification	✓	-	-	-
What-If [77]	✓	-	image	classification	✓	✓	-	-
Boxer [21]	✓	-	image	classification	-	-	-	-
LIT [67]	✓	✓	language	classification	✓	-	-	-
DeepCompare [47]	✓	-	language	classification	✓	✓	-	-
NJM-Vis [36]	✓	✓	language	classification	✓	✓	-	-
VAC-CNN [82]	✓	✓	image	classification	✓	✓	-	-
ExplAIner [62]	✓	✓	image	classification	-	-	-	-
VASS [27]	-	✓	image	generative	-	✓	-	-
Rapsai	-	✓	image, **audio**, **video**	**generative**, classification	✓	✓	✓	✓

Table 1: Overview of target domains and key features of Rapsai and prior visual analytics systems for machine learning.

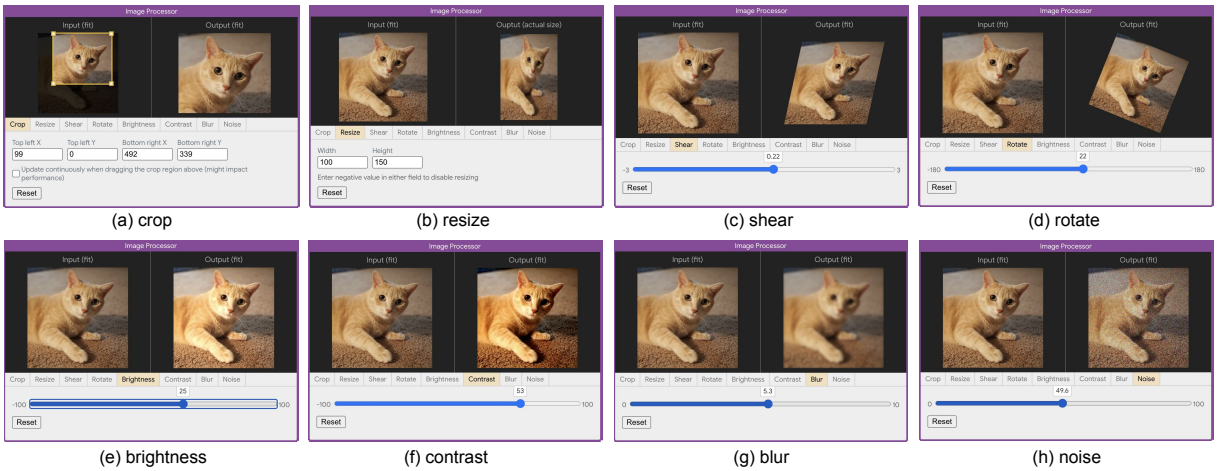


Figure 14: Supported visual data augmentation operations in our Image Processor node: (a) crop, (b) resize, (c) shear, (d) rotate, (e) brightness, (f) contrast, (g) blur, (h) noise.

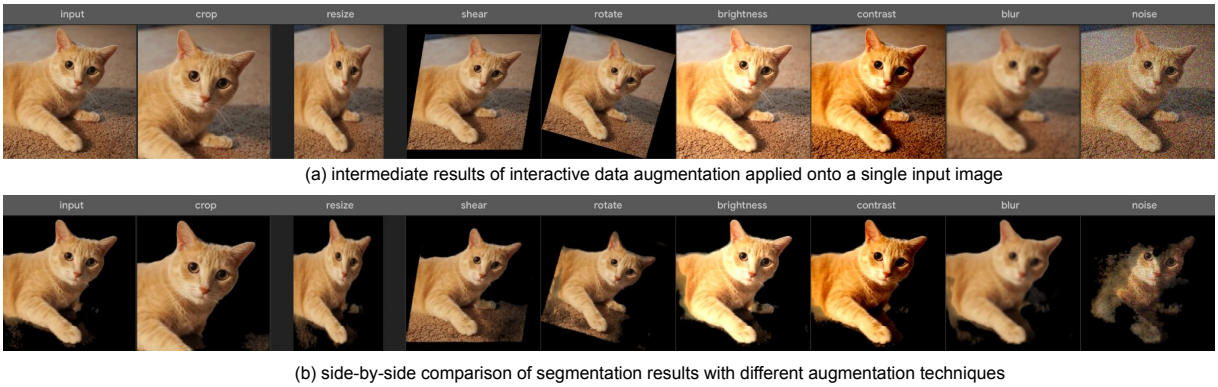


Figure 15: Rapsai enables ML practitioners to quickly generate side-by-side comparison of a wide range of data augmentation techniques with an image in-the-wild.

visualizations, such as bar charts, as shown in Fig. 20. Many participants in our case studies also play with this pipeline and expressed their interest in having aggregated metrics together with this side-by-side comparison. With Rapsai, our major focus is to deliver a rapid prototyping tool for multimedia applications and we retain this feature to be further implemented by the community after releasing the code. Our study participants proposed an interesting direction of combining aggregated metrics. For instance, P9 suggested: *"I'd like to have image classification models with smoothing*

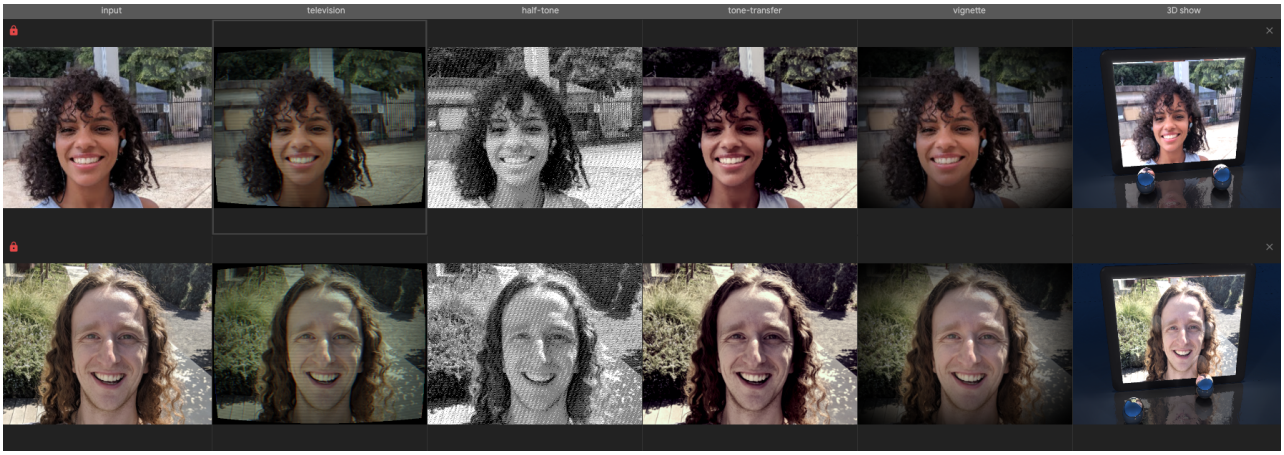


Figure 16: Shader Effects nodes allows users to choose a pre-defined shader from a library of over 50 visual effects or write their own fragment shader using GLSL.

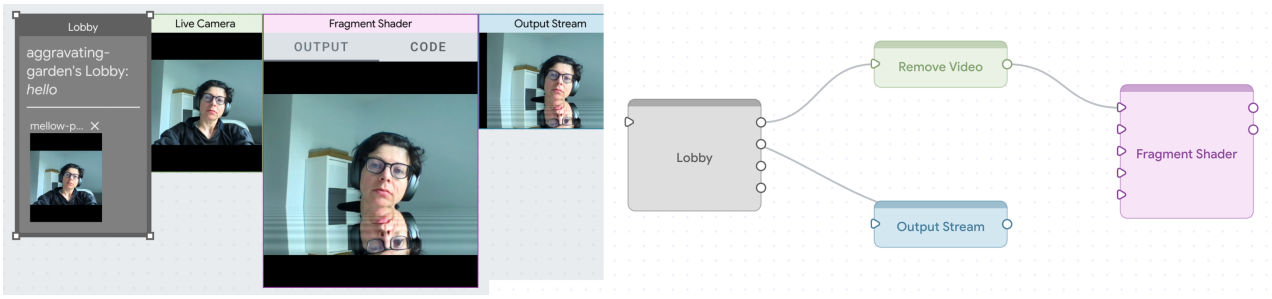
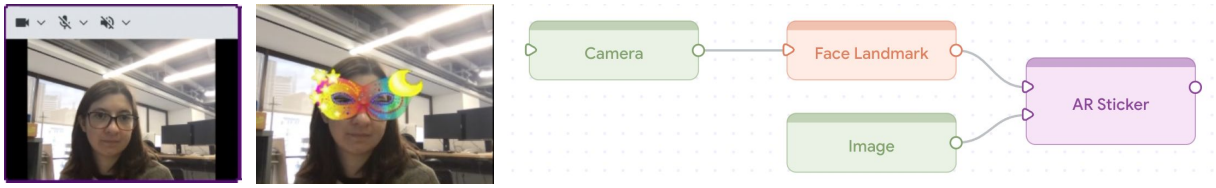


Figure 17: Lobby nodes allows users to stream camera videos from another device (*e.g.*, mobile phone) and run the pipeline on a another device (*e.g.*, a laptop).

(a) Rapsai pipeline with LLMs (b) input text node and ChatGPT node (c) processed video with edge highlighting

Figure 18: Rapsai can automatically extract code snippets from ChatGPT results and apply real-time visual effects to an input video stream. (a) shows the Rapsai pipeline in the node-graph editor, (b) shows our input prompts and the responses from ChatGPT, and (c) shows the input video frame and the resulting output.



(a) camera input (b) AR sticker effect (c) example AR sticker pipeline

Figure 19: ML prototypers can use Rapsai to quickly build AR sticker applications using real-time tracking of facial landmarks.

filters, and see how it averages the probabilities of a class.". At the current stage, Rapsai can act as a complement to existing tools such

as Colab, and speed up the prototype and collaboration process in image classification.

C.4 Photo Enhancement with GFPGAN

Using a pretrained GFPGAN model [75], which can be used to restore old photos or improve AI-generated faces, one of the authors constructed a photo-enhancement pipeline in less than ten minutes using Rapsai, as illustrated in Fig. 21. The pipeline is built with only seven nodes in Rapsai: input node, image processor, preprocessing node, customized model node, postprocessing node, tensor to image node, and image comparison node.

D FULL LIST OF NODES IN RAPSAI

We list all the supported nodes in Rapsai before the case study. Note that Rapsai is extendable for expert users by adding new nodes in JavaScript.

D.1 Input nodes

(1) **image**: capture a photo from webcam, upload from hard drive, or fetch from a list of remote URLs.
(2) **video**: record a video with external webcam or upload a video from disk or YouTube.
(3) **audio**: record sounds from microphone, or upload audio files from disk or Internet
(4) **live camera**: use live camera stream, similar for the live audio node.
(5) **remote stream**: stream input from another device (*e.g.*, mobile phone) via WebRTC, by opening a URL of the page with a lobby node.
(6) **lobby**: create a WebRTC server to accept video streaming from other devices. Remote stream nodes that connect to the lobby node with the same name are sharing output via WebRTC streaming.

D.2 Effect nodes

(7) **image processor**: crop and translate a region of interest in the input to verify an image model's invariance to translation; rotate, shear, resize an image to examine potential biasing issues in the training sets; apply blur and noise to test a model's robustness.
(8) **image mixer**: mix images with GPU-based blending modes[10]
(9) **audio processor**: trim the audio, change volume, and add background noise from a collection of 17 presets.
(10) **fragment shader**: program and apply a screen-space graphical shader effect.
(11) **shader library**: offers a pre-defined list of shader code to avoid coding into the "fragment shader".

D.3 Model nodes

(12) **custom model runner**: enter the URL of an TensorFlow.js model or upload a TensorFlow model into the pipeline.
(13) **body segmentation**: run a deployed MediaPipe body segmentation model.
(14) **audio denoising**: run a deployed MediaPipe body segmentation model.
(15) **MobileNet**: run a deployed MobileNet model for image classification.

[10] https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/globalCompositeOperation

D.4 Output nodes

(16) **image viewer** node displays an image.
(17) **audio player**: play a single audio output.
(18) **image comparison**: qualitatively compare output from multiple models with zoom-in tools.
(19) **audio comparison**: qualitatively compare output from multiple models with automatic track switching.
(20) **JSON viewer**: read the raw output from a model for debugging.
(21) **bar viewer**: view the classification results from a model such as MobileNet.
(22) **3D model viewer**: view 3D models from an URL or tensor.
(23) **tensor to image**: view a tensor as images.
(24) **tensor to depthmap**: view a tensor as depthmaps with different transfer functions.
(25) **output stream**: see remote video streams from the **lobby node** input.

D.5 Tensor nodes

(26) **preprocess image**: converts an input image to a 4D tensor as an input that is required by most image models.
(27) **tensor picker**: select a tensor from an array of output tensors.
(28) **tensor postprocess**: convert a tensor to an image and apply normalization calculators.
(29) **binary op**: apply "and", "or", "xor", and "not" operations between two input tensors.
(30) **clip by value**: clamp the values of an input tensor.
(31) **crop and resize**: crop and resize a two-dimensional tensor.
(32) **preprocess tensor**: normalize a tensor, expand dimensions, and optionally convert to grayscale image for many generative models.
(33) **postprocess tensor**: normalize and resize a tensor for image output.
(34) **tensor picker**: select a certain tensor from an array of model output. Note that most models only have one output so it is optional.
(35) **remap value range**: select an input and an output ranges to remap tensor values.

D.6 Miscellaneous nodes

(36) **webpage**: append a Google Form or a custom webpage for filling in surveys.
(37) **image size**: obtain image size for outputting to some models.

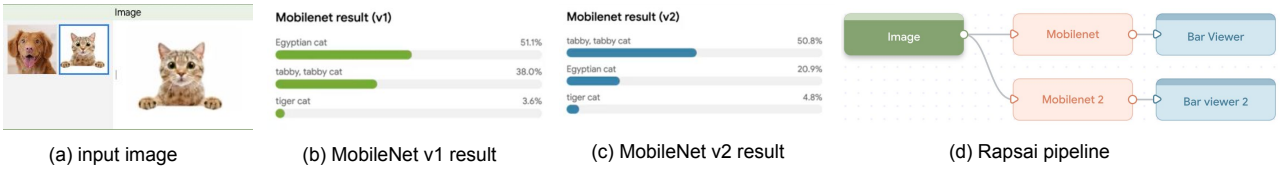
E STUDY PROTOCOLS

E.1 Formative Study Protocol

[Introduction] (Start timing! 60 min max.)

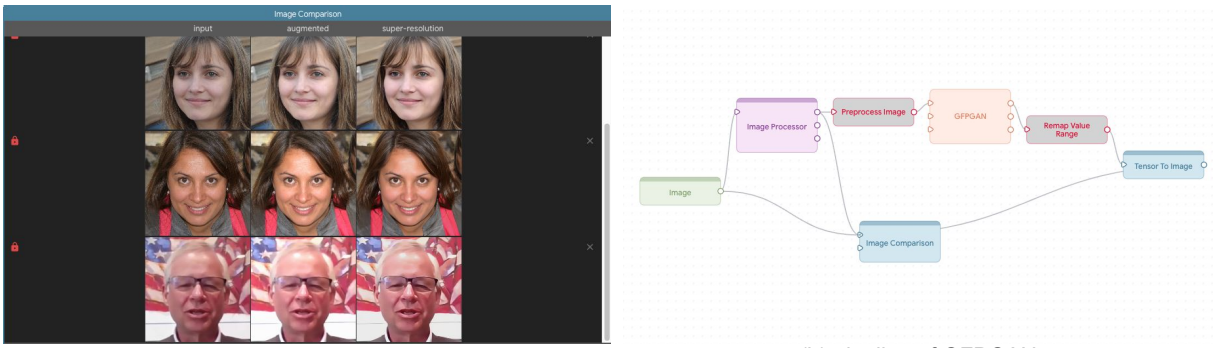
Hello, my name is X.

First, I would like to thank you for your participation. Today, you will be a participant in a formative study regarding machine



(a) input image (b) MobileNet v1 result (c) MobileNet v2 result (d) Rapsai pipeline

Figure 20: Rapsai also supports qualitatively comparing image classification models such as MobileNet at the instance level.



(a) results comparison of GFPGAN (b) pipeline of GFPGAN

Figure 21: Rapsai also supports rapidly building augmented reality effects with publicly available models such as GFPGAN.

learning and rapid prototyping. Our goal is to identify the pain points when identifying key issues of machine learning models, when comparing performance of different models, and when iterating on models development between training and deploying in products.

Before we begin the interview, we need you to complete a consent form. After this, we will begin.

Your data will be kept anonymous. Additionally, as a researcher I have no position on this topic and ask that you be as open, honest, and detailed in your answers as possible. Do you have any questions before we begin?

The interview is broken down into three components: 1) Your background in deep learning. 2) Your challenges of the current workflow and TensorBoard. 3) Survey about our envisioned web-based rapid prototyping platform.

[Background Questions] (5 - 10 min)

(1) What kinds of neural networks projects have you worked on?
(2) What kinds of challenges have you faced when tuning deep learning models?
(3) What kinds of online resources have you found useful when tuning deep learning models?
(4) How long have you spent on examining individual examples when tuning neural networks?

[Process Questions] (15 - 20 min)

(1) Would you like to describe your current ML debugging procedure of model quality to me?
(2) What do you like or dislike about the tool you are currently using (e.g., TensorBoard)?
(3) What information you wished to know about a neural network that is not visualized in the current tool?
(4) Have you ever used MediaPipe Editor? What do you like or dislike about MediaPipe Editor?
(5) What else editor would you like to use for debugging model quality?
(6) How do you make high-level design decisions of a neural network for now?
(7) How do you usually test the robustness of a neural network?
(8) What methods do you use for data augmentation?
(9) How do you infer the generalizability of a neural network?

[Envisioned Rapsai Questions] (20 - 30 min)

(1) What's your first impression of the interface?
(2) What visualization technique would like to have in this interface?
(3) What features would you like to add to the envisioned rapid prototyping system?
(4) What suggestions would you like to give to improve the design of the virtual interfaces of a rapid prototyping platform?
(5) What outcomes would you like to get from this pipeline editor? For example, a executable demo, a pipeline graph, or something else.
(6) With a comparison node, what information you might want to gather from it?
(7) What criteria would you want to use when comparing two deep learning models?
(8) What information would you like to have to increase confidence in the value of the results?

E.2 Case Study Protocol

[Introduction]

Same as Section E.1.

[Background Questions]

Same as Section E.1.

[Build a Node Graph Pipeline]

(1) To get started, let's watch a 1-min tutorial to have an overview of the Rapsai system.
(2) Next, we would like to instruct you to build <a specific pipeline> to use <specific> ML models.
(3) What's your impression of the node-graph building procedure?
(4) What's good and what may be improved?

[Interactive Data Augmentation]

(1) Next, we would like to instruct you to perform interactive data augmentation to your pipeline.
(2) What can you learn from the interactive data augmentation procedure?
(3) Have you ever applied similar data augmentation in your ML workflow?
(4) Do you think it may be useful to your ML workflow?
(5) What's your impression of the interactive data augmentation interface?
(6) What's good and what may be improved?

[Compare the Models]

(1) Next, we would like to instruct you to use the comparison node and data augmentation nodes to compare two or more models.
(2) Which model do you think is preferred to be used in production and why?
(3) What's your impression of the interactive data augmentation interface?
(4) Do you think it may be useful in your ML workflow?
(5) What can you learn from the interactive data augmentation procedure?
(6) What's good and what may be improved?

[Discussion]

< Ask the participants to explore freely using Rapsai >

(1) What features would you like to add to Rapsai?
(2) Do you have any suggestions to improve the design of Rapsai?
(3) What product questions would you like to answer using Rapsai?

That's all for our case study. Thank you for your participation and we would like to hear more feedback from your experience in your future workflow!

E.3 Exit Survey

Inspired by Wu *et al.* [80], we invited participants to fill in an exit survey after the case studies. We included screenshots and URLs for both Rapsai and Colab to assist the participants in completing the survey. The participants self-rated their experience using Rapsai and Colab on five aspects in the form of seven-point Likert scale [42]. Each question was presented twice, once on Colab and once on Rapsai, randomly in a counter-balanced way. The participants explained their reasoning alongside their ratings and also answered three free-form questions as follows:

Likert-scale questions:

(1) **Match goal**: I'm satisfied with my final results from [Rapsai/Colab]; they met my task goals (*e.g.*, comparing two model performances and testing model robustness).
(2) **Think through**: The [Rapsai/Colab] system helped me think through what kinds of outputs I would want to complete the task goal and how to complete the task.
(3) **Transparent**: The [Rapsai/Colab] system is transparent about how it arrives at its final results; I could roughly track its progress.
(4) **Controllable**: I felt I had control creating with the [Rapsai/Colab] system. I can steer the system towards the task goal.
(5) **Collaborative**: In [Rapsai/Colab], I felt I was collaborating with the system to come up with the outputs.

Free-form questions:

(1) **Timing**: How long do you estimate it takes you to create an ML application pipeline using [Rapsai/Colab], and why?
(2) **Difference**: What were the differences, if any, between the experience of completing the task using Rapsai and Colab? How about comparing to other debugging / visualization systems if you have been using any in the past?
(3) **Vision**: If you were using machine learning models in your work, in what situations would you prefer to use Rapsai/Colab? Can you think of 1-3 concrete examples?

Please note the following limitations of the exit survey:

(1) The survey could have been conducted more effectively by comparing Rapsai to a sandbox system [80], whose UI resembles Rapsai's but lacks the node-graph editor feature. Due to time constraints, we were unable to create a Sandbox system that met our requirements without the node-graph editor for another round of case studies. We expect that a future long-term case study will capture the merits and drawbacks of Rapsai in greater detail.
(2) The modest sample size (N=15) of our survey within the same organization may not be representative of the AI+HCI community as a whole.
(3) In our case study, participants were obliged to use Rapsai to complete the tasks, since not all tasks are easily accomplished in Colab.