Operating a Complex SLT System with Speakers and Human Interpreters

Ondřej Bojar, Vojtěch Srdečný, Rishu Kumar, Otakar Smrž
[surname]@ufal.mff.cuni.cz Charles University, MFF, ÚFAL

Felix Schneider
felix.schneider@kit.edu Karlsruhe Institute of Technology, Germany

Barry Haddow, Phil Williams
bhaddow@ed.ac.uk, pwillia4@inf.ed.ac.uk University of Edinburgh

Chiara Canton
chiara.canton@pervoice.it PerVoice

Abstract

We describe our experience with providing automatic simultaneous spoken language translation for an event with human interpreters. We provide a detailed overview of the systems we use, focusing on their interconnection and the issues it brings. We present our tools to monitor the pipeline and a web application to present the results of our SLT pipeline to the end users. Finally, we discuss various challenges we encountered, their possible solutions and we suggest improvements for future deployments.

1 Introduction

In April 2021, a European international organisation hosted an international congress for its members. While the event was originally planned to be in-person, the COVID-19 pandemic meant that all the foreign participants connected remotely. The event was run in 5 languages (English, German, French, Spanish and Russian) covered by human interpreters. The remote audience spanned 51 countries with the total of 42 desired languages. Our role was to provide live translation into these additional languages in text form.

The technical backstage of the event operated in the standard in-person mode, with interpreters in their booths, following the main live video stream and providing or relaying interpretation as needed. This resulted in six audio channels being available, one for each language and one additional channel called "the floor" which always contained the speech of the current speaker regardless the language. The interpreters delivered their interpretation to the appropriate language-labelled channels. Each of the interpreters was translating either from English to their assigned language, or vice versa. At any given moment, there was thus supposed to be exactly one source of English speech, either directly from the speaker or from one of the interpreters. The channel of the language spoken at the floor at a given point was silent (e.g. the German channel when German was coming from the floor, because the German interpreters' booth was busy providing the English interpretation to the English channel).

While this arrangement caused some technical challenges, it also provided novel opportunities. The first challenge was to concurrently follow all the channels using portable equipment

and the second challenge was to direct the correct audio channel to the respective speech processing server. The multiple input languages can make the setup interestingly robust: For example, when the speaker is speaking a non-English language, their speech can be automatically transcribed and then machine-translated to English. At the same time, an interpreter is providing human interpretation into English, which can be in turn processed by the English speech recognition system. We thus have two sources of English text and we can choose the better one, live, bypassing any processing hurdles at any of the paths. Conversely, when the speaker is speaking English, the interpreters will be providing their assigned languages, perhaps captured in better sound conditions or better articulated. These languages can again be automatically transcribed and translated to English, serving as alternative sources if the main speaker is hard to follow for the technology.

The paper is structured as follows: in Section 2, we describe our hardware setup at the backstage. Section 3 provides a complete picture of the processing pipeline from 6 input channels to 42 output languages. Section 4 briefly summarizes the key building blocks, namely the speech recognition (ASR) and machine translation (MT) systems used. These systems were cluster-based, run on premises of the individual research institutes contributing them, connected via the Internet. Section 5 presents our web-based solution for live display of the many translations. Several members of our team were on duty to monitor all the components during the event and only one skilled "system operator" was present in person at the backstage. The operator's experience is provided in Section 6 and some of the monitor tools at his disposal are described in Section 7. Section 8 summarizes the planned improvements of the setup and components and Section 9 concludes the paper.

2 Lightweight Hardware Setup for Multi-Source Speech Processing

Building upon our previous experience with speech processing at live events, we knew that desk space would be a limiting factor and that at most one person would be admitted to take care of the system on site. Aside from providing the translation service for the (remote) participants, we also needed to fully record the session for future analysis. The sound engineers facilitating the event itself did not have any recording equipment suitable for our purposes.

Our final solution consisted of one laptop (Dell Vostro 3583) running Ubuntu 20.04 and two Behringer U-Phoria UMC404HD external USB sound cards, each following up to 4 mono sound channels.

To minimize the risk of losing the recording, the system setup was *primarily* geared towards recording. Throughout the session, two `arecord` tasks were recording raw outputs of each of the external sound cards, producing two 4-channel PCM sound files sampled at 44 kHz.

Any sound processing, be it for monitoring purposes or for the actual speech processing, was based upon these growing files. We avoided touching the software sound devices to prevent any software conflicts during the session.

To monitor the incoming sound of any channel across the two recording devices, we simply followed the most recent additions to the respective file and selected the channel with `ffmpeg`:

```
tail -c0 -f RAW_RECORDING.pcm \
| ffmpeg -y -f s32le -acodec pcm_s32le -r 44100 -ac 4 -i - \
    -map_channel 0.0.DESIRED_CHANNEL ... - 2>/dev/null
```

The added benefit of this file-based access to the live sound was that until the actual session was running, we could easily simulate live session by slowly copying data from a pre-recorded sound to mock "raw recording" file:

```
cat SAVED_4-CHANNEL_RECORDING.pcm \
| pv -L 688K -q | dd obs=16 > SIMULATED_RAW_RECORDING.pcm
```

The `pv` command limits throughput, simulating real-time growth, with the byte rate of 688K determined empirically. The `dd obs=16` ensures that the output file grows in multiples of 16 bytes. When watching the "current" sound with `tail -f`, it is guaranteed that the processing starts aligned to the 4 channels in the file.

Such a simulation proved invaluable esp. immediately before the start of the live event. No tests of the components can ensure that the whole complex ensemble is running and ready for an immediate launch.

3 SLT Pipeline Description

Our "SLT pipeline" consists of ASR and MT systems and various components transforming and transporting data between them. The actual setup – which languages to follow, how to switch among them, which ones to use the source for the final translation – varies across events that we already took part in. Here we focus on the particular setup of the international (remote) congress but our tools allow for a rather flexible configuration of the "wiring".

The inputs to the pipeline are the audio sources: English and the five other languages. English audio is converted to text by an English ASR system, while the other languages are converted to English text by the respective ASR system and a subsequent MT system. There is still some room for system optimization by deploying multilingual ASR systems. For MT, we already make use of multi-linguality (a single system trained to translate from any of a small set of languages into English). To achieve independent processing of each of the input languages, each audio source is processed by a separate ASR and MT system.

The central component of the pipeline is a *selection tool*. Given the multiple variants of inputs (all converted to English text), the operator has the option to dynamically choose which one is currently most suitable for the translation into all the desired target languages, as discussed in Section 6 below.

Then, the chosen English text source is fed to a single one-to-many multi-lingual MT system. In our case, the MT system translates the same input text from English to 41 target languages at once.

Finally, the selected English and all the translations are sent to the web application presenting the outputs to the users (Section 5). It is worth mentioning that depending on the sound channel the user is following, they can observe bigger or smaller delay between the speech and the shown translation. For instance, a German user would most likely follow the German speech, but the automatic German transcript can actually be the result of the German interpreter producing English followed by English ASR and English-to-German MT.

There is a potential for improvement in this setup: for the five other non-English spoken languages, we could present their transcription to the users, instead of displaying the output of the one-to-many MT system. This setup would, however, emburden the system operator even more, because the final outputs of these 5 languages would be running on separate paths with independent risk of a crash, requiring independent monitoring. We thus opted for a more uniform approach which was easier to operate: a single input English translated to all languages at once. If any of the target languages stopped updating, the operator knew that all are affected and vice versa.

See Figure 1 for an overview of the data flow when processing non-English speech.

3.1 Pipeline Technical Details

Individual components of the pipeline, such as the ASR or MT systems are distributed across multiple servers at different sites. This has the primary benefit of "immediate deployment". In other words, research systems (as summarized in Section 4 below) are launched by their authors in the known conditions, so the integration time is limited to implementing a simple

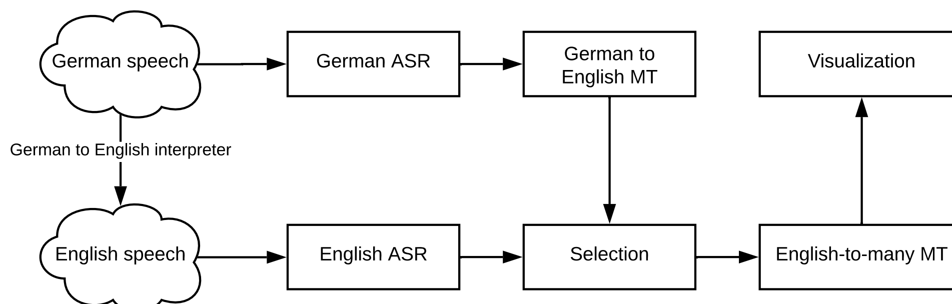

Figure 1: Example of the pipeline when processing German speech.

communication protocol and connecting via TCP connections. Updated models can be included to the pipeline at any point, at their respective author's site and without any involvement of other partners.

Technically, the communication relies on a client-broker service. Individual systems register themselves to the broker, telling the broker what service, e.g. English to German ASR they provide. The broker then publishes a list of available services. Clients then ask the broker for a service and if the service has a provider, the broker facilitates the communication between the client and the service provider. Furthermore, client connectors for audio and text transfer were developed, so the clients can easily send audio and text to the service providers. This setup allows the pipeline operator to pick and choose from various services to integrate into a pipeline, while not having to run all the components locally. A detailed description of the architecture can be found in [Franceschini et al., 2020].

Because the pipelines can grow to be very complex, it was necessary to develop a tool to declaratively describe the pipeline. The pipeline is represented as a directed acyclic graph, with vertices being the individual services, such as English ASR, and edges being the data flow between the services. Each vertex has a set of inputs and outputs, with edges connecting a single output of a vertex to a single input of another vertex. For each particular pipeline, the graph is built in a Python script. The operator describes the vertices and then adds edges between them. Each vertex also contains a command description that starts the component representing the vertex. The command can run fully locally, or it can be the one of the client connectors which passes the task to a remote service offered by the broker. The resulting graph is then compiled to a single bash script which launches all the local commands.

The bash script heavily utilizes network communication on localhost ports to transport the data. The outputs (typically the standard output) of a vertex are captured, replicated and then exposed on different ports. Each output is replicated as many times as there are outgoing edges from that output. Vertices connected to the output then receive one output copy. The splitting is done using standard Unix tools: `tee` that splits the output of a component into multiple subshells, where `netcat` reads it and passes it to another component.

To enable debugging and later reviewing, all standard error outputs of the components are captured and saved to separate files, as well as the edge traffic between individual components. If the captured data are in a plaintext form, each line is also timestamped. This is crucial during analysis of a pipeline failure, as it allows us to deduce what failed and when – if a component fails, the components transitively depending on it usually fail, too.

The local ports are introduced for better process independence. We could in principle directly connect `tee` to the two subsequent components with Unix pipes but in the case of any unexpected exit of any of the components, there would be no way to restart it without restarting the whole pipeline – and also the behaviour of the individual components with respect to the "broken pipe" signal would have to be standardized for predictable behaviour. `netcat` allows us to ensure that the subsequent component's standard input is connected in a stable way and *reopens* the local port upon any failure. This approach is not fully fail-safe but it considerably improves the stability of the whole system.

The compiled bash script can reference many different executables, utilities and files. When multiple people are collaborating on the development of local parts of the pipeline, the deployment can get complicated because everyone has to make sure that various necessary files are being referenced to in a portable way, in addition to the common compilation issues at different systems. Furthermore, these issues are hard to debug, because they are not easily reproducible, unlike e.g. compilation issues.

To alleviate this problem, the bash script is executed inside a Docker container of an image, which already has all of the necessary tools, such as the client connectors or various utilities installed. Another benefit of this approach is a consistent file system structure, allowing to use relative and absolute paths with safety. Similarly, we benefit from the separate network namespace in the Docker container, so ports used for communication between the pipeline components will not collide with ports that the host system might use. While this approach creates some additional technical challenges during development, such as having to rebuild the Docker image when any of the underlying tools is updated, the benefits of having an automatized deployment setup are overall very well worth it.

4 ASR and MT

All ASR systems we used followed the architecture proposed in [Nguyen et al., 2020c] for low-latency online speech recognition. The modeling for all input languages are handled by the streaming sequence-to-sequence model proposed in [Nguyen et al., 2020b] with the use of a multi-domain speech dataset [Nguyen et al., 2020d]. The dynamic transcription mechanism in [Niehues et al., 2016, Nguyen et al., 2020a] is adopted in all ASR systems to achieve very low user-perceived latency.

The MT systems into English are multilingual systems based on the Transformer architecture [Vaswani et al., 2017, Pham et al., 2019]. The system uses a *re-translation* strategy [Niehues et al., 2016] in order to reduce the latency of the MT, as opposed to streaming approaches such as [Ma et al., 2019]. In the re-translation approach, incoming text from ASR is translated afresh starting from the beginning of the sentence, or the end of the stable ASR output, whichever is earlier. Consequently, the new output of the MT system can rewrite, or "flicker" what has already been shown, leading to the question of how best to present this to the user (see Section 5 below). Following [Niehues et al., 2018], we inject partial sentence pairs (prefixes) into the training data so that the system is better able to deal with this at runtime.

Similarly, the one-to-many system that translates out of English is a multilingual Transformer, using the pseudo-word approach to identify the desired language [Johnson et al., 2016]. It is trained on 231M sentence pairs sampled from the OPUS collection [Tiedemann, 2012] and covers 41 target languages, including all official EU languages. The out-of-English systems also use re-translation and prefix training.

To connect ASR and MT, we deploy an NMT-based segmentation component which converts the ASR output (all lower-case, no punctuation, with speech phenomena) into more standardized text by inserting punctuation, inferring capitalization and removing disfluency phenomena [Cho et al., 2012, Cho et al., 2017]. In addition to improving the readability of the

transcript, this component is necessary for MT, which is trained to expect orthographically correct (partial) sentences.

5 Presenting Translations

As mentioned above, our ASR systems are gradually *updating* their outputs, not necessarily in the incremental fashion. Similarly, the automatic segmentation can update the placement of punctuation symbols and as a result, the translations from the English-to-many MT systems can and will change over time. This situation (and the problems it brings when the space for the output translation is limited) is thoroughly explained by [Macháček and Bojar, 2020]. In short, there is no easy and non-disturbing solution if an update changes some content which has already been scrolled away due to a small presentation space. Eventually, a translated hypothesis becomes *confirmed*, meaning that the translation is final and will not change any more. While we could simply wait for the translation of a sentence to become confirmed and then display it to the users, it would introduce a needless and sometimes unacceptable delay.

Luckily, our setting allows us to use larger screen space than just the few lines of subtitles as [Macháček and Bojar, 2020] consider. We use the term "paragraph view" for this. Specifically, we developed a web-based interface for presenting the full live transcript of which the tail keeps changing. The web application is hosted on a web server, which receives the translated hypotheses from the MT system. The hypotheses are then published on a websocket, to which the browsers of end users connect. As the browser receives updates and finalized hypotheses from the websocket, it displays them to the end user. Finalized messages are displayed in black and unconfirmed hypotheses are displayed in grey, so the user can distinguish between them.

One important aspect of our setup is the relatively high number of available languages. While the presentation interface is flexible and can accommodate any number of languages, shown as columns, in practice showing too many languages leads to very narrow columns and, subsequently, the text scrolling too fast to be read. Additionally, each user will be interested in following only a very small number of languages and will want them displayed close to each other. A simple table of language codes therefore allows the user to choose which languages get displayed. This choice of languages can also be preloaded by an argument to the tool's URL entrypoint, allowing the event organizers to choose different default languages for different groups of users by spreading different versions of the link.

Based on our experience from several test sessions, we added the option for the operator to *broadcast* messages to users. There are many conditions of operation where some information from the operator would be very valuable for the spectators and would comfort them, such as "the show is delayed, stay tuned", "thank you for watching and we would like some feedback from you", or apologies for the current technical issues etc. We saw in practice that event organizers tend to choose very varied means and platforms of communication with the users, and the attention of users can also wander across them, so it is never certain *where* they would best notice. Broadcasting these messages interleaved with the main content of the transcribed and translated speech is a unifying option here.

It is important to note that these messages have to be delivered in all the supported target languages. To ensure the correctness of these messages, we collected a list of about 20 potentially useful English messages prior to the event. We translated them with our multi-target MT systems and asked many colleagues to review the automatic translations. For a few target languages, no native speaker of the language was available and the automatic translations remained unchecked. Based on the experience at the event, 8 more messages were added, primarily explaining immediate failures or delays that we observed in the *source* stream.

To differentiate them from the translations, these operator messages were displayed in bold. Figure 2 contains an example of the messages displayed to the user.

EN

1. This is a finished hypothesis.
2. **This is a message from the operator**
3. This is an unfinished hypothesis. It can be replaced by a more accurate hypothesis.

Figure 2: Example of the hypotheses and operator messages.

6 Operator Experience

The operator was facing a challenging task when the pipeline was running during the live event. He had to monitor all individual parts of the pipeline, spanning from the sound input to the very final presentation in the web interface, and he was also selecting the current best variant of English source for the multi-target translation, as described in more details below.

Because our current pipeline still misses automatic language identification, the operator had one additional task: based on the floor sound which he was constantly following inform the system about any change of the language spoken at the floor. The system was then prepared to redirect "sound pipes" accordingly, so that each of the 5 ASR system inputs always received its language, regardless whether it came from the original speaker (floor) or from an interpreter.

6.1 Noticing Problems

Despite long-term efforts in debugging all the components, some crashes did happen. They can be attributed to unexpected peculiarities of the incoming data and unexpected network conditions, for instance music or video with speech and music played in the main stream. Unexpectedly long silence (e.g. from an interpreter's booth) also occasionally caused the ASR+MT input pipe to timeout and crash.

Crashes are generally easy to spot (if the operator has the screen space and capacity to watch): some outputs become unavailable. What is more difficult to identify is *delay* in processing, e.g. due to some temporary network or system overload. With real interpreters and end-to-end neural ASR systems, a delay in the order of 4 to 7 seconds is the current standard [Macháček et al., 2021]. Noticing that this delay has grown to e.g. 10 or 20 seconds is not easy, esp. considering that there are several such inputs and each of them can suffer the problem individually and to a varying extent. In Section 7, we describe our new means to simplify the task.

6.2 Selecting the Current Best Source

The main responsibility of the operator in our setup was deciding which source of English text will be used as the input to the one-to-many MT system. As described above, there are multiple possible sources of the English text: directly from the speaker or from an interpreter, automatically translated into English as needed.

Each of the possible sources arrives as a sequence of updates. Our processing pipeline uses the automatically predicted punctuation to break it down into "events" aligned with sentence beginnings. An update can lead to multiple events if it contains several sentences in a row. Typically, updates are growing as more input words are recognized and processed, but regularly a "confirmation" update indicates that some history has been finalized and it will no longer appear in the updates. Updates from the different sources are fully independent of each other, with no synchronization at all.

As mentioned above, the subsequent step in the pipeline is the one-to-many MT system, which expects one stream of sentence events.

While we envision many cleverer techniques of input combination, for the described event, the operator was simply choosing which and only which stream on input events should be directed to the one-to-many MT; events from all other streams were discarded during that period.

Technically, each event is a line of text in the pipes. We needed a tool which serves as the `cat` command but allows to choose the source pipe on the fly. For this purpose, a simple Python program was developed. The program consumes an arbitrary number of line-oriented input channels using localhost ports and shows the latest message for each of those outputs to the operator. One of the streams is pre-selected as the default but the operator has the option to signal that for subsequent messages, a different stream should be used. The selection is done by writing the stream identifier to a special file which the program is monitoring.

The user interface for the operator was extremely simplistic for a start: a terminal window running the `watch` command repeatedly monitoring the last few lines of each input. Based on the past events and on the sound from the floor, the operator had to anticipate which input would be most reliable *in the future* events. Due to the asynchronicity of the updates, monitoring the sources was not always easy. What caused a particular problem were large updates that the fully neural ASR tended to make randomly.

At multiple times, the operator experienced a delay in the original English text while the interpreted and translated message was already available. In other words, the double interpretation (e.g. English speaker manually interpreted to French and machine-translated back to English) arrived sooner than the direct English ASR. This can be explained by the interpreter articulating the message to smaller and clearly identifiable chunks, so that the fully neural ASR was confident enough to ship them. With continuous English speech, the ASR was still waiting for a signal of the end of the sentence. Another possible explanation could be some temporary overload at the ASR system. At such occasions, the operator was tempted to (and often did) select some other language as the new source. Sometimes, this was a good choice because the direct English ASR was indeed stuck, but sometimes an update arrived shortly after switching away from that source.

An interesting opportunity to "travel in time" arose from the length of the updates. Sometimes, the operator switched to e.g. the German source because it was more up-to-date at that point. However, after the switch, the original English source was updated and this update covered also a portion of time before the beginning of the already emitted German source. Switching back to the English source thus actually repeated some of the transcribed speech of the current speaker, but worded differently. This situation allowed the operator to occasionally "rewrite" the latest updates, potentially improving the final text for the users. We still want to analyze this situation in a closer detail but improving the technique of input selection and combination is of a higher importance.

6.3 Interesting Specific Cases and Considerations

Live events always bring unexpected situations, beyond what any previous evaluation can cover. For example, one of the remote speakers started presenting in a fully unsupported language, so neither our system nor any of the interpreters knew what to do. This short unexpected silence caused some issues to our components.

Another unexpected situation occurred when a poor Internet connection distorted the speech of one of the (remote) speakers to the point where the interpreters refused to interpret it altogether. However, our English ASR systems were still able to process the audio, so for a short while our SLT service was actually the only source of translated speech. It surely suffered from recognition errors, but it was better than nothing.

We described most of the benefits and problems of the setup above. We have the recording and detailed logs from the event and the permission to use them for a limited period of

time. Portions of the recording which do not contain any confidential information will be released later, when the event organizers finish the manual check for confidentiality. When the publishable subset of data is selected, we plan a rigorous evaluation. We want to assess the true extent to which the alternative sources could have helped in producing better outputs and what the operator should have seen and noticed when selecting them. It is also likely that some translations were better for one source while for others, a different source should have been followed instead. Evaluating this aspect is even trickier: using standard reference-based evaluation methods cannot work because different sources lead to different reference translations and comparing scores across different translations is not possible.

6.4 User Feedback

During the event, we distributed a form for the users of the SLT system using operator broadcasting, mentioned in Section 5. Sadly, we received only three responses. Two users were using the SLT system all the time, reporting they preferred quicker, partial translations rather than slower, but more accurate translations. They indicated they would prefer subtitles over the paragraph view described in Section 5, but they were unaware of the problems the limited space of the subtitles brings.

The explicitly mentioned issues were too much text to read and distracting or laughable words. This calls for an improvement not only in terms of recognition and translation quality but also for some text condensation.

7 Pipeline Monitoring Tools

The pipeline is complex and consists of many separate components. It can be only expected that something will break, sooner or later. Thus, a set of tools monitoring the health and status of the pipeline was developed. Coming back to the pipeline representation as a directed acyclic graph, it makes sense to monitor two parts of the graph structure: the health of individual components (vertices) and the data flow between them (edges).

To monitor the individual components, the pipeline (as compiled to the bash script) saves (UNIX) process IDs of all the components. Then, a simple script regularly checks if processes with these IDs are still running, showing the status of each component to the operator. This allows for cursory checks of which components of the pipeline, e.g. a client connected to an ASR system, are up and running and which components have fully broken down.

Similarly, all intermediate component outputs and standard error outputs are duplicated to separate files via a modified `tee` which adds exact timestamps at the beginning of each saved line. This detailed (and now fully automated) logging proved essential both during the development as well as during the live event. These logs are recorded only on the operator's machine but it proved very useful to regularly upload them to a shared space where all technical team members could help investigating what is going on because the operator is generally fully occupied with other tasks. We are aware that there are server-based logging solutions but our approach is flexible, lightweight and does not need any external tools.

In debugging, absolute timestamps in the logs are necessary when investigating why the pipeline crashed. This usually involves cross-checking many logs and timestamps are the best means of finding the culprit. In several occasions, we also made use of these timestamped logs to replay some problematic input, allowing us to debug only one isolated component.

During live deployments, these log files are also monitored. The simplest approach taken at the reported event is to `tail -F` all the standard error outputs at once to see the latest errors of any component.

For the intermediate output files (i.e. the data that are passed along each of the pipeline graph edge) we developed a new tool. This tool tracks the moving average of the time between

output updates (simply checking for changes in log modification time). Whenever the average extension time elapses with a suitable margin but no output is added, the operator is notified. This is a very flexible detection of situations where the components are running, but for some reason they stop or slow down outputting data.

8 Future Improvements

Although the event went quite smoothly for us and there were no major technical issues or hiccups, we discovered some sore points and opportunities to improve. First, the task of the operator is quite demanding, as they have to constantly monitor the state of the pipeline, select the currently best-performing English text source and broadcast operator messages to the end users when necessary.

While the operator already has some tools to monitor the health of the pipeline, they still have to juggle multiple monitoring tools. To alleviate this, we propose to use a web application as the control center for the operator. The server for the web application would live in the Docker container, as described in Section 3.1, along with the pipeline and it would provide an API to obtain the last few lines of each stored-output file, list of running processes and other necessary information. The operator would then be able to simply observe and control the pipeline in their web browser.

Another improvement could be automatically switching the English text source input of the English-to-many MT system based on ASR confidence levels and other criteria. This would free the operator from having to constantly monitor the English text sources and judge which one is currently performing the best. However, the definition of these criteria would not be straightforward due to the different nature (and possible means of confidence estimation) of the components.

The operator messages were pre-translated before the event and revised for quality. A nice addition would be the option to simply type a new operator message in English, let that sentence be translated by the English-to-many MT system and then broadcast it to the end users.

9 Conclusion

We described our experience with running a complex system for spoken language translation aimed at substantially extending the set of provided target languages. From five official languages of the event, provided by human interpreters, we were able to cover 42 languages spoken in the participant's countries.

We proposed a novel technique increasing the overall robustness of the system to technical or human failures, namely following multiple sources at once and dynamically choosing the current best one. While the technique was so far tested in its simplest form, switching between the sources manually, it helped us to navigate through partial system failures. At one occasion, our system was the only translation service available, because even human interpreters have given up processing the sound from a distorted remote call.

For the future, we plan to improve the user interface for the operator. Any means of automatic diagnostics, incl. recognition and translation confidence, would be highly desirable. We will also focus on more advanced techniques for combining multiple inputs.

Acknowledgements

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreements No 825460 (ELITR).

References

[Cho et al., 2012] Cho, E., Niehues, J., and Waibel, A. (2012). Segmentation and punctuation prediction in speech language translation using a monolingual translation system. In *IWSLT 2012*.

[Cho et al., 2017] Cho, E., Niehues, J., and Waibel, A. (2017). Nmt-based segmentation and punctuation insertion for real-time spoken language translation. In *Interspeech 2017*.

[Franceschini et al., 2020] Franceschini, D., Canton, C., Simonini, I., Schweinfurth, A., Glott, A., Stüker, S., Nguyen, T.-S., Schneider, F., Ha, T.-L., Waibel, A., Haddow, B., Williams, P., Sennrich, R., Bojar, O., Sagar, S., Macháček, D., and Smrž, O. (2020). Removing European language barriers with innovative machine translation technology. In *Proceedings of the 1st International Workshop on Language Technology Platforms*, pages 44–49, Marseille, France. European Language Resources Association.

[Johnson et al., 2016] Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2016). Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *ArXiv e-prints*.

[Ma et al., 2019] Ma, M., Huang, L., Xiong, H., Zheng, R., Liu, K., Zheng, B., Zhang, C., He, Z., Liu, H., Li, X., Wu, H., and Wang, H. (2019). STACL: Simultaneous Translation with Implicit Anticipation and Controllable Latency using Prefix-to-Prefix Framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, Florence, Italy. Association for Computational Linguistics.

[Macháček and Bojar, 2020] Macháček, D. and Bojar, O. (2020). Presenting simultaneous translation in limited space. In *Proceedings of the 20th Conference Information Technologies - Applications and Theory (ITAT 2020)*, pages 32–37, Košice, Slovakia. Tomáš Horváth.

[Macháček et al., 2021] Macháček, D., Žilinec, M., and Bojar, O. (2021). Lost in interpreting: Speech translation from source or interpreter? In *Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association*. ISCA.

[Nguyen et al., 2020a] Nguyen, T. S., Niehues, J., Cho, E., Ha, T.-L., Kilgour, K., Muller, M., Sperber, M., Stueker, S., and Waibel, A. (2020a). Low latency asr for simultaneous speech translation. *arXiv preprint arXiv:2003.09891*.

[Nguyen et al., 2020b] Nguyen, T.-S., Pham, N.-Q., Stüker, S., and Waibel, A. (2020b). High performance sequence-to-sequence model for streaming speech recognition. *Proc. Interspeech 2020*, pages 2147–2151.

[Nguyen et al., 2020c] Nguyen, T.-S., Stueker, S., and Waibel, A. (2020c). Super-human performance in online low-latency recognition of conversational speech. *arXiv preprint arXiv:2010.03449*.

[Nguyen et al., 2020d] Nguyen, T.-S., Stüker, S., and Waibel, A. (2020d). Toward cross-domain speech recognition with end-to-end models. *arXiv preprint arXiv:2003.04194*.

[Niehues et al., 2016] Niehues, J., Nguyen, T. S., Cho, E., Ha, T.-L., Kilgour, K., Müller, M., Sperber, M., Stüker, S., and Waibel, A. (2016). Dynamic transcription for low-latency speech translation. In *Interspeech 2016*, pages 2513–2517.

[Niehues et al., 2018] Niehues, J., Pham, N.-Q., Ha, T.-L., Sperber, M., and Waibel, A. (2018). Low-Latency Neural Speech Translation. In *Proceedings of Interspeech 2018*.

[Pham et al., 2019] Pham, N.-Q., Nguyen, T.-S., Ha, T.-L., Hussain, J., Schneider, F., Niehues, J., Stüker, S., and Waibel, A. (2019). The iwslt 2019 kit speech translation system. In *Proceedings of IWSLT 2019*.

[Tiedemann, 2012] Tiedemann, J. (2012). Parallel data, tools and interfaces in opus. In Chair), N. C. C., Choukri, K., Declerck, T., Dogan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.