SCI-FII: Speculative Conversational Interface Framework for Incremental Inference on Modularized Services

Jinho Lee
IBM Research
Austin, Texas, USA
leejinho@us.ibm.com

Inseok Hwang*
IBM Research
Austin, Texas, USA
ihwang@us.ibm.com

Thomas Hubregtsen
IBM Research
Austin, Texas, USA
hubregts@us.ibm.com

Anne E. Gattiker
IBM Research
Austin, Texas, USA
gattiker@us.ibm.com

Christopher M. Durham
IBM Research
Austin, Texas, USA
cdurham@us.ibm.com

***Abstract*—We propose Sci-Fii, a speculative conversational interface framework for incremental inference on modularized services. To build one's own conversational interface with existing business logic, cloud-based modularized services offer a suite of ready-to-use components to ease development, ensure cross-platform flexibility, and encapsulate computational complexities. However developing with the modularized services often results in a chain of discrete modules with limited inter-module data sharing, which yields unnecessarily long response times of the conversational interface, aggravates user experiences, and eventually harms user retention. Sci-Fii offers a uniform framework that enables existing service modules to benefit from intermediate data and early parallel execution. Transparent to developers, Sci-Fii helps the end-to-end conversational interface system work fluidly and exhibit faster and more natural response times.**

***Index Terms*—conversational interface system, natural language processing;**

I. Introduction

What will be the next prevailing human-computer interfaces in the ubiquitous computing era? A design philosophy found in many experimental systems would be to minimize artificial intermediaries and let humans interact with the data or task in a direct and intuitive way. Visionary strategies include making digital entities tangible [1] or even making computing interfaces seem nonexistent [2]. Pursuing the visions, speech-based interactions have been highlighted [3], [4] as a natural modality to be weaved into the fabric of ubiquitous computing. In fact a growing number of commercial speech-based interaction systems are rapidly penetrating into our everyday mobile devices as well as living space, such as Amazon Echo [5], Google Home [6], and Apple Siri [7], allowing users to issue a query or command in a naturally spoken language.

The next leap towards even more natural speech-based interaction would be conversational interfaces [8]. Powered by advanced artificial intelligence and computational linguistics technologies, the user engages in a discourse with a computational agent, which is often conveniently referred to as a *chatbot*. Through a conversation with such an agent, the user takes turns and eventually completes a task or transaction, beyond just asking discrete queries. Conversational interfaces offer a huge potential in many third-party business front-ends, enabling their customers to make a booking, a purchase, or a problem resolution in the same way as they talk with a human agent. Moreover, conversational interfaces would facilitate developing a new genre of intelligent applications such as helping those with communication difficulties [9]–[11] or emulating specific conversation situations [12]–[14]. However, enabling a conversational interface in one's business front-end sets a higher bar than enabling discrete speech queries. The conversational interface is not solely a speech-to-text engine, but should have some knowledge of the existing business logic and work tightly together with the underlying transaction flows.

To provide *conversational interface-as-a-service* in an easy and flexible manner, cloud-based suites of modularized services have emerged [15], [16]. The building blocks of conversational interfaces are offered in individual modules, such as *speech-to-text* [17], *natural language classification* [18], *conversation flows* [19], *linguistic tone analytics* [20], *text-to-speech* [21], and so on. Each module can operate independently through its own APIs communicating with a client-side application. Such a suite of modularized services lets a business flexibly build their conversational interface–either with only the serviced modules or with a mixture of their own internal modules, third-party modules, and some of the serviced modules. Offering not only a flexible choice of individual blocks, this approach also benefits cross-platform deployments where different native modules are available and

*The corresponding author.

meets various security requirements.

In this paper, we propose Sci-Fii, a speculative conversational interface framework for incremental inference on modularized services. Sci-Fii is designed to circumvent the latency disadvantage we observed in conversational interface systems built with modularized services. The latency disadvantage is a side-effect of the modularized structure. Each module is supposed to be a self-contained box so that consecutive modules mostly hand over well-defined high-confidence outputs, rather than seamlessly sharing capricious internal status while a module is in the middle of computation. Also, each cloud-side module is often not communicating directly with other modules at neighboring stages but mediated by the client-side application. The limited inter-module connection makes it difficult for the resulting conversational interface to apply think-ahead or act-ahead strategies based on partial information, such as incremental dialog processing [22]–[24]. This makes modules begin operation only after the preceding module completes, accumulating latencies one after another.

Sci-Fii accelerates the end-to-end processing along multiple connected modules by creating speculative execution features on each module, while keeping every module unmodified as they are serviced. Sci-Fii is uniformly applicable to most existing modules, enabling each module to early-execute based on intermediate data, and aggressively hold the intermediate outputs for potential future reuse. In some way, it is conceptually similar to applying dynamic pipelined scheduling techniques [25]–[28] to modularized services.

In the following sections, we outline the common building blocks of conversational interface systems, discuss the details of what Sci-Fii offers to existing modules, and demonstrate the overall latency savings with a Sci-Fii-applied conversational interface system we built and evaluated against real utterance workloads.

II. Background

A. Common Modular Building Blocks

A conversational interface system takes a user through a multi-turn spoken dialog, naturally exchanging questions and answers to eventually complete a given task. In a modularized service structure, a conversational turn may go through multiple modules in order, such as:

- Speech-to-Text (STT): takes input of audio chunks of user utterances; outputs the transcribed text with confidence ratings. The transcription may have multiple candidates. Example STT services include Watson STT [17], Google Cloud Speech [29], Bing Speech APIs [30], and Nuance Speech Recognition [31]. Most STT services provide APIs for getting intermediate results.
- Natural Language Processing (NLP): takes input of the transcribed utterances; outputs a key intent embedded in the given utterance despite many different ways to paraphrase it [18], [32]–[34]. An example of form of NLP for conversational interface systems is Natural Language Classfication (NLC) [18], [34], which outputs pre-defined classes and provides confidences. NLC may use a pre-trained machine learning model to classify sentences. Pre-training may be done with the application-specific example utterances.
- Response generation: takes input of a classified intent label; outputs a textual response or an action command; updates the internal state that represents the conversation progress [19].
- Text-to-Speech (TTS): takes input of a textual response; outputs synthesized audio that verbalizes the response [21], [30].
- Other: additional statistical or machine learning tools may be involved in the middle to determine the user emotions [20], retrieve personalized recommendation items [35], classify a user persona [36], translate the input or output into a different language [37]–[39], or retrieve external data fields (e.g., product names or prices).

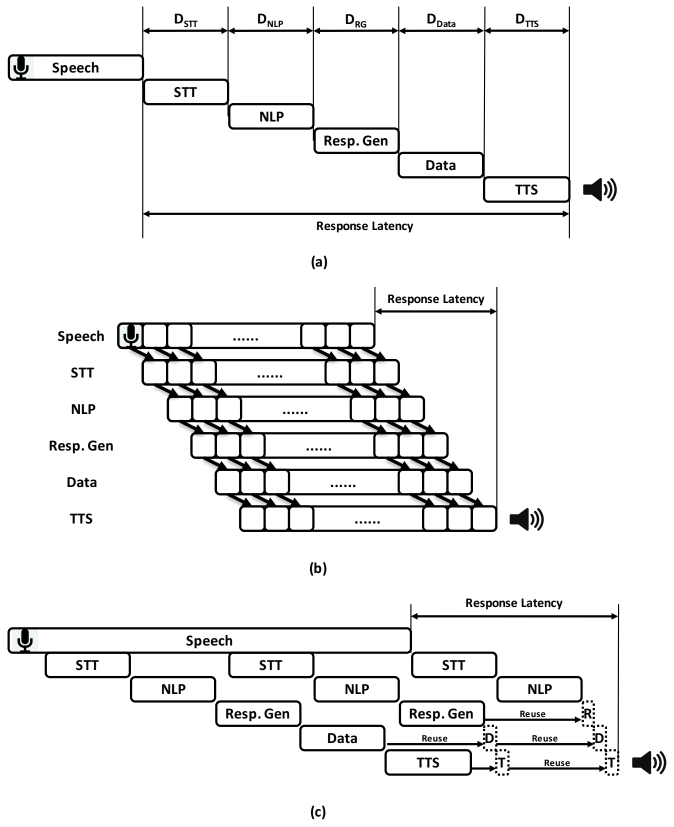

Fig. 1. Timing diagrams of (a) baseline conversational interface system (b) incremental processing system (c) Sci-Fii. (Diagrams are not in scale)

B. Incremental Processing Basics

Fig. 1 (a) shows the timing diagram of the example baseline conversational interface system consisting of five stages: STT, NLP, Response generation, Data management, and TTS. As outlined in § I, This structure often accumulates a long latency that may annoy the user.

On the other hand, incremental dialog processing helps accelerate the processing and mitigate the latency issue. Fig. 1

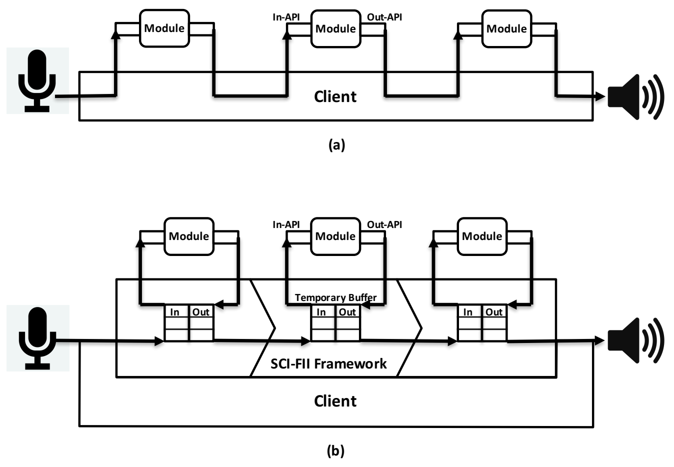

Fig. 2. Architecture of (a) baseline conversational interface system (b) Sci-Fii.

(b) shows the timing diagram of a dialog system with incremental processing. Incremental processing allows a module to initiate as soon as some intermediate information is available from the preceding module. In this example, the STT module outputs updated transcription as the user continues speaking. Upon every update of the partial transcription, the NLP module attempts to extend the understanding of the intent with the newer transcript. While this could reduce a great amount of latency in the overall conversation interface system, it requires each module to be able to access each other in a deeper level, or to define sophisticated protocols compatible with other modules to exchange intermediate data in between. This requirement is nontrivial for a system with heterogeneous modules from different platforms, making them hardly interoperable, aggravating the development complexity, and often elevating security issues. For many businesses building their own conversational interfaces with modularized services, incremental processing features currently face significant implementation barriers.

III. Sci-Fii Architecture

A. Overview

A conceptual architecture of the baseline system and Sci-Fii are shown in Fig. 2. To help a conversational interface system built with existing modularized services benefit from incremental processing features, Sci-Fii provides a uniform, convenient framework that wraps each module provided from existing modularized services. Transparent to developers, Sci-Fii wraps each module with its own temporary buffer that holds recent inputs and outputs of this module. Each wrapper provides a subscription-and-triggering service that lets the module subscribe to the temporary buffer of the preceding module and trigger itself whenever the buffer updates the output. It is a uniform feature that is applicable to virtually every module in a conversational interface system, enabling most modules to execute as soon as an utterance begins.

The timing diagram of a Sci-Fii-applied conversational interface system is shown in Fig. 1 (c). The system is a sequence of stages each of which is built with a module wrapped with Sci-Fii. Sci-Fii runs in two phases: *speculative execution* and *final execution*. In the speculative execution phase, the framework keeps triggering each stage upon any partial update of the on-going utterance and fill each stage's temporary buffers with speculative execution results. After detecting an *EOS* (End-Of-Speech), it switches to final execution where all the stages are orchestrated to generate the actual responses. Sci-Fii achieves the latency saving by holding all the speculative execution results in the temporary buffers. Time savings are achieved if Sci-Fii finds those results reusable when the utterance ends, so that the internal operation of a stage may be skipped.

In the remainder of the section, we will first explain how the two phases work in more detail (§ III-B), and provide an analytic speedup model regarding to the flow (§ III-C). Lastly, we will provide two additional optimizations which could increase the speedup (§ III-D).

```
1   Speculative_exe (input, stage) :
2     if input == tmpBuf.input
3         return
4     tmpBuf.in_progress = true
5     tmpBuf.input = input
6     output = stage.run(input)
7     tmpbuf.output = output
8     tmpBuf.in_progress = false
9     if stage.is_last
10        return
11    else
12        thread_start (Speculative_exe,
13                   output, stage.next)
                          (a)
```

```
14  Final_exe (input, stage) :
15    if input == tmpBuf.input
16        while(tmpBuf.in_progress == true)
17        output = tmpbuf.output
18    else
19        output = stage.run(input)
20    stage.commit(output)
21    if stage.is_last
22        inform_user(output)
23    else
24        Final_exe (output, stage.next)
                          (b)
```

Fig. 3. Procedures for (a) speculative execution and (b) final execution.

B. Execution Flow

As mentioned in § III-A, Sci-Fii continuously executes the stages with partial inputs before speech is done. This is called the speculative execution phase and its procedure is shown in Fig. 3 (a). Sci-Fii keeps a temporary buffer for each stage. Each buffer entry stores the most recently processed input and the corresponding output. In addition, there's a 1-bit "in-progress" flag. At each stage, the input field in the temporary buffer is compared with the input (line 2). If they match, the speculative execution terminates since there's no need to proceed any further (line 3). If they do not match, the input field is overwritten and the progress flag is set (line 4-5). After the execution (line 6), the output field is filled, and the progress flag is cleared (line 7-8). Finally, the next stage's execution is instantiated and the current execution (i.e., thread) terminates (line 12).

After EOS is detected, Sci-Fii enters the final execution phase (Fig. 3 (b)). Similar to the speculative execution phase, the temporary buffer is checked at every stage to see that the input has been already processed (line 15). If the input in the buffer entry does not match, the stage is executed normally (line 18-19). However, the buffer entry is not written in this case since this is the final execution and the result is not going to be reused afterwards. If the input matches the value in the temporary buffer, it means that the same input has been processed by speculative execution and the output from the buffer entry can be used without further execution (line 17). To avoid reading the output field before the speculative execution finishes the execution, the final execution thread waits for the in-progress flag to be unset before reading the output (line 16). In practice, the in-progress flag can be implemented using simple locks.

A negative side effect that speculative execution imposes is unwanted state/data modifications. While most modules such as STT or NLP are stateless[1], some stages in conversational interface systems are stateful or involve external data modifications. For instance, the response generation stage might be waiting for a greeting, but after a speculative execution, it would proceed to a new state and produce a wrong output for the actual response. Also, if the data management stage is writing something to a database, many speculative execution would leave non-intended multiple writes on the database.

For such reasons, certain stages need to be made transactional, which support separate APIs for execution and commit. Transactional stages produce outputs on execution calls, but make actual updates on the states or data only on commit calls. The speculative execution phase only makes execution calls, while the final execution phase calls commits after executing or re-using the results to reflect changes (line 20).

While we expect the modularized services to eventually adopt transactional APIs, those are typically not available as of today. Fortunately, any stages that write data outside the system are likely to be managed by the end-developer or a highly customizable service [40] so that the end-developer can provide transactional APIs. For others stages that heavily depend on internal states (e.g., Watson Dialog [41]), a *roll-forward* method can be used to mimic the transactional stages. In a roll-forward method, the dialog system may employ a pool of multiple instances holding the state of the conversation progress, and keep the history of inputs for the stage. On speculative execution, only one of the stages is executed while the others remain in the same state. When the speculative execution turns out to be a failure, the system restarts the instance and rolls it forward to the correct state using the input history, making it available for receiving inputs again.

[1]STT or NLP may actually have some state changes such as adopting to certain domains, but it would not incur major differences.

C. Estimated Speedup

Using modularized service, the latency L_{base} of the baseline dialog system can be simply calculated as below:

$$L_{base} = \sum_{0 \le i < S} D_i \tag{1}$$

where S is the number of stages in the system, D_i is the average processing time of stage i, including the network delays.

In speculative execution, if the temporary buffer for a stage matches, none of the later stages is executed, while the final execution might have to wait for speculative execution to finish (i.e., wait for the progress flag to be unset). In such a manner, the expected latency L_{spec} can be described as below[2]:

$$L_{spec} = \sum_{0 \le i \le S} \Big(\big(\prod_{0 \le j < i} M_j \big)(1 - M_i) \big(\max(L_{base} - \Delta T_i,\ 0) \big) \Big) \tag{2}$$

where M_i is the probability of mismatch for the final execution in stage i, and ΔT_i represents the time difference between initiation of the final execution and the initiation of the speculative execution that wrote the temporary buffer for stage i. M_S and ΔT_S (for "$S+1$"th stage) are constants where $M_S = 0$ and $\Delta T_S = 0$. They are dummy variables used to represent the case where no match is found in all stages and therefore $L_{spec} = L_{base}$.

The insight from the equation is simple: the only control knob we have is M_i, and it's likely that we'll achieve more speedup if we can make M_i smaller for earlier stages. In next subsection, we'll introduce a few techniques to reduce M_i, thereby maximizing the speedup coming from Sci-Fii.

D. Optimizations

1) Multi-entry temporary buffer: In the middle of a speech, the stages occasionally make a mistake and correct them. For example in the STT stage, the phrase "*check in*" might get correctly transcribed, mistakenly transcribed as "*chicken*", and come back to "*check in*" again. On such events, the later executions cannot re-use the outputs produced by earlier stages. To cope with such situations, we keep multi-entry temporary buffers for each stage, which consists of multiple (input, output) pairs. Each execution checks all pairs available before executing the stage. We implement the multi-entry buffers using hash tables to minimize the lookup latency. Also, when the memory size for the entries is limited, we set the maximum number of entries and use LRU policy to evict them.

[2]Here, we made two simplifications: 1) We assume that the stage delays are independent of the input; 2) We do not consider cases where results from two or more iterations of speculative execution being reused. In reality a speculative executionmay encounter a match on results from earlier speculative executions. If such recursively matched cases are considered, the expected latency will be even shorter.

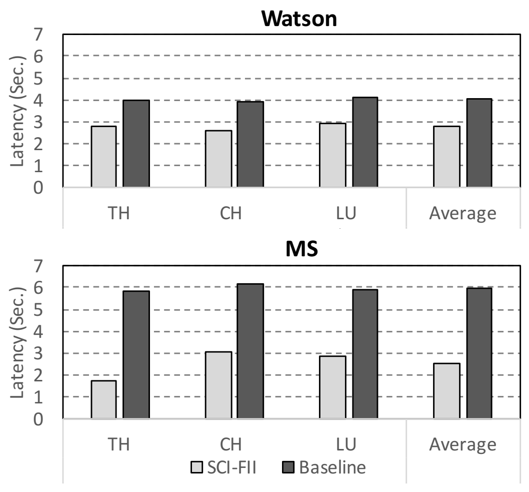

Fig. 4. Average latency of the baseline conversational interface system and SCI-FII over different scenarios.

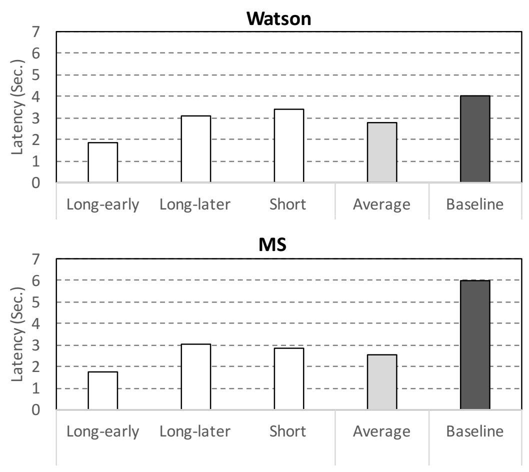

Fig. 5. Average latency of the baseline conversational interface system and SCI-FII over different type of speeches.

2) Multi-instances: To further reduce M_i, multiple speculative executions can be instantiated instead of one. With the increasing prevalence of deep learning, many modularized services are likely to provide multiple outputs with different confidences (e.g., "*Check in*" with 70% confidence, and "*chicken*" with 30% confidence). We take advantage of this feature and instantiate next stage speculative execution with the top F outputs in parallel, where F is the fan-out parameter. We can control F to exhibit a system resource-latency trade-off.

IV. Evaluation

A. Evaluation Platform

We designed our Sci-Fii prototype on top of publicly available IBM Watson services [15] and Microsoft (MS) cognitive services [16]. Note that Sci-Fii requires an STT module to support real-time audio streaming with intermediate result updates. At the time of writing, both services provide this feature in their respective STTs, but on different platforms and through different technologies. Watson STT provides WebSocket APIs for real-time streaming with intermediate updates [42], so that Sci-Fii can work on any platform where WebSocket is available. However, in Bing Speech APIs, real-time streaming with intermediate updates is provided through platform-specific client libraries [43]. Due to this reason, we built our test systems on different platforms for each service. To work with Watson services, we built a test system in Python 2.7 using an off-the-shelf WebSocket package [44] on Mac OS X 10.11. To work with Microsoft services, we built a test system in Java using their Android Client Library [45] on Android 7.1.1.

Each test system includes four modularized service stages: STT [17], [30], NLC [18], [34], Response generation, and TTS [21], [30]. We omitted data management because it would be so specific to the application and it would not be fair to make latency comparisons with a specific instance of it included. Also, we built our own response generation as close as possible to Watson conversation [19], because Watson conversation did not support transactional execution, and no equivalent response generation service was available in Microsoft cognitive services at the time of writing.

We deployed three conversational interface systems featuring the scenarios below:
What's in the theaters (TH) [46] is an example of Watson NLC and Dialog that makes a conversation about movies bases on databases. **Airline check-in (CH)** is an in-house scenario which represents a situation in an airport where the customer wants to check in. **What's for Lunch (LU)** is another in-house example that recommends a few restaurants nearby.

We recorded utterances from four people, and measured the latencies. Each person was requested to ask a question in several different ways per scenario, collecting a total of 45 utterances.

B. Experimental Results

Fig. 4 shows the latency results. The brighter bars represent the latencies of Sci-Fii-applied conversational interface system, while the darker bars represent those of the baseline conversational interface system. Each utterance was run through the system ten times at different times of a day and the average was taken, to amortize the diurnal deviations of network conditions and the internal service workloads of modularized services that we do not have control over.

With Watson services, the average latency of the baseline system was 4.02 seconds after the user's utterence ends. Sci-Fii achieved 2.78 seconds latency, reducing 30.9% from the baseline. There was no meaningful difference between the scenarios, since the modularized services used usually do not depend on the contents. For MS cognitive services, Sci-Fii shows 2.54 seconds latency on average and baseline shows 5.98 seconds where Sci-Fii marks 57.5% improvement. The trends were almost the same, except that the baseline latency is longer then that of the Watson services.

We found out that the differences between two services mostly come from the trend that MS cognitive services tend to detect EOS in a more conservative way with a longer silence threshold. Note that a shorter EOS does not necessarily mean an advantage but is rather a matter of different optimization goals. While Watson's shorter EOS threshold generally yields shorter latency, we observed that Watson STT sometimes misinterpreted an EOS prematurely upon the user's short pause in the middle of speech. On the contrary, MS cognitive service with a longer EOS threshold would generate fewer such premature EOSes but may increase the overall latency. We also note that our test systems run on different platforms for each service. Therefore, we do not conclude at this point that one service performs better than the other service.

To further analyze the speedup, we manually categorized the utterance samples into three cases: long speeches where the keywords appear in the earlier half of the utterance (*Long-early*), long speeches where the keywords appear in the later half (*Long-later*) and short speeches having less then six words (*Short*).

Fig. 5 shows the latency results for different cases. As expected, speeches in the Long-early category shows the lowest latencies, and more than 2 seconds of saving compared to the baseline. Actually, we found that in many cases the response times were almost zero, indicating the correct response was already fully prepared while the speech was going on. The most frequent source of speedup was between the NLC and Response generation stage, where the NLC stage classified the partial speech into the same one as one in the full speech.

Long-later and Short type speeches experience much longer latencies, showing relatively small improvements compared to the long-early cases. For Short speeches, Sci-Fii simply does not have enough time to do a speculative execution due to the short duration of speech. However, Sci-Fii still achieves a considerable latency reduction. This is because STTs usually detect EOS upon the presence of a certain duration of silence (about 0.5 seconds in Watson STT, 2 seconds in Bing speech API). When STT informs EOS and Sci-Fii starts the execution, the output of STT stage would be exactly the same as before the silence, and the results after STT stages could be re-used. This accounts for the latency reduction in Short speeches.

Similar explanation can be applied to Long-later cases, because what Sci-Fii has speculated in earlier part of the utterance is not likely to match the result of the final execution. However, there's still a small difference between Long-later cases and Short cases, because long sentences still give some chance to make a correct speculative execution even if the keyword is at the later part. For example, consider the sentence "*Hi, I am going to Chicago today, can I check in?*". We categorized this sentence as Long-later, because the keyword was "*check in*". Sci-Fii may pick up "*going to Chicago*" and generate the same NLC result as the final one, or a different NLC result but the same output of response generation.

V. Discussion

Cost consideration. As demonstrated in § IV, adopting Sci-Fii into existing modularized conversational interface systems seems to be a no-lose scenario in terms of latency. Sci-Fii likely reduces the overall response latency and still retains the same latencies at the worst case. However, using Sci-Fii would multiply the volume of computational resources, which turns into the metric of money for most clients utilizing cloud-based services on a pay per API call pricing. A single user utterance would generate multiple partial input updates, each of which may trigger API calls in some or all modules along the execution flow. Thus, simply adopting Sci-Fii in all utterances may multiply the monetary cost to maintain the conversational interface. A smarter strategy would be to refer to the confidence scores that come with intermediate results, so that the system selectively calls a succeeding module's API only if the preceding module produce an intermediate result with a reasonable confidence score. An appropriate threshold of the confidence score would be a trade-off between the expected additional API cost and the longer user retention improved by the reduced latencies.

Streaming feature of STT. Sci-Fii requires the STT module to support the streaming feature, i.e., producing intermediate transcription while the user utterance is in progress. We believe that it is not a strong requirement, as many commercial STT services provide both API entry points for non-streaming and streaming. Non-streaming APIs are mostly provided in REST APIs, while streaming APIs are provided in varying technologies, e.g., WebSocket [17], RPC [29], libraries for native client platforms [30]. Some conversational interface systems may use textual input instead of speech (e.g., chatbots) where taking inputs of individual key strokes naturally supports the streaming feature. Overall, we believe that the streaming feature would be already a commodity available to many conversational interface systems.

Tailoring the training strategies. So far we have developed and evaluated Sci-Fii with the original natural language classification model which has been trained with complete user utterances. If the use of Sci-Fii is expected at the time of training, one may consider extending the training corpus with not only complete utterances but also partial utterances. Extending the corpus will be trivial once the complete utterances are given. An open question is whether including partial utterances in the training corpus would indeed improve the early classification accuracy, or negatively impact the accuracy due to some elevated ambiguity. While there are some literatures on understanding natural language based on partial speeches [47], investigating the effect of training modularized services with partial utterances and crafting the corpus are still left as future issues to be studied.

Transactional stages. The only modification that Sci-Fii requires to the modularized services is transactional stages. Even though we provide a roll-forward method (§ III-B) to work around the transactional APIs, it would be a waste of system resources and may cause explosion in API calls, resulting in being over-charged under pay per API call pricing. As of our understanding, providing transactional APIs would not be a significant burden at all from the service provider's perspective. Furthermore, transactional APIs would be useful to not only speculative execution methods but also many other purposes such as debugging or internal testing. We envision that service providers will incorporate these features in the future.

Extension to non-serial pipeline structures. So far in this paper, we assumed that the execution of conversational interface systems follow a single execution path without branching or merging. However, conversational interface systems could easily require much complicated structure. For instance, consider that the conversational interface system above also wants to see a live video stream to figure out the speaker's emotional status by reading face. It would make more sense to process the video in parallel with the speech than to do them serially. Fortunately, extension of Sci-Fii to support non-serial structure does not require a sophisticated method. When different execution flows merge, we add an additional buffer for each input port at the merging stage to save the most recent input value from the port. When any one of the inputs arrives, Sci-Fii runs an speculative execution using the arrived input and the saved inputs from other ports together.

VI. Conclusion

In this paper, we propose Sci-Fii, a speculative conversational interface framework for incremental inference on modularized services. Thanks to modularized services universally available, building speech-based interaction systems has become as easy as mix-and-matching different services together. However at the same time, they suffer long latencies mainly from shallow interfaces between each other. Sci-Fii provides a framework that can mitigate the latency problem by creating speculative executions on each modules in a developer-transparent manner. Without invading the modularized services as being serviced, Sci-Fii reduces response latency by 42.2% on average from two popular service providers. We envision methods similar to Sci-Fii to be widely used, and thus steering the modularized services to provide features friendly to speculative executions, where specifying those features would be our future work.

References

[1] H. Ishii and B. Ullmer, "Tangible bits: towards seamless interfaces between people, bits and atoms," in *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*. ACM, 1997, pp. 234–241.

[2] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[3] G. D. Abowd and E. D. Mynatt, "Charting past, present, and future research in ubiquitous computing," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 1, pp. 29–58, 2000.

[4] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 1, pp. 3–28, 2000.

[5] Amazon echo. Accessed: Dec. 18, 2016. [Online]. Available: http://www.amazon.com/oc/echo/

[6] Google home. Accessed: Dec. 18, 2016. [Online]. Available: http://home.google.com/

[7] Apple siri. Accessed: Dec. 18, 2016. [Online]. Available: http://www.apple.com/ios/siri/

[8] C.-M. Karat, J. Vergo, and D. Nahamoo, "Conversational interface technologies," in *The human-computer interaction handbook*. L. Erlbaum Associates Inc., 2002, pp. 169–186.

[9] K. H. Jeon, S. J. Yeon, Y. T. Kim, S. Song, and J. Kim, "Robot-based augmentative and alternative communication for nonverbal children with communication disorders," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2014, pp. 853–859.

[10] I. Hwang, C. Yoo, C. Hwang, D. Yim, Y. Lee, C. Min, J. Kim, and J. Song, "Talkbetter: family-driven mobile intervention care for children with language delay," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 1283–1296.

[11] Y. Lee, C. Min, C. Hwang, J. Lee, I. Hwang, Y. Ju, C. Yoo, M. Moon, U. Lee, and J. Song, "Sociophone: Everyday face-to-face interaction monitoring platform using multi-phone sensor fusion," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 2013, pp. 375–388.

[12] M. E. Hoque, M. Courgeon, J.-C. Martin, B. Mutlu, and R. W. Picard, "Mach: My automated conversation coach," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013, pp. 697–706.

[13] B. Kang, C. Min, W. Kim, I. Hwang, C. Park, S. Lee, S.-J. Lee, and J. Song, "Zaturi: We put together the 25th hour for you. create a book for your baby," in *Proceedings of the 20th ACM conference on Computer supported cooperative work & social computing*. ACM, 2017.

[14] H. Trinh, L. Ring, and T. Bickmore, "Dynamicduo: co-presenting with virtual agents," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 1739–1748.

[15] IBM Watson. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/

[16] Microsoft cognitive services. Accessed: Jan. 4, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/

[17] Watson speech-to-text. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/developercloud/speech-to-text.html

[18] Watson natural language classifier. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/developercloud/nl-classifier.html

[19] Watson conversation. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/developercloud/conversation.html

[20] Watson tone analyzer. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/developercloud/tone-analyzer.html

[21] Watson text-to-speech. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/developercloud/text-to-speech.html

[22] F. Ghigi, M. Eskenazi, M. I. Torres, and S. Lee, "Incremental dialog processing in a task-oriented dialog." in *INTERSPEECH*, 2014, pp. 308–312.

[23] S. Heintze, T. Baumann, and D. Schlangen, "Comparing local and sequential models for statistical incremental natural language understanding," in *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, 2010, pp. 9–16.

[24] G. Skantze and D. Schlangen, "Incremental dialogue processing in a micro-domain," in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 745–753.

[25] J. Lee and K. Choi, "Memory-aware mapping and scheduling of tasks and communications on many-core SoC," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2012, pp. 419–424.

[26] J. Lee, M.-K. Chung, Y.-G. Cho, S. Ryu, J. H. Ahn, and K. Choi, "Mapping and scheduling of tasks and communications on many-core soc under local memory constraint," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1748–1761, 2013.

[27] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proceedings of the International Symposium on Microarchitecture*. ACM, 1994, pp. 63–74.

[28] M. V. Eriksson and C. W. Kessler, "Integrated modulo scheduling for clustered vliw architectures," in *International Conference on High-*

Performance Embedded Architectures and Compilers. Springer, 2009, pp. 65–79.

[29] Google cloud speech API. Accessed: Dec. 21, 2016. [Online]. Available: https://cloud.google.com/speech/

[30] Bing speech APIs. Accessed: Jan. 4, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/speech-api

[31] Nuance. Accessed: Dec. 21, 2016. [Online]. Available: https://developer.nuance.com/

[32] L. Del Corro and R. Gemulla, "Clausie: clause-based open information extraction," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 355–366.

[33] S. Han, M. Philipose, and Y.-C. Ju, "Nlify: lightweight spoken natural language interfaces via exhaustive paraphrasing," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013, pp. 429–438.

[34] Microsoft language understanding intelligent service. Accessed: Jan. 4, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/language-understanding-intelligent-service-luis

[35] Microsoft recommendations API. Accessed: Jan. 4, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/recommendations-api

[36] Watson personality insights. Accessed: Jan. 5, 2017. [Online]. Available: https://www.ibm.com/watson/developercloud/personality-insights.html

[37] Watson translator. Accessed: Jan. 5, 2017. [Online]. Available: https://www.ibm.com/watson/developercloud/language-translator.html

[38] Microsoft translator API. Accessed: Jan. 4, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/translator-api

[39] Google cloud translation API. Accessed: Jan. 5, 2017. [Online]. Available: https://cloud.google.com/translate/

[40] Node-RED Watson Bluemix starter application. Accessed: Dec. 18, 2016. [Online]. Available: https://github.com/watson-developer-cloud/node-red-bluemix-starter

[41] Watson dialog. Accessed: Dec. 18, 2016. [Online]. Available: https://www.ibm.com/watson/developercloud/dialog.html

[42] Watson stt - using the websocket interface. Accessed: Apr. 6, 2017. [Online]. Available: https://www.ibm.com/watson/developercloud/doc/speech-to-text/websockets.html

[43] Bing speech api overview. Accessed: Apr. 6, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/Speech-api/documentation/overview

[44] ws4py - a websocket package for python. Accessed: Apr. 6, 2017. [Online]. Available: https://ws4py.readthedocs.io/en/latest/

[45] Get started with bing speech recognition in java on android. Accessed: Apr. 6, 2017. [Online]. Available: https://www.microsoft.com/cognitive-services/en-us/Speech-api/documentation/GetStarted/GetStartedJavaAndroid

[46] What's in theaters. Accessed: Dec. 21, 2016. [Online]. Available: https://watson-movieapp-dialog.mybluemix.net/watson-movieapp-dialog/dist/

[47] K. Sagae, G. Christian, D. DeVault, and D. R. Traum, "Towards natural language understanding of partial speech recognition results in dialogue systems," in *Proceedings of the 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*. Association for Computational Linguistics, 2009, pp. 53–56.