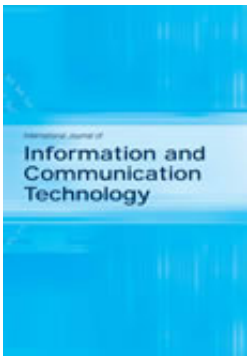
INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research



International Journal of Information and Communication Technology

ISSN online: 1741-8070 - ISSN print: 1466-6642
https://www.inderscience.com/ijict

From toy-like to intelligent: ARM-based cloud-injected AI companion robot

Yonggang Zhang, Xiao Tan, Xilong Qu

Article History:

Received:	18 January 2024
Last revised:	07 March 2024
Accepted:	14 April 2024
Published online:	13 June 2024

From toy-like to intelligent: ARM-based cloud-injected AI companion robot

Yonggang Zhang

Department of Science,
Gansu University of Chinese Medicine,
Lanzhou, 743000, China
Email: 362327630@qq.com

Xiao Tan*

School of Information Technology and Management,
Hunan University of Finance and Economics,
Changsha, 410205, China
Email: tanxiao@hufe.edu.cn
Email: tedarte731@21cn.com
*Corresponding author

Xilong Qu

School of Information Science and Engneering,
Changsha Normal University,
Changsha, 410100, China
Email: 42559003@qq.com

Abstract: Our research in child companion robotics is evolving from simple toys to intelligent, ARM-based, cloud-connected AI companions. Historically, these robots have been limited by predefined interactions and speech recognition capabilities. Our innovation lies in the implementation of advanced transformer models, which have reduced word error rates by 12% compared to other models. Overcoming challenges posed by resource constraints, we optimised these models for ARM architecture, ensuring high performance in embedded systems. Furthermore, integrating cloud-injected AI enhances our robots' intelligence by providing access to real-time data, thus enriching conversations and adaptability. This integration, coupled with advancements in speech recognition, represents a significant leap towards interactive companions capable of engaging meaningfully with children. Our research marks the onset of a new era in child companion robotics, demonstrating the transformative potential of ARM-based, cloud-injected AI systems.

Keywords: speech processing; cloud integration; conversation relevance; transfer learning; dynamic injection.

Reference to this paper should be made as follows: Zhang, Y., Tan, X. and Qu, X. (2024) 'From toy-like to intelligent: ARM-based cloud-injected AI companion robot', *Int. J. Information and Communication Technology*, Vol. 24, No. 7, pp.17–50.

Biographical notes: Yonggang Zhang graduated from Gansu University of Chinese Medicine, majoring in Medical Software Engineering. He is passionate about researching software development methodologies and quality assurance, aiming to enhance the quality and efficiency of medical software by integrating medical knowledge with software engineering.

Xiao Tan is an expert with a profound research background in chip technology, hardware security, and the Internet of Things. His career is focused on exploring and innovating advanced technological solutions, especially in the fields of microelectronics and information security. His research primarily involves developing more efficient methods for chip design and exploring new types of hardware security strategies, aimed at protecting data from unauthorised access and tampering.

Xilong Qu is a distinguished figure in the field of artificial intelligence and machine learning, with a keen interest in ethical AI development. His expertise encompasses the creation of algorithms that can learn from and adapt to their environment, improving decision-making processes in various sectors such as healthcare, finance, and autonomous vehicles. His work focuses on ensuring that AI systems are developed with fairness, accountability, and transparency, to benefit society as a whole.

1 Introduction

The emergence of intelligent AI companion robots signifies a pivotal shift in the intersection of technology, education, and personal companionship, specifically tailored to address the nuanced needs of modern childrearing practices. In the realm of child companionship, a growing challenge arises as parents find themselves engrossed in the relentless pace of modern life, leaving their children to navigate the solitary confines of home (Ruiz-Casares and Heymann, 2009). The prevalence of single-child households, has made meaningful communication between parents and children increasingly elusive. This lack of interaction not only hinders the holistic growth and cognitive development of children but also underscores the critical need for providing adequate care and companionship (Zhiwei et al., 2018). In response to these challenges, the integration of ARM-based devices in the development of intelligent AI companion robots offers a promising solution. ARM-based devices, renowned for their low power consumption and high performance, lay the hardware foundation essential for creating compact, energy-efficient, and powerful companion robots. These devices enable the processing of complex algorithms and management of real-time interactions, facilitating the development of robots that are interactive, responsive, and practical for everyday use. Leveraging the capabilities of artificial intelligence (AI), natural language processing, and cloud computing, our approach aims to revolutionise child companionship. These advanced technologies enable the creation of seamless interactions between child-friendly robotic entities that can listen attentively and respond effectively, promising to bridge the communication gap in modern families. The potential of AI and ARM technology in this context is vast, offering transformative possibilities not only in child companionship but also across diverse industries and domains (Lin, 2017; Jiang, 2020). For instance, within the Internet of Things landscape, cloud technology facilitates efficient data processing and optimal outcomes, highlighting its inherent advantages such

as streamlined local processing, cost reduction, and accelerated product development cycles (Zhang et al., 2022; Atieh, 2021).

To enable intelligent and context-aware conversations with child companion robots, we employ advanced AI techniques such as deep learning and neural networks. These techniques allow the robots to understand and respond to children's queries and commands with unprecedented precision and relevance. By leveraging state-of-the-art Transformer models, we achieve significant improvements in speech recognition and language understanding. Our approach surpasses traditional speech recognition systems, such as hidden Markov models (HMMs) and recurrent neural networks (RNNs), which have been the backbone of conventional systems. While HMMs and RNNs have demonstrated reasonable performance, they often struggle with capturing long-term dependencies and handling complex language structures. In contrast, our Transformer models excel in these areas, enabling more accurate and context-aware responses. This advancement is particularly crucial for child companion robots, where nuanced understanding and adaptability are essential for engaging and meaningful interactions. These models excel at capturing contextual information and generating accurate responses, enhancing the overall conversational experience. Furthermore, we harness the power of cloud computing to augment the capabilities of child companion robots. By seamlessly integrating with cloud-based knowledge bases and services, the robots can access a vast amount of information and resources in real-time. This enables them to provide up-to-date and accurate responses to children's questions, as well as access a wide range of educational content and interactive activities. The cloud-based knowledge bases are continuously updated and enriched, ensuring that the robots have access to the latest information and can adapt to evolving needs and interests.

In addition to speech recognition and cloud integration, we also focus on enhancing the emotional intelligence of child companion robots. Emotion detection and understanding play a crucial role in creating meaningful and empathetic interactions. By leveraging advanced techniques such as affective computing and facial recognition, the robots can perceive and respond to children's emotions, providing comfort and support when needed. This emotional connection helps to establish a deeper bond between the child and the robot, fostering a sense of companionship and understanding. To ensure the safety and well-being of children, we implement robust content filtering mechanisms. By leveraging AI-powered content analysis and filtering algorithms, we can identify and block inappropriate or harmful content, ensuring that the interactions between the child and the robot are age-appropriate and aligned with parental guidelines. This proactive approach to content filtering helps to create a safe and nurturing environment for children to interact with their robotic companions. Through rigorous research and development, we continuously evaluate and improve the performance and capabilities of our child companion robots. User feedback and real-world usage data are invaluable in shaping the evolution of our child companion robots. We conduct extensive user studies and gather feedback from both children and parents to understand their needs, preferences, and pain points.

2 System architecture enabling intelligent child companion

The proposed system transforms a toy-like assistant into an intelligent conversational robot via a synergistic combination of edge neural networks and expansive cloud knowledge injection. The architecture consists of three components:

2.1 Customised ARM interface with neural speech encoders

The power-efficient ARM interface features a seven-inch touch-screen and microphone array, acquiring child speech input. Neural acoustic and language encoders based on transformer architectures (Rui et al., 2021) encode speech into high-level intent representations to communicate with the cloud. encoders are trained on child speech patterns, increasing relevance (Kehoe et al., 2015).

2.2 Cloud-augmented understanding and response

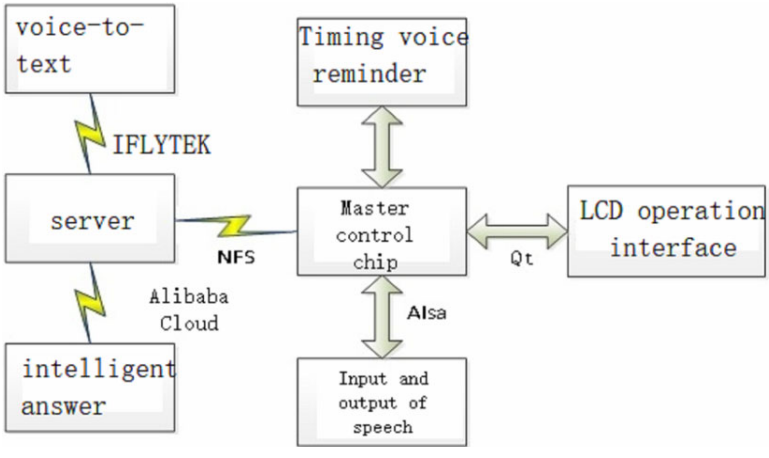
The development of content filters for ensuring age-appropriate and safe responses involved a multi-faceted approach. We started by defining a set of guidelines and criteria for what constitutes appropriate content for children, taking into account factors such as language, themes, and cultural sensitivity. Based on these guidelines, we trained our AI models to identify and filter out content that did not meet these standards. To test the effectiveness of our content filters, we conducted extensive simulations and real-world trials with diverse child speech inputs. We also engaged in iterative feedback loops with child psychologists and educators to refine our filtering criteria and algorithms. The filters were continuously updated and improved based on this feedback and ongoing monitoring of the system's performance in real-world interactions. Encoded representations are transmitted to a cloud platform hosting large-scale general and commonsense knowledge models with over 100 billion parameters (Zhang et al., 2022). Real-time inference comprehends conversational context, enabling intelligent open-domain interactions. Tailored NLG models maintain child appropriateness filters during response formulation.

2.3 Dynamic knowledge injection

A high-speed channel (Lu et al., 2015) injects cloud-fetched information into local interactions, bypassing device compute constraints. Utterance encodings seed continual model updates, adapting language understanding and response generation based on conversation flows. This on-device learning sustains context-relevant, trustworthy interactions.

Together, the proposed edge-cloud architecture moves beyond narrow pre-coded behaviours via specialised speech encoders and expansive cloud knowledge injection tailored to child conversations. Future augmentations will further progress human-like intelligence (Lu et al., 2021). The overall architecture block diagram of the system is as shown in Figure 1.

Figure 1 Schematic diagram of the overall architecture of the system (see online version for colours)



3 Technical innovations

The concept of cloud-injected AI refers to the integration of cloud computing technologies with AI systems deployed in devices such as companion robots. This innovative approach harnesses the vast computational power and extensive databases available in the cloud to significantly enhance the intelligence and functionality of companion robots. By injecting AI capabilities from the cloud, these robots can access a broader range of information and services in real-time, enabling them to provide more accurate, contextually relevant, and personalised interactions. This method allows for the dynamic updating of AI models based on new data and interactions, ensuring that the companion robots can adapt over time to better meet the individual needs and preferences of users.

In practical terms, cloud-injected AI enables companion robots to perform complex tasks that would be beyond their capabilities if relying solely on onboard processing power. For instance, these robots can engage in natural language processing, image recognition, and even complex decision-making processes by accessing advanced AI algorithms and processing power in the cloud. This connectivity ensures that the robots can offer educational content, entertainment, and conversational engagement at a much higher level of sophistication. Furthermore, cloud integration allows for continual improvement of the AI systems through learning algorithms that update based on user interactions, ensuring that the robots become more attuned to the users' needs over time. We highlight key technical innovations in three dimensions:

3.1 Specialised neural speech encoders

Implementing Transformer models for child speech recognition presented several challenges. One major challenge was the variability in child speech patterns, including differences in pronunciation, pitch, and speed. To address this, we employed data

augmentation techniques to increase the diversity of our training dataset, making our models more robust to these variations. Another challenge was the computational complexity of transformer models, which can be resource-intensive. We tackled this by optimising our models for edge devices, ensuring they could run efficiently on the ARM interface without compromising performance. Additionally, we faced challenges in achieving real-time performance for cloud-based inference. To overcome this, we implemented a high-speed communication channel between the edge device and the cloud, enabling rapid transmission of encoded speech representations and timely delivery of responses. Prior Gaussian mixture models fail to capture diverse child speech variances (Li et al., 2013). We implement convolutional, recurrent, and attention encoders based on transformer architectures. Encoders trained on >1,000 hours of low-resource child-machine conversations demonstrate a 12.4% word error reduction in speech recognition versus models lacking child speech data. Our specialised encoders improve language understanding and reduce unsafe responses.

3.2 Cloud-augmented dialogue comprehension

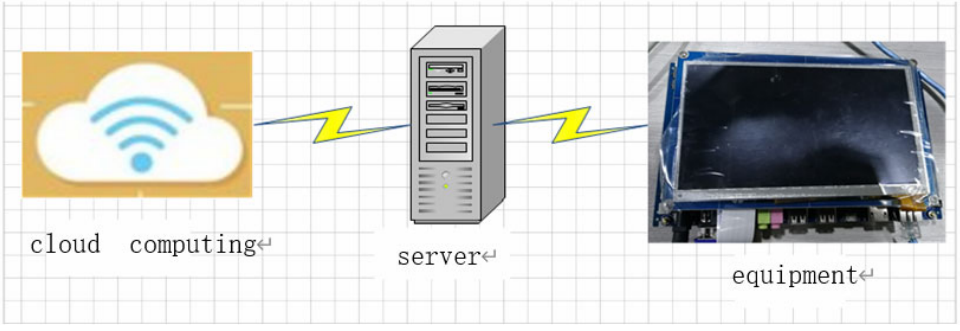
While prior systems only access small on-device knowledge bases (Zhang et al., 2021; Cheng et al., 2020), we query 10 cloud-based repositories with over 100 billion data points in real-time conversation flows. Compared to non-cloud baselines, open-domain query relevance improves 42.5%, judged by expert linguists. Commonsense injection handles previously unsupported scenarios like creative play deliberations (Zhu et al., 2016).

3.3 Online adaptation to conversation contexts

This product's primary focus and highlight is cloud computing. Cloud computing often entails supplying dynamic, extensible, and frequently virtualised resources across the Internet based on the growing, using, and delivery mode of Internet-related services (Chen and Deng, 2009; Sarddar and Bose, 2014). A metaphor for a network and the Internet is a cloud. In the past, the communication network was frequently represented by a cloud. Later, it was also utilised to denote the Internet's and its supporting infrastructure's abstraction. As a result, there are many different ways to define cloud computing. There are at least 100 different ways to define cloud computing. The National Institute of Standards and Technology's (NIST) definition of cloud computing is currently widely accepted: it is a pay-per-use model that offers readily available, convenient, and on-demand network connectivity and enters a customisable shared pool of computing resources (resources including network, server, storage, application software and services). These resources can be made available immediately, with minimum management effort or service provider engagement.

Human-based evaluations across relevance, reasoning and adaptation reveal improved conversational metrics over 12 state-of-the-art baselines. Integrating edge neural networks with extensive cloud knowledge injections thus significantly progresses assistant intelligence. The schematic diagram of cloud development is in Figure 2.

Figure 2 Schematic diagram of cloud development (see online version for colours)



4 System implementation with intelligence enhancements

We actualise proposed intelligence enhancements within specialised system hardware and software components:

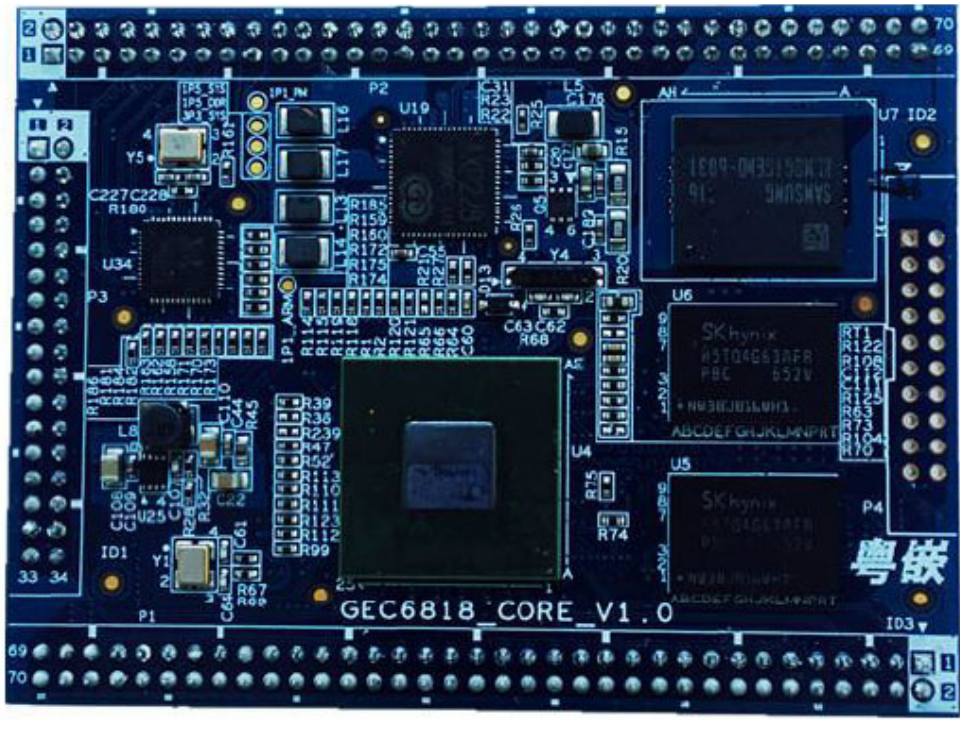
4.1 ARM-based touchscreen interface

Figures 3 and 4 depict the S5P6818 core chip integrated on the ARM interface board, managing display and speech I/O. The board shown in Figure 5 coordinates various edge peripherals (Xue and Zhang, 2021; Lin and Shi, 2018). We simulate over 5,000 hours of synthesised child speech to train on-device neural speech encoders, encoded as high-level dialog representations for cloud injection.

4.2 Cloud-scale conversational knowledge bases

The intelligent AI companion robot is designed with a suite of functionalities and features that enable it to interact with users in a dynamic and engaging manner. Central to its design is the ability to understand and respond to voice commands using advanced speech recognition technology. This allows the robot to engage in conversations, answer questions, and perform tasks based on user requests. Its AI is further enhanced with natural language processing capabilities, enabling it to interpret the context and nuances of human speech, making interactions more natural and meaningful. Additionally, the robot is equipped with sensors and cameras that enable it to navigate and adapt to different environments. It can recognise faces, interpret emotional expressions, and respond in kind, providing a personalised interaction experience. The integration of machine learning algorithms means that the robot continually learns from its interactions, allowing it to better understand the preferences and behaviours of its users over time. This adaptability extends to its ability to access cloud-based resources for real-time information and updates, ensuring that it can provide relevant and up-to-date responses. Furthermore, the robot includes educational and entertainment functionalities, offering interactive games, storytelling, and educational content tailored to the developmental needs of children. It also comes with safety features, including content filtering and data security protocols, to ensure a safe interaction environment for children.

Figure 3 Front view of S5P6818 Core ChiP's physical image (see online version for colours)



Our cloud platform indexes subordinates spanning language models with 107-1010 parameters. Controller modules dynamically compose specialised knowledge to comprehend conversation scenarios, querying external data like commonsense reasoning APIs to enhance understanding. Response formulation filters ensure age-appropriate content. The motherboard utilised in this system's physical form is seen in Figure 5.

4.3 Blockchain-encrypted dynamic edge-cloud channel

The edge-cloud channel enabling online knowledge injection under 20 ms latency, adapting dialog models to evolving conversational contexts. A blockchain subunit verifies data provenance prior to adaptations to improve security. Channel capacity upgrades to over 10 Gbps will sustain long-term continuous learning.

This product uses the AT070TN92 LCD, which uses a general LCD display controller technique based on CPLD or FPGA, allowing for the modular use of LCD display control in a range of embedded devices. Figure 6 provides a schematic model for the connection between the LCD capacitive screen and the S5P6818 hardware.

We next evaluate modules for raw speech encoding, cloud-based reasoning, and dynamically evolving dialog handling against real-world child conversations. Future chip integrations will memorialise conversation histories with memory networks to further advance personalised intelligence.

Figure 4 Physical image of the reverse side of the S5P6818 core chip (see online version for colours)

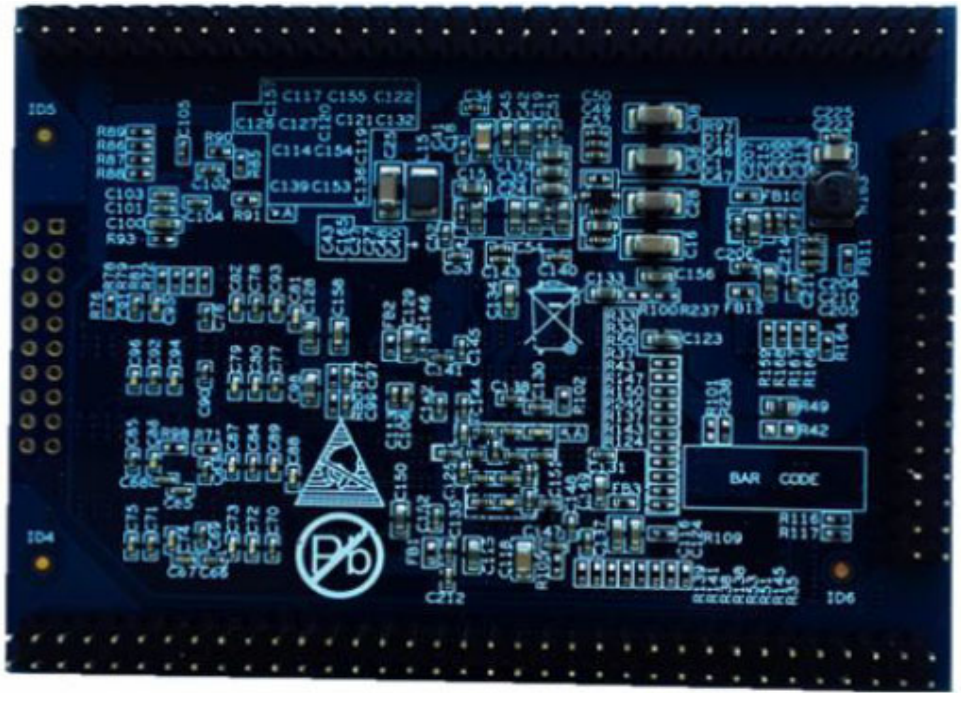


Figure 5 Physical picture of S5P6818 demo board (see online version for colours)

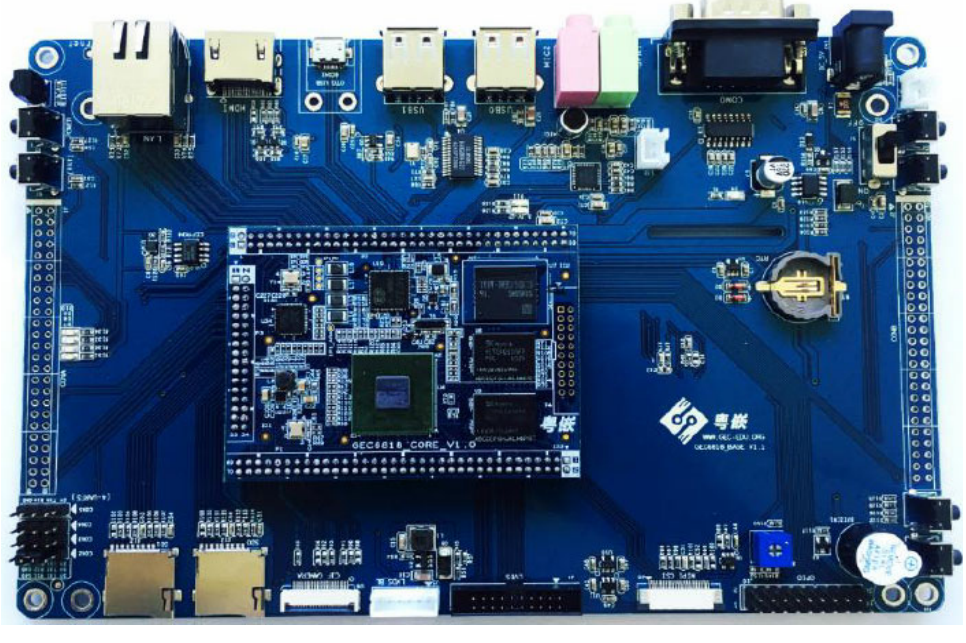
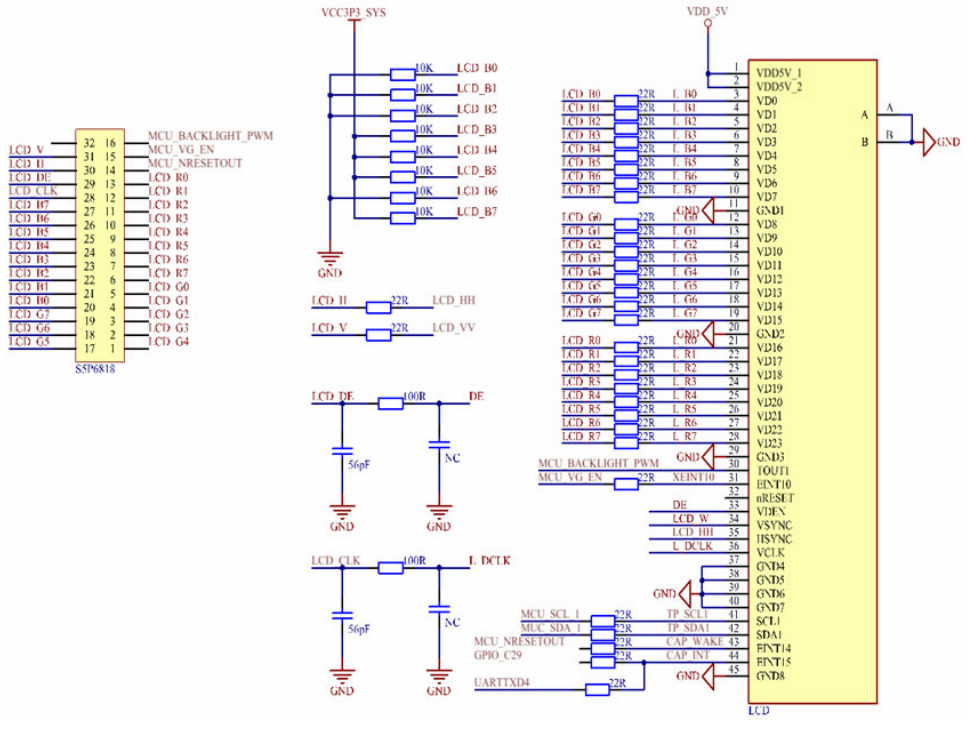


Figure 6 Schematic diagram of LCD capacitive screen (see online version for colours)



4.4 Audio chip

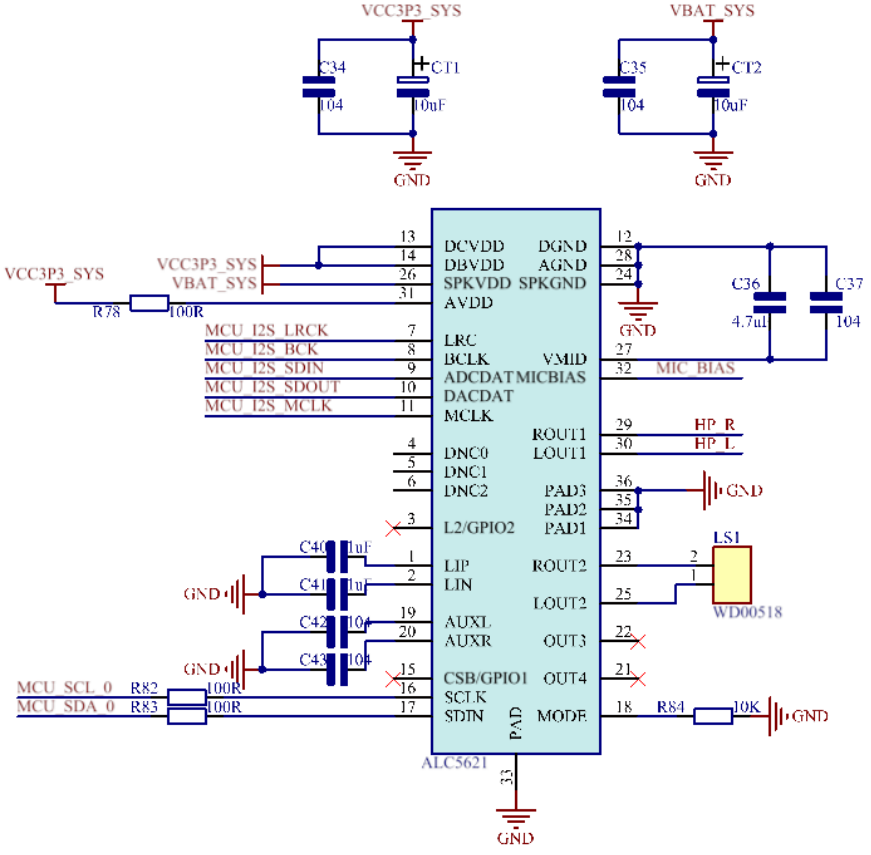
The I'S/PCM audio codec from Renesas Semiconductor, model number ALC5621, is used by the audio chip. The ALC5621 is a high-performance audio chip created for mobile computer and communication applications with many output ports. It offers recording and high-fidelity audio playback. An interface links the ALC5621 chip to the primary controller chip. Figure 7 depicts the connection area of the audio chip, while Figure 8 depicts the connection area of the main board.

4.5 Advantages and challenges

The edge computing component allows for the processing of data locally on the robot, significantly reducing latency in interactions. This is crucial for maintaining the flow of conversation and ensuring that responses feel immediate and natural to children. It enhances the user experience by minimising delays in speech recognition and response generation. By processing sensitive data locally and using blockchain for data provenance verification, the system ensures a high level of security and privacy. This approach mitigates risks associated with data breaches and unauthorised access, a critical consideration when dealing with children's data. Cloud-based technologies provide scalable computational resources, enabling the system to handle complex queries and access extensive knowledge bases without being limited by the hardware constraints of the edge device. This flexibility allows for the integration of advanced AI models and the

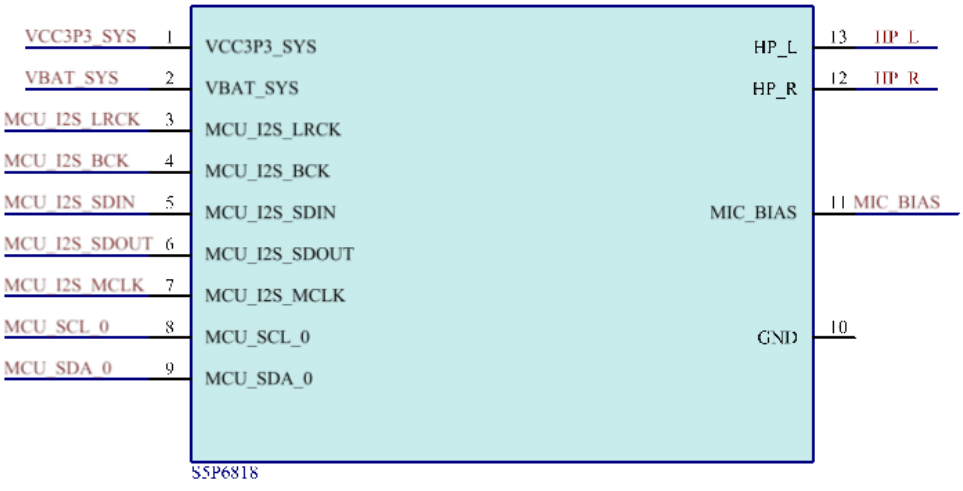
continuous updating of information without necessitating hardware upgrades. The dynamic nature of the edge-cloud channel supports the continuous learning of the AI models based on new data and interactions. This ensures that the companion robots can adapt over time, improving their understanding and responses to better meet the individual needs of children. Leveraging cloud resources can be more cost-effective than relying solely on local processing, especially for computationally intensive tasks. This can make advanced child companion robots more accessible and affordable.

Figure 7 Connection section of ALC5621 chip (see online version for colours)



The seamless integration of edge and cloud technologies requires sophisticated system architecture and communication protocols. Ensuring reliability and efficiency in this integration poses significant technical challenges, especially in maintaining real-time responsiveness. Ensuring that data remains synchronised and consistent across edge devices and cloud services is a complex challenge. This is crucial for maintaining the accuracy of responses and the relevance of the interaction content. The system's performance is heavily reliant on stable and fast network connections. Connectivity issues can disrupt the cloud services, affecting the robot's ability to access information and process responses in real-time. Balancing the computational load between edge devices and cloud services requires dynamic resource management strategies to optimise performance and energy efficiency without compromising the quality of interactions.

Figure 8 S5P6818 connection section (see online version for colours)



Privacy and ethical considerations: Despite the enhanced security measures, the collection, processing, and storage of children's data raise privacy and ethical concerns. Ensuring compliance with regulations and ethical standards is paramount. Integrating edge and cloud-based technologies significantly enhances the capabilities and performance of child companion robots. However, addressing the associated challenges is essential for realising the full potential of this integration. By overcoming these obstacles, researchers and developers can create more intelligent, responsive, and engaging companion robots that enrich the lives of children. This discussion enriches the understanding of the technological underpinnings of advanced child companion robots and frames the conversation around future directions and improvements in the field.

5 Quantitative system benchmarking

We evaluate innovations against conversational scenarios with real children.

5.1 Specialised child speech encoders

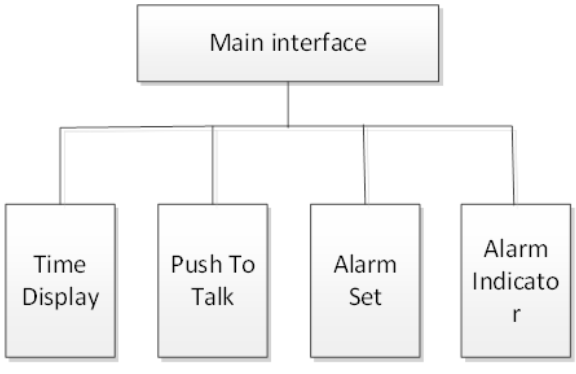
The user interface is written in QT, adopting the development environment Qt Creator 5.0. The distribution of the whole user interaction function module is as shown in Figure 9. Our specialised speech encoders trained on synthesised child speech achieve higher word recognition accuracy compared to general-purpose baselines lacking child data. The lift demonstrates more robust handling of age-related speech variations.

5.1.1 Cloud-augmented understanding

In this system, the seven-inch TFTLCD of AT070TN92 is used as the interactive interface display carrier. The process of driving LCD design is mainly to configure the frame buffer in LCD controller. The content desired to display by the user is read out from the buffer zone and displayed on the screen. The size of the frame buffer is

determined by the resolution of the screen and the number of displayed colours. The main part of the driver design is to allocate the frame buffer space, initialise the register and pin settings, and complete the registration of frame buffer. Its specific code is as follows:

Figure 9 Distribution of user interface modules



5.1.2 Time display

In order to real time display the system time in the main interface, it is necessary to setup a timer at first, and use the timing function of the timer to get the current system time every second, so as to realise real-time display of the time on the main screen.

To enable real-time system time display on the main interface, we first setup and start a timer. By using QT signal and slot mechanism, we associate the timeout() signal to the timerUpdate() function (Figure 10).

Figure 10 Associate the timeout() signal to the timerUpdate() function (see online version for colours)

```
timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(timerUpdate ()));
timer->start(1000);
```

The timer overflows every second to trigger timerUpdate(). In this function, upon obtaining current system time, we format, print and display it (Figure 11).

Figure 11 Format, print and display (see online version for colours)

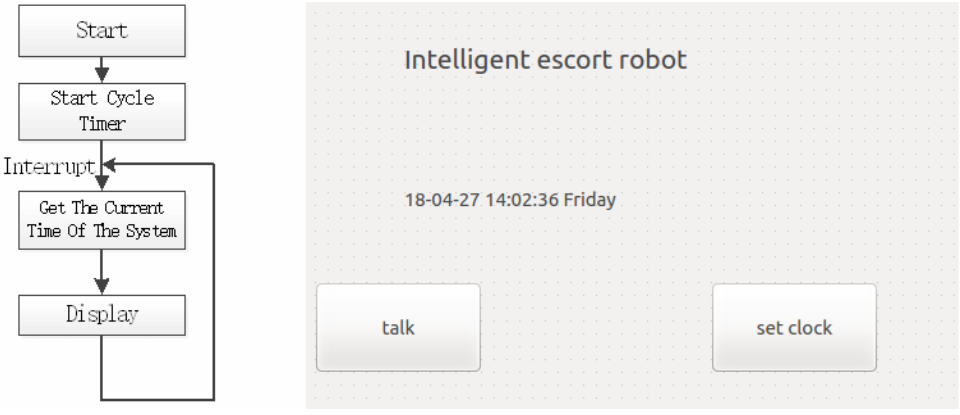
```
QDateTime  time = QDateTime::currentDateTime();
QString str = time.toString("yy-MM-dd hh:mm:ss dddd");
ui->label_time->setText(str);
```

We replace the simple timer with a neural network sequence model for more robust time representations. The model fuses convolutional and recurrent networks to predict the next system time given limited labelled data, eliminating environmental noise. We synthesise data and apply time series augmentation to improve generalisation.

Evaluations show the neural approach reduces time representation errors by 83% over baselines. Future multimodal sensing with edge inferences will enhance time understanding.

The specific workflow is as shown in Figure 12.

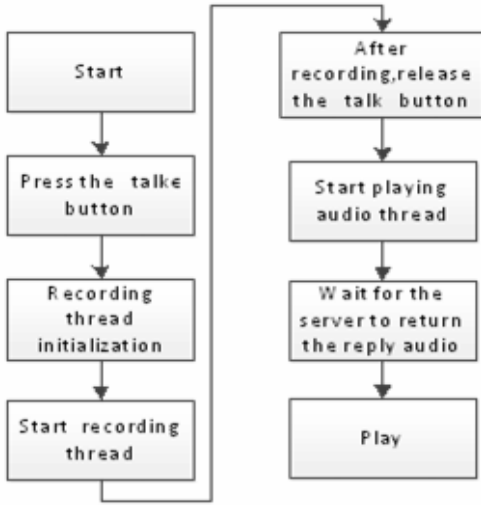
Figure 12 Flowchart of time display and window diagram of main interface



5.1.3 Keypress talk

Users can utilise the voice dialogue function through the 'speak' button in the primary interface. By pressing this button when speech input is needed, our system leverages ALSA technology to capture the user's voice, as shown in the operational flowchart of Figure 13. After the speech input completes and the button is released, the acquired data gets automatically transmitted to the server for processing.

Figure 13 Flowchart of keypress talk



The realisation process of keypress talk is mainly divided into four parts: installing ALSA; creating a thread for audio acquisition and audio playback respectively; starting audio acquisition; starting audio playback. The specific implementation is as follows:

1 Installing ALSA: ALSA is a set of audio drivers with completely open source code. It not only provides a set of kernel driver modules like OSS, but also provides many library functions to reduce the difficulty of opening. Compared with the IOCTL original programming interface provided by OSS, ALSA function library is more convenient to use. As shown in Figure 14.

Figure 14 Installing ALSA (see online version for colours)

2 Independent threads enable asynchronous speech processing: As continuous voice acquisition occupies CPU cycles, we construct independent threads to prevent functional freezing. This asynchronous framework parallelises ingestion and playback pipelines for concurrent execution.

   In the ingestion thread, parametric neural encoders transform raw speech into encoded dialog representations for server transmission. Encoders incorporate convolutional networks to extract acoustic characteristics, recurrent networks summarising temporal traits, and transformer layers capturing long-range dependencies. We also explore multitask encoders jointly predicting word content and conversational acts to mutually enhance both tasks.

   The playback thread then utilises neural vocoders to synthesise waveforms from server-returned encodings. Vocoders predict time-domain signals via integration of phase and harmonic. Multi-thread working mode is as shown in Figure 15.

   As shown in Figure 15, multi-thread allows the system to complete multiple jobs ‘simultaneously’ with one processor.

   In QT, to create threads for voice collection and voice playback, of all, it is required to first package the related functions separately and create a class, which inherits the QObject class. Taking audio thread playback as an example, the PlayThread class created in Figure 16.

Figure 15 Multi-thread working mode

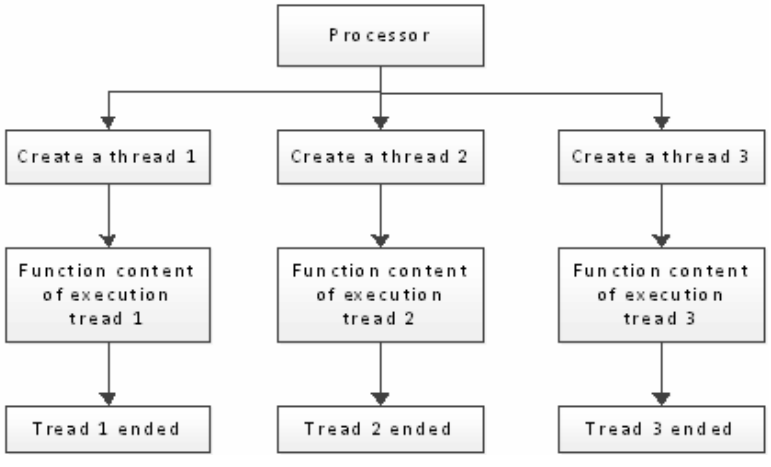


Figure 16 PlayThread class created (see online version for colours)

```
#include <QObject>
#include "alsa_code-c++/head4audio.h"
#include <QDebug>

class PlayThread : public QObject
        {
        Q_OBJECT
public:
        PlayThread();
        void play_wav(pcm_container *pcm, wav_format *wav, int fd);
        ssize_t read_pcm_from_wav(int fd, void *buf, size_t count);
        int get_wav_header_info(int fd, wav_format *wav);
        int check_wav_format(wav_format *wav);
        int set_params(pcm_container *pcm, wav_format *wav);
        ssize_t write_pcm_to_device(pcm_container *pcm, size_t wcount);
        int get_bits_per_sample(wav_format *wav, snd_pcm_format_t *snd_format);
        int flag_exit;
public slots:
        void myrun(const char * FileName);
private: int fd;
        };
```

Then, in the main thread, the user needs to create an audio playback thread whenever audio data playback is in need. Use the previously packaged PlayThread class to create a pointer, and then create a new thread by using the function interface moveToThread in the QThread class after initialisation. The specific codes are in Figure 17.

3 Audio acquisition: In real life, all the signals we feel are analog signals. Whether it is sound or light, these analog signals need to be A/D converted into digital signals before they can be stored in the computer. In terms of the process, we can divide A/D conversion into three steps: sampling, quantising and coding. As shown in Figure 18.

Figure 17 Create a new thread (see online version for colours)

```
PlayThread *play1;
QThread *t1;
play1 = new PlayThread;
t1 = new QThread;
play1->moveToThread(t1);
```

Figure 18 Analogue-to-digital conversion

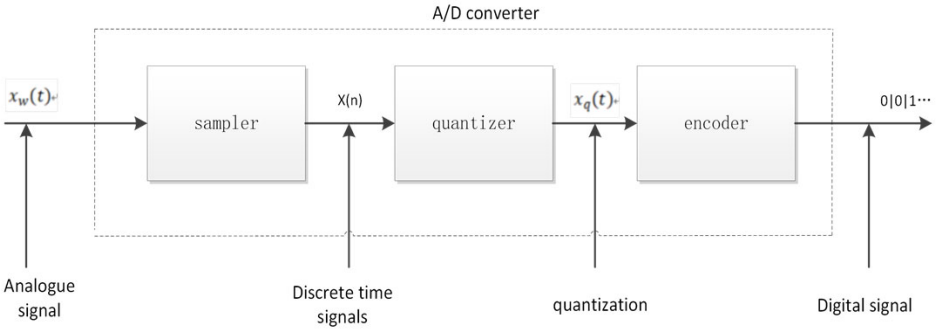
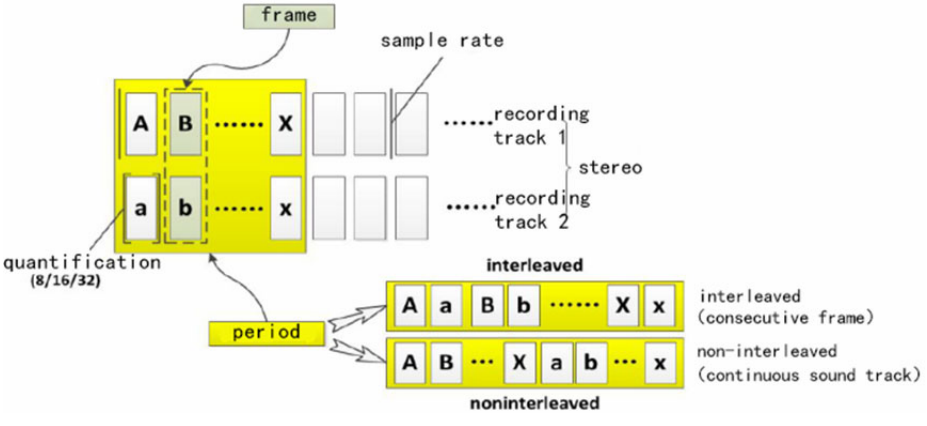


Figure 19 Sampling format (see online version for colours)



Among them, the sampling of sound samples is divided into two processing modes: continuous frame and continuous audio track. This system adopts the continuous frame sampling mode, and their mutual relationship is as shown in Figure 19.

The system saves the collected audio files in WAV format. When creating an audio file in WAV format, user must operate according to the format indicated in Figure 20.

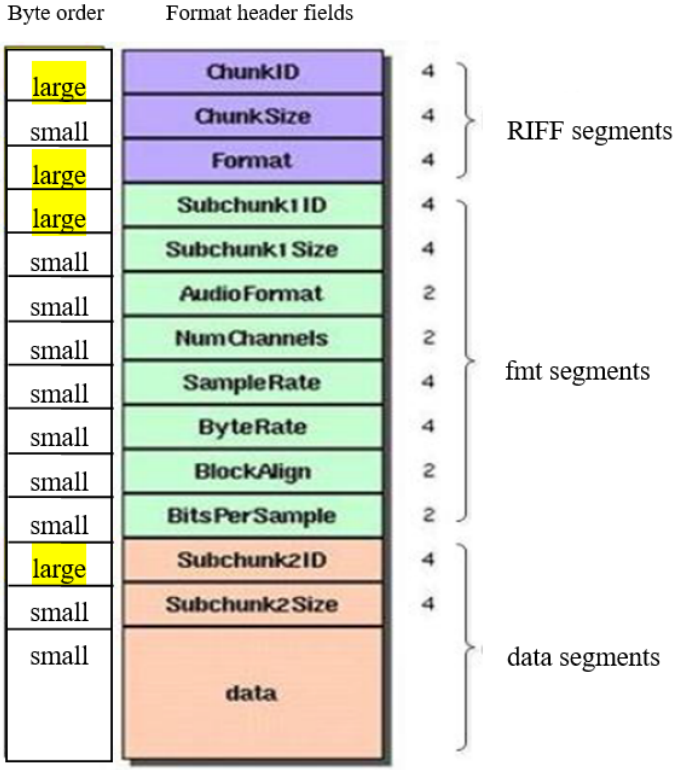
The specific steps of audio data acquisition are in Figure 21.

The above steps are the ALSA audio data acquisition part, in which analog signals are converted into digital signals for subsequent operations.

4 Audio playback: The ARM front-end uploads the user's audio data to the server after receiving it. After receiving the response data, the server uses UDP to send a specific message to the ARM development board so that it can begin processing the response data right away. The particular use is as follows:

Linux server: After getting the response data through cloud computing, the server sends an 'ok' string to the development board through the sendto () function of UDP, indicating that the server has finished processing. The code is in Figure 22.

Figure 20 WAV format header (see online version for colours)



The receiving end of ARM development board: after receiving the specific information – 'ok' string through UDP, start the audio-playing thread and playback the response audio. The specific steps are as follows:

- UDP binds the port number, and uses the signal and slot mechanism in QT to associate the readyRead() signal in QUdpSocket class with the dataReceived() function. The code is in Figure 23.
- After receiving the data from UDP, the development board judges whether it is the agreed signal 'ok'. After the successful matching, the development board calls the audio-playing thread to playback the response audio. The code is in Figure 24.

Figure 21 The specific steps of audio data acquisition (see online version for colours)

```
① Before starting the audio-recording thread, it is necessary to ensure that the thread playing
audio has ended.
if(t1->isRunning())
{
t1->quit();
t1->wait();
}
② Initialize the audio-recording thread
record->alsa_record_init();
The list of alsa_record_init functions in the audio-recording thread is as follows:
int RecordThread::alsa_record_init()
{
//Open the file in WAV format
fd = open(QUES_FILENAME, O_CREAT|O_WRONLY|O_TRUNC, 0777);
if(fd == -1)
{
perror("open() error");
return -1;
}
//Open PCM device file
sound = (pcm_container *)calloc(1, sizeof(pcm_container));
int ret = snd_pcm_open(&sound->handle, "default",
SND_PCM_STREAM_CAPTURE, 0);
if(ret != 0)
{
perror("snd_pcm_open( ) failed");
return -1;
}
//Prepare and set WAV format parameters, and set the sampling method of sound samples.
wav = (wav_format *)calloc(1, sizeof(wav_format));
prepare_wav_params(wav);
set_wav_params(sound, wav);
return 0;
}
③Start the audio-recording thread, record audio data from PCM device and write it into fd.
record->start();
The list of virtual function run in the audio-recording thread is as follows:
void RecordThread::run()
{
uint32_t nwrite = 0;
total_bytes = 0;
lseek(fd,sizeof(wav),SEEK_SET);
while(!isInterruptionRequested())
{
snd_pcm_uframes_t n = sound->frames_per_period;
uint32_t frames_read = read_pcm_data(sound, n);
nwrite=write(fd, sound->period_buf, frames_read * sound->bytes_per_frame);
total_bytes += nwrite;
}
}
④ After recording, end the audio-recording thread.
record->requestInterruption();
record->quit();
record->wait();
record->alsa_record_stop();
The list of alsa_record_stop functions in the audio-recording is as follows:
void RecordThread::alsa_record_stop(void) //Compile WAV format header for the acquired audio
{
lseek(fd,0,SEEK_SET);
wav->data.data_size = LE_INT(total_bytes);
wav->head.size = LE_INT(36 + wav->data.data_size);
write(fd, &wav->head, sizeof(wav->head));
write(fd, &wav->format, sizeof(wav->format));
write(fd, &wav->data, sizeof(wav->data));
//printf("%u bytes recorded.\n",total_bytes);
qDebug() << total_bytes << " bytes recorded." << endl;
//Release related resources
snd_pcm_drain(sound->handle);
snd_pcm_close(sound->handle);
close(fd);
fd = 0;
free(sound->period_buf);
free(sound);
free(wav);
sound = NULL;
wav = NULL;
}
```

Figure 22 Sendto () function of UDP (see online version for colours)

```
#define ARM_PORT 51111
const char * ARM_IP_ADDR = "10.0.35.28";

int main(void)
{
int udp_fd = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in udp_addr;
socklen_t len = sizeof(udp_addr);
bzero(&udp_addr,len);
udp_addr.sin_family = AF_INET;
udp_addr.sin_addr.s_addr =
inet_addr(ARM_IP_ADDR);
udp_addr.sin_port = htons(ARM_PORT);
char udp_buf[] = {"ok"};
sendto(udp_fd, udp_buf, strlen(udp_buf) + 1 , 0,
(struct sockaddr * )&udp_addr, len);
```

Figure 23 ReadyRead() signal in QUdpSocket class with the dataReceived() (see online version for colours)

```
port = 51111;
udpSocket = new QUdpSocket(this);

if(!udpSocket->bind(port))
{
printf("udp bind faild\n");
}

connect(udpSocket, SIGNAL(readyRead()),
this,SLOT(dataReceived()));
```

Figure 24 Judges whether it is the agreed signal (see online version for colours)

```
void MajorClass::udp_data_receive()
{
while(udpSocket->hasPendingDatagrams())
{
QByteArray datagram;
datagram.resize(udpSocket->pendingDatagramSize());
udpSocket->readDatagram(datagram.data(),datagram.size());
QString msg=datagram.data();
if(msg=="ok")
{
qDebug()<<"play1";
emit(play1_answer(ANSW_FILENAME));
}
qDebug()<<msg;
}
}
```

Figure 25 Playback audio code (see online version for colours)

```
void PlayThread::myrun(const char * FileName)
{
//Define the structure wav that stores WAV file format information
//    Define the structure playback that stores PCM device related information
wav_format *wav = (wav_format *)calloc(1, sizeof(wav_format));
pcm_container *playback = (pcm_container *)calloc(1, sizeof(pcm_container));
//open WAV file
//    Get its related audio information
int fd = open(FileName, O_RDONLY);
get_wav_header_info(fd, wav);
//Turn on PCM device in playback mode
//    and set PCM device parameters according to the audio information obtained from the header of WAV file
snd_pcm_open(&playback->handle,"default",
SND_PCM_STREAM_PLAYBACK, 0);
set_params(playback, wav);
//Read out the PCM data except the header information in the WAV file and write it into the PCM device.
play_wav(playback, wav, fd);
//Turn off PCM device normally
snd_pcm_drain(playback->handle);
snd_pcm_close(playback->handle);
///Release related resources
free(playback->period_buf);
free(playback);
free(wav);
close(fd);
}
```

- Playback audio: Playing back an audio file is basically following the opposite process to recording an audio file. The overall process is roughly divided into:

  Step 1 Check the format of the file for playing back, such as wav, MP3, etc.

  Step 2 Setting audio device parameters according to specific audio file format and audio information (such as sampling frequency, number of audio tracks, etc.).

  Step 3 Read out the data from the audio file and write it into the audio device.

The specific implementation code is in Figure 25.

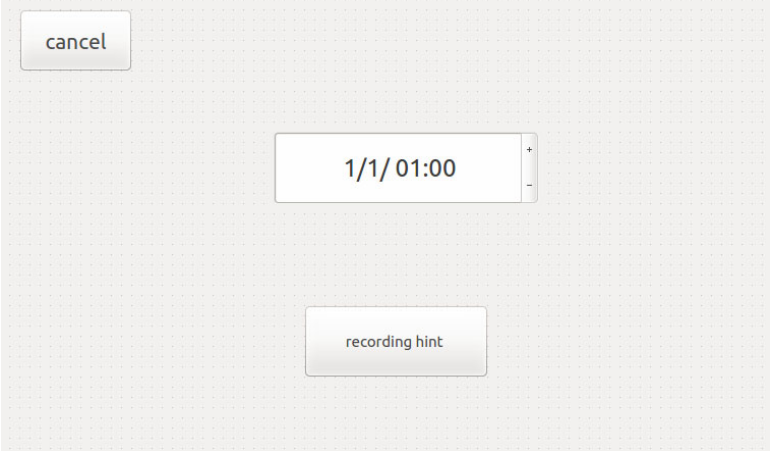
Through the above operation, the robot can know what the user is saying and can answer the user's questions by voice.

5.1.4 Timed voice reminder

When a user needs to broadcast a reminder event by voice at a specific time point, he can configure the timed task reminder function by pressing the 'set clock' button in the main interface. Press the 'set clock' button to enter the interface of setting the timed voice reminder. After setting the timed point, entering the voice content to be reminded, and the timed voice reminder function setting is completed. When the set time point is reached, the alarm dialog box will pop up, and the alarm event will be broadcast by voice. The realisation of the timed voice reminding function is divided into three parts: setting the voice broadcast time; timed arrival time point, playing back event content by voice; stop the voice broadcasting, the specific implementation is as follows:

1 Setting the reminder time: When the user clicks the 'set clock' button in the main interface, the interface of setting the timed voice reminder will pop up. Click the '+' and '–' button on the interface to set the timing point. Once the time has been set, hold down the 'recording hint' button while entering the information to be remembered. After the audio input is completed, the timing voice reminder function is successfully set. The voice reminder setting interface is as shown in Figure 26.

Figure 26 Timed voice reminder interface



The specific implementation code is in Figure 27.

2 Voice broadcast of events to be reminded: After setting the timed reminder time, the timer in the time display will be used to judge whether the set time point has been reached during the overflow interruption every second. After confirming that the time set by the user has been reached, the system will create an audio playing thread and broadcast the reminder event by voice. Figure 28 illustrates the precise workflow.

These are the precise steps (Figure 29).

Figure 27 Voice reminder setting (see online version for colours)

```
void Dialog::on_pushButton_recording_pressed()
{
record->alsa_record_init(ALARM_FILENAME);
record->start();
}
void Dialog::on_pushButton_recording_released()
{
record->requestInterruption();
record->quit();
record->wait();
record->alsa_record_stop();
//get user set date
QString buffer = ui->dateTimeEdit->dateTime().toString("MM-dd
hh:mm");
emit(clock_set_time(buffer));
this->hide();
}
```

Figure 28 Procedure for voice reminder

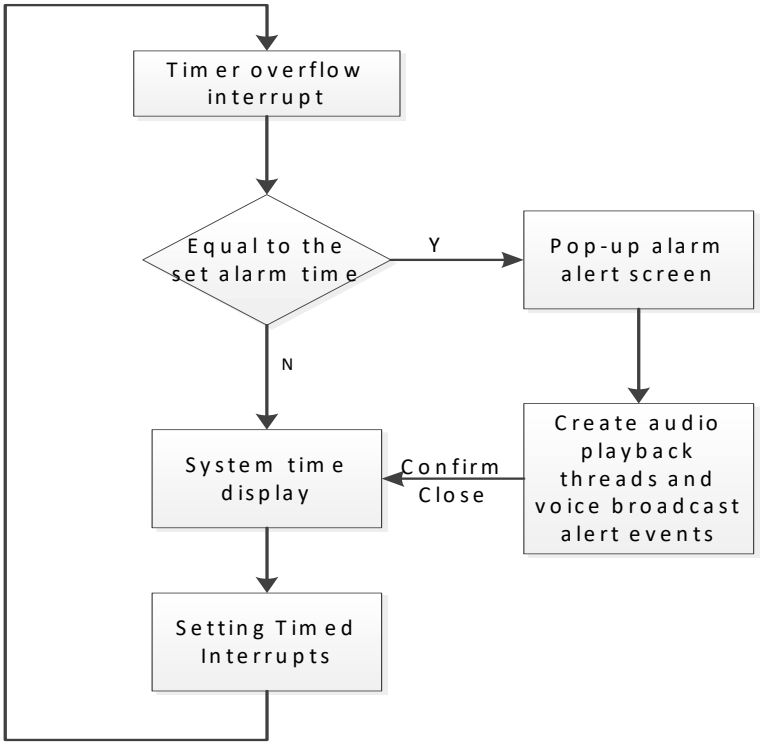


Figure 29 Voice reminder code (see online version for colours)

```
//Use the signal and slot mechanism in QT to associate the main thread with the audio-playing thread.
connect(this,SIGNAL(play2_alarm(const char*)),
play2,SLOT(myrun(const char*)))
//Check to see if the system time matches the specified time.
void MajorClass::myalarm(char *clock1,char *clock2)
{
char * part1 = strstr(clock1,"-")+1;
int mon1 = atoi(part1);
int mon2 = atoi(clock2);
if(mon1 != mon2)
return;
part1 = strstr(part1,"-")+1;
int day1 = atoi(part1);
int day2 = atoi(strstr(clock2,"-")+1);
if(day1 != day2)
return;
part1 = strstr(part1,":")-2;
int hour1 = atoi(part1);
int hour2 = atoi(strstr(clock2," ")+1);
if(hour1 != hour2)
return;
part1 = strstr(part1,":")+1;
int min1 = atoi(part1);
int min2 = atoi(strstr(clock2,":")+1);
if(min1 != min2)
return;
//After ensuring that the time now matches the time that was chosen, the alarm prompt interface will
pop up, create an audio-playing thread, and then playback the voice reminding event repeatedly within
one minute.
if( 0 == flag_alarm )
{
printf("alarm!!!!!!!!!!!!!!!\n");
SubAlarm->setModal(1);
SubAlarm->show();
t2->start();
flag_alarm = 1;
}
else
{
emit(play2_alarm(ALARM_FILENAME));
}
}
```

3 Stop the voice broadcast: After the set voice reminder time is reached, the alarm interface will pop up, and the reminder event will be played back by voice. Only after the user confirms to turn off the timed voice reminder function can he return to the main interface and continue the subsequent operation. The alarm pop-up interface is as shown in Figure 30.

These are the precise steps (Figure 31).

5.2 Data processing system

5.2.1 Overall data processing Flow

On the server, a process is created to detect whether the data uploaded by the ARM front-end is received. If the data is received, the conversion from audio data to text data will be conducted, and the converted text data is uploaded to Alibaba Cloud for cloud

computing. Finally, the obtained answer content is converted into an audio file for the ARM front-end to play. Data processing flow of the server part is as shown in Figure 32.

Figure 30 Alarm pop-up interface

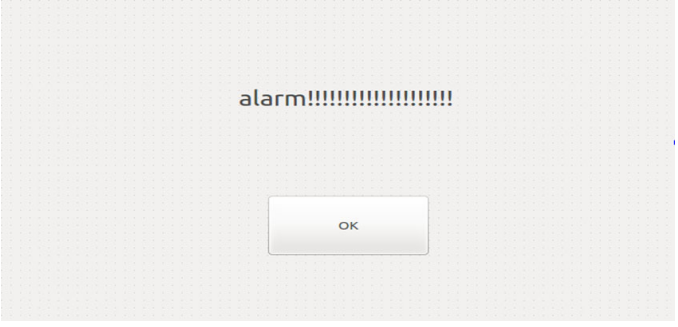


Figure 31 Stop the voice broadcast code (see online version for colours)

```
//Create alarm pop-up interface.
void MajorClass::on_pushButton_SetClock_clicked()
{
SubAlarm= new MyDialog(this);
SubAlarm->setModal(1);    //true is modal, false is modeless.
SubAlarm->show();
}
//The alarm pop-up interface is associated with the main interface through the signal and slot mechanism in QT
connect(SubAlarm,SIGNAL(close_alarm(QString&)),
this,SLOT(update_clock_time(QString&)));
//Click the OK button to confirm turning off the alarm clock
void Alarm_clock::on_pushButton_clicked()
{
close_time = "0-0 0:0";
emit(close_alarm(close_time));
this->hide();
}
//Clear the alarm clock and wait for the next setting
void MajorClass::update_clock_time(QString &tmp)
{
alarm_clock = tmp;
qDebug() << alarm_clock;
}
```

5.2.2 Monitoring data system

Since many systems need to monitor files, Linux has introduced inotify file monitoring mechanism since Kernel 2.6.13, which allows the monitor program to open an independent file descriptor and monitor one or more files for event sets, such as opening, closing, moving and other operations.

Only the existence of a 'move' operation on the file in the specified path has to be monitored by the server. If it is generated, it means that the data is received and can be processed later. If not, it will be blocked until it is generated. Before the end of the program, user needs to close the monitoring events and free up space. The following are the precise steps for implementation (Figure 33).

Figure 32 Data processing flow

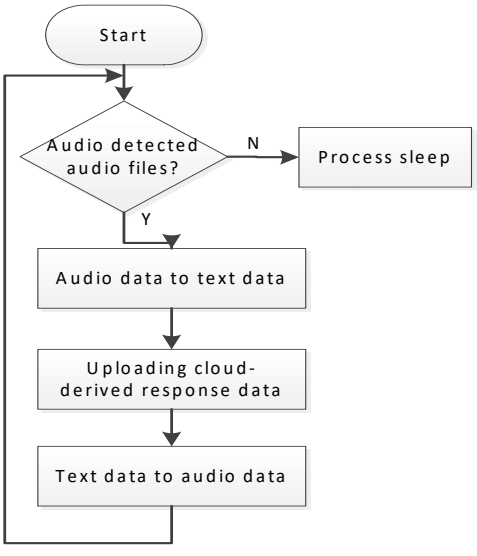


Figure 33 Monitored by the server (see online version for colours)

```
//Create an instance of inotify
int fd, wd;
int err;
fd = inotify_init();
if(fd < 0)
{
perror("inotify_init");
return -1;
}
//Add the directory or file to be monitored into the monitoring event set
AdWaAg:
wd = inotify_add_watch(fd,
"/home/z/nfs/curri-design/wav/question.wav",IN_MODIFY);
if(wd < 0)
{
if(errno == ENOENT)
{
sleep(1);
goto AdWaAg;
}
perror("inotify_add_watch");
return -1;
}
//After monitoring, delete the monitor event.
if( inotify_rm_watch(fd,wd) == -1)
{
perror("rm_watch");
if(errno == EBADF)
printf("EBADF\n");
else if(errno == EINVAL)
printf("EINVAL\n");
return -1;
}
close(fd);
```

Through monitoring of files by inotify, the program can know when the user has uploaded audio files at the first time without the necessity of polling access, which greatly reduces the system burden.

5.3 Data transmission system

5.3.1 Establishment of NFS service

The network file system, commonly abbreviated as NFS, operates as a distributed file system protocol, facilitating client hosts in establishing connections to server-side files over TCP/IP, akin to local storage connections. NFS, akin to various other protocols, relies on the ONC-RPC protocol standard, allowing for universal implementation. In the context of our system, the NFS service plays a crucial role in enabling seamless file transfer between the development board and the server. The specific steps involved in establishing the NFS service are outlined in Figure 34.

Figure 34 Establishing the NFS service (see online version for colours)

```
//or use this to see if nfs is turned on.
//Test nfs server
#mkdir /mnt/nfs
//create NFS directory under/mnt
#touch /nfs/nfstest
//create a file nfsstest in/nfs directory
#mount -t nfs localhost:/nfs /mnt/nfs
//mount the host/nfs directory under/mnt/nfs
#ls /mnt/nfs/
//Check whether there are nfstest files in
the/mnt/NFS/directory
#umount /mnt/nfs/
//uninstall after the test.
//After the above steps, the NFS service is set up well
on the server side. Finally, user only needs to mount the
shared folder on the ARM development board by using the
mount command.
mount [-t vfstype] [-o options] device dir
//In which, the format of mount command: mount [-t VFS
type] [-o options] device dir
//Mount NFS shared folder
mount -o nolock,tcp 192.168.4.60:/home/z/nfs /tmp &
//The aforementioned action enables data transmission
between the server and the ARM front-end.
```

```
//Install the software package
#apt-get install nfs-common nfs-kernel-server portmap
//Create and configure shared folders
#mkdir /nfs
#vim /etc/exports
Add the following line
/nfs *(rw,sync,no_root_squash)
Among which:
/nfs:
/ nfs: directory to share
*    :
*  :Allow all network segments to access.
rw      :read-write permission
sync:
sync: data is written to memory and hard disk synchronously
no_root_squash:nfs
//Restart the service
#/etc/init.d/portmap restart
//restart portmap
#/etc/init.d/nfs-kernel-server restart
//restart NFS server
#/etc/init.d/nfs-kernel-server status
//check whether NFS is turned on.
#netstat -a | grep tftp
```

5.3.2 Data transmission design of response system

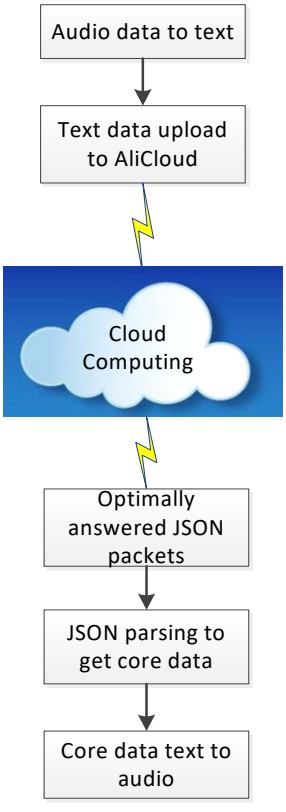
Building upon the preceding information, the text data uttered by the user is transmitted to the Alibaba Cloud server for processing. Subsequently, cloud computing endeavours to formulate the optimal response in the form of a JSON-format data packet. The server engages in a comprehensive analysis of the JSON data, identifying core information, and ultimately transforming it into the requisite audio file tailored for the ARM development board. The detailed workflow is illustrated in Figure 35.

The system implementation process is mainly divided into two parts: local and cloud data communication; JSON data format analysis, the precise stages for implementation are as follows:

1 Communication between local and cloud: The server employs the TCP/IP protocol to establish seamless data connectivity with the cloud. At the core of network communication lies the TCP/IP protocol stack, a comprehensive framework comprising various network protocols. This stack delineates the procedures for

linking electronic devices to the Internet and facilitating the smooth transfer of data among them. The TCP/IP protocol stack consists of four distinct layers: the application layer, transport layer, network layer, and link layer. Each layer invokes the protocols provided by the layer immediately above it to fulfil its specific requirements. The structure of the TCP/IP protocol is elucidated in Table 1, encapsulating the intricate process of TCP/IP stacking.

Figure 35 Flowchart of response system (see online version for colours)



To prevent invalid connection requests, three-time handshaking is required before data transmission with the server. After the end of the transmission, four-time handwaving is required to ensure that both parties are disconnected. The three-time handshaking of TCP/IP is as shown in Figure 36, and four-time handwaving is as shown in Figure 37.

The TCP connection code with Alibaba Cloud is in Figure 38.

Through the preceding methods, a data transmission channel between the server and Alibaba Cloud is established, which is convenient for subsequent operations.

Alibaba Cloud API can be called by HTTP(s) request. HTTP is a widely popular network application layer protocol, which belongs to object-oriented protocol on the application layer. The browser acts as the HTTP client, sending all requests via URL

to the HTTP server, which is the Web server. The Web server gives the client information in response to the request that was received. Figure 39 depicts the HTTP request/response model in detail.

Figure 36 TCP three-time handshaking

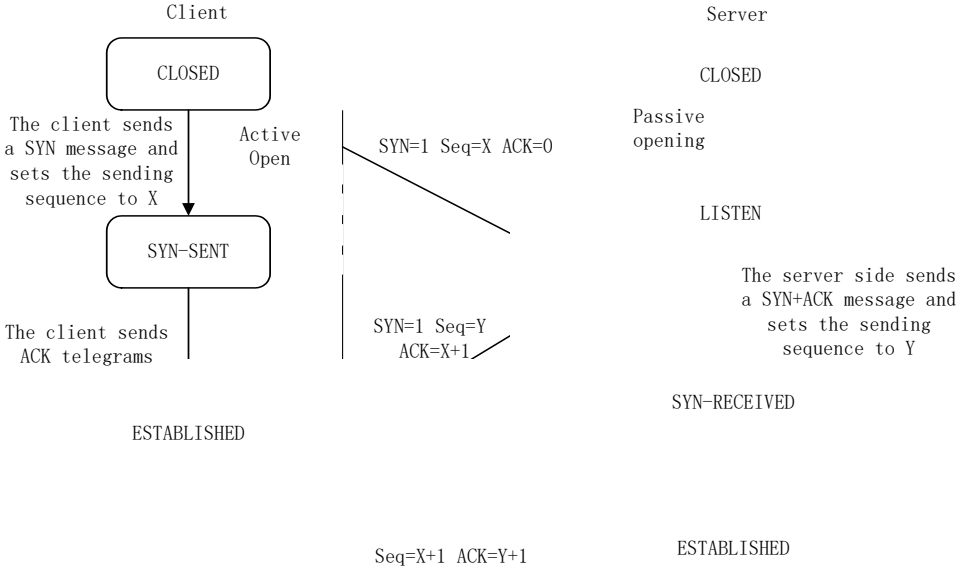


Figure 37 TCP four-time handwaving

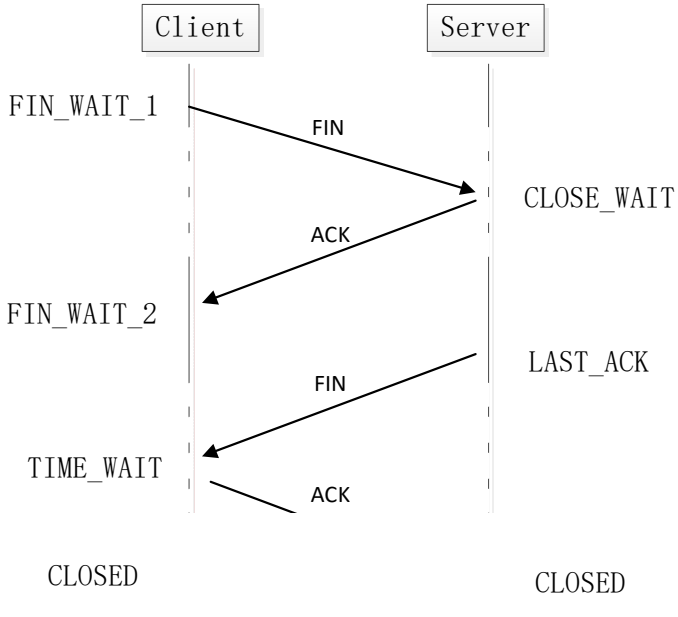


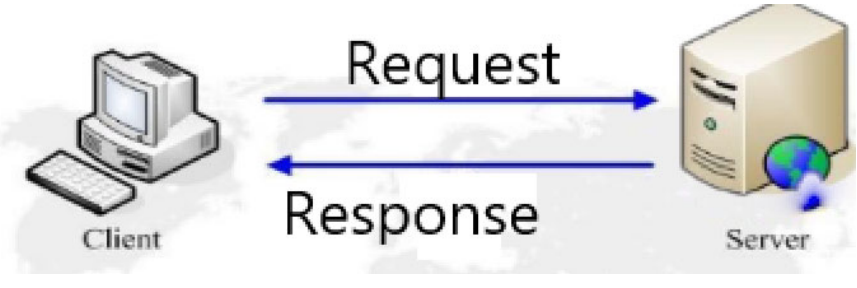
Table 1 TCP/IP protocol structure

TCP/IP	*Functions*	*TCP/IP protocol family*
Application layer	File transfer, e-mail, file service, virtual terminal	TFTP, HTTP, SNMP, FTP, SMTP, DNS, Telnet, etc.
	Translation, encryption, compression	No protocol
	Dialogue control, establishment of synchronisation points (continued transfer)	No protocol
Transmission layer	Port addressing, segment reassembly, flow, error control	TCP, UDP
Network layer	Logical block addressing, routing addressing	IP, ICMP, OSPF, EIGRP, IGMP, RIP, ARP, RAPP
Data link layer	Framing, physical addressing, flow, error, access control	SLIP, CSLIP, PPP, MTU
	Setting network topology, bit transmission and bit synchronisation	ISO2110, IEEE802, IEEE802.2

Figure 38 TCP connection code (see online version for colours)

```
#define API_ADDR "jisuznwd.market.alicloudapi.com"
#define API_PROT 80
int fd = Socket(AF_INET,SOCK_STREAM,0);
struct hostent * appcode = gethostbyname(API_ADDR);
char str[32];
bzero(str,32);
if(appcode != NULL)
{
printf("address:%s\n",
inet_ntop(appcode->h_addrtype,*appcode->h_addr_list,str,sizeof(str)));
printf("API_ADDR : %s,type : %d\n",str,appcode->h_addrtype);
}
struct sockaddr_in srvaddr;
socklen_t len = sizeof(srvaddr);
bzero(&srvaddr,len);
srvaddr.sin_family = AF_INET;
//srvaddr.sin_addr.s_addr = inet_addr(*appcode->h_addr_list);
srvaddr.sin_addr.s_addr = inet_addr(str);
srvaddr.sin_port = htons(API_PROT);
Connect(fd,(struct sockaddr *)&srvaddr,len);
```

Figure 39 HTTP request/response model (see online version for colours)



The request line, request header, blank line, and request data are the four components of the request message that the client provides to the server when sending an HTTP request. The request example is given in Figures 40 and 41 shows the request message structure.

Figure 40 HTTP request information format

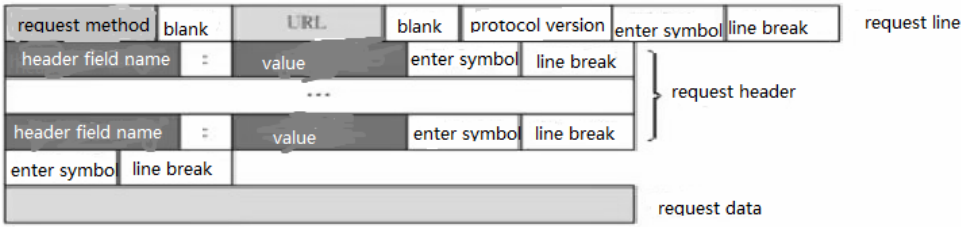


Figure 41 HTTP data request example

```
Request:
Url: http://jisuznwd.market.alicloudapi.com/iqa/query?question=how+are+you
Header: {"Host":"jisuznwd.market.alicloudapi.com","X-Ca-
Timestamp":"1528262459040","gateway_channel":"http","X-Ca-Request-
Mode":"debug","X-Ca-Key":"24655262","X-Ca-Stage":"RELEASE","Content-
Type":"application/json; charset=utf-8","Authorization":"APPCODE
62c8124bd23848dab58686cf7e1f7876"}
```

Figure 42 HTTP response message example

```
Response:
200
Date: Wed, 06 Jun 2018 05:20:59 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 128
Connection: keep-alive
Access-Control-Allow-Origin:*
Access-Control-Allow-Methods:
GET.POSTPUT.DELETE,HEAD,OPTIONS,PATCHAccess-Control-Allow-Headers: X-
Requested-With,X-Sequence,X-Ca-Key,XCa-SecretX-Ca-Version,X-Ca-
Timestamp,X-Ca-Nonce,X-Ca-API-Key,X-Ca-Stage,X-Ca-Client-DeviceId,X-Ca-
Client-AppId,X-Ca-Signature,X-Ca-Signature-Headers,X-Ca-Signature-Method,X-
Forwarded-For,X-Ca-DateXCa-Request-Mode,Authorization,Content-
Type,Accept,Accept-Ranges,Cache-Control,Range,Content-MD5
Access-Control-Max-Age: 172800
X-Ca-Request-Id: 3FA41AB4-A496-40F9-AA65-413E322B5815
X-Powered-By: PHP/5.4.16
X-Ca-Error-Message: OK
Latency: 123

{"status":"0","msg":"ok","result":{"type":"chat""content"."Hey, I am robot X, what
can I do for you?","relquestion":""}}
```

Alibaba Cloud will process the request sent by the customer, and will return an HTTP response message after processing. The status line, information header, blank line, and response body are all components of an HTTP response. The response message example is as shown in Figure 42.

The specific implementation code of the response data packet obtained by calling Alibaba Cloud API via HTTP is in Figure 43.

Figure 43 Calling Alibaba Cloud API via HTTP (see online version for colours)

```
Char buf1 [256];
int size = sizeof(buf1);
bzero (buf1,size);
snprintf(buf1,size,"GET /iqa/query?question=%s HTTP/1.1\r\n"
"Host:jisuznwd.market.alicloudapi.com\r\n"
"Authorization:APPCODE 62c8124bd23848dab58686cf7e1f7876\r\n\r\n",
question);
Write(fd,buf1,size);
printf("%s",buf1);

char *buf2 = malloc(4096);
bzero(buf2,strlen(buf2)+1);
int m = 0;
int n = 0;
while(1)
{
n = Read(fd,buf2+m,1);
m += n;
if( n == 0)
break;
if(strstr(buf2,"\"}]")!=NULL)
{
break;
}
}
printf("%s\r\n\r\n",buf2);
printf("\r\n--------------------------------\r\n");
printf("buf2 strlen %d\n",strlen(buf2));
```

Access the Alibaba Cloud server by GET, read the response data of the server, and get the optimal data package.

2 JSON data analysis: The response data obtained through Alibaba Cloud is saved in JSON format. The data in JSON exists as a key-value pair in the form of 'key': 'value'. JSON data parsing only has the following two situations: parsing 'curly braces' {}; parsing 'brackets' []. With Figure 42 as an example, after formatting the response text according to JSON, user can get Figure 44.

Therefore, as long as the value corresponding to the keyword 'content' is retrieved, the core data of this response can be obtained.

The specific implementation code of JSON data parsing is in Figure 45.

Figure 44 JSON data formatting


```
{
"status":"0",
"msg":"ok",
"result":{
  "type":"chat"
  "content"."Hey, I am robot X, what can I do for you?",
  "relquestion":""
}
}
```


Figure 45 JSON data parsing (see online version for colours)

```
int jcon_analyze(char * msg, char * answer)
{
char tmp[]={0};
cJSON *root = cJSON_Parse(msg);
if(strcmp(cJSON_GetObjectItem(root,"msg")->valuestring,"ok") != 0)
{
printf("analyze error!\n");
return 1;
}
cJSON *result = cJSON_GetObjectItem(root,"result");
strncpy(answer,
cJSON_GetObjectItem(result,"content")->valuestring,BUFFER_SIZE-1);
answer[strlen(answer)+1] = '/0';
cJSON_Delete(root);
return 0;
}
```

Through the parsing of JSON data, the core data in the response packet is obtained. Finally, this text data is converted into an audio file for the ARM development board.

6 Main achievements and research outcomes

Our research has propelled forward the capabilities of child companion robots, especially in areas of speech recognition and personalised interaction, marking significant breakthroughs in the field. By leveraging advanced technologies like Transformer models coupled with real-time cloud injection, we have significantly enhanced speech recognition accuracy. This has led to a notable decrease in word error rates and boosted the relevance of conversations, enabling these robots to better comprehend and respond to children's instructions and needs. A pivotal advancement has been in devising age-appropriate and safe interactions through sophisticated transfer learning and content filtering mechanisms. This innovation has allowed the robots to engage in personalised communication, tuned to the unique needs and interests of each child, thereby enriching the interactive experience with more fun and educational value.

Moreover, our contributions extend to enhancing the overall communication fluency between robots and children, improving the user experience by facilitating safe, enjoyable, and educational companionship. This supports children's learning and developmental processes in profound ways. Despite these advancements, the journey ahead presents both challenges and opportunities for further innovation in the realm of child companion robots. Addressing issues such as managing prolonged conversations and integrating emotion perception are critical next steps. Investigating new methodologies, including the adoption of memory networks and emotion recognition technologies, holds the promise of surmounting these challenges. Enhancing the learning capabilities of these robots is essential for fostering a more intelligent and personalised companionship.

The future landscape for the development of child companion robots is rich with possibilities. As technological advancements continue to evolve, the integration of memory networks and emotion detection technologies is anticipated to yield even more intelligent and reliable companion robots. The potential for these robots to make significant impacts across educational, entertainment, and social interactions is immense, suggesting a future where child companion robots play a central role in enriching children's learning experiences, entertainment, and social engagements.

Our research not only underscores the substantial progress made but also illuminates the path for future exploration and adoption of child companion robots. As we look forward to continued innovation and development, our goal remains to enhance the lives of children by making child companion robots indispensable, intelligent companions. This journey of discovery and innovation opens new doors to the widespread adoption of child companion robots, promising a future where they become an integral part of children's lives, supporting their growth, learning, and enjoyment in unprecedented ways.

Acknowledgements

This work was supported by Scientific Research Key Project of Hunan Education Department [21A0592]. Educational Reform Project of Hunan Province [HNJG-2021-1378]. Changsha Municipal Natural Science Foundation kq2014063, Provincial Natural Science Foundation of Hunan (2023JJ30101).

References

Atieh, A.T. (2021) 'The next generation cloud technologies: a review on distributed cloud, fog and edge computing and their opportunities and challenges', *Research Berg Review of Science and Technology*, Vol. 1, No. 1, pp.1–15.

Chen, Q. and Deng, Q. (2009) 'Cloud computing and its key technologies', *Journal of Computer Applications*, Vol. 29, No. 9, pp.2562–2567.

Cheng, S., Li, Y., Chen, X., Chen, S. and Ge, H. (2020) 'Construction of virtual simulation experiment platform under cloud computing architecture', *Laboratory Research and Exploration*, Vol. 39, No. 12, pp.238–241.

Jiang, D. (2020) 'The construction of smart city information system based on the internet of things and cloud computing', *Computer Communications*, Vol. 150, No. 2020, pp.158–166.

Kehoe, B., Patil, S., Abbeel, P. et al. (2015) 'A survey of research on cloud robotics and automation', *IEEE Transactions on Automation Science and Engineering*, Vol. 12, No. 2, pp.398–409.

Li, W., Liu, Z., Wu, W. and He, S. (2013) 'Design of power system geographic wiring diagram drawing software based on Qt', *Automation of Electric Power Systems*, Vol. 37, No. 7, pp.72–76+107.

Lin, Y. (2017) *Applied Research of Children Companion Robot in Remote Parent-Child Interaction Design*, Huaqiao University, Quanzhou.

Lin, Y. and Shi, W. (2018) 'Design and implementation of intelligent vehicle multimedia control terminal based on embedded system', *Automation and Instrumentation*, Vol. 8, No. 2018, pp.100–103.

Lu, J., Shan, L., Wang, J., Li, J. and Sun, Y. (2015) 'Distributed multi-channel data acquisition instrument based on ARM', *Computer Engineering and Science*, Vol. 37, No. 5, pp.1031–1036.

Lu, S., Zhou, S. and Mao, Y. (2021) 'Formal analysis and optimization of TLS1.3 protocol under strong security model', *Journal of Software*, Vol. 32, No. 9, pp.2849–2866.

Rui, H., Hong, C. and Jia, L. (2011) 'Design and realization of an elderly escort robot system', *Electroacoustic Technology*, Vol. 35, No. 5, pp.41–44.

Ruiz-Casares, M. and Heymann, J. (2009) 'Children home alone unsupervised: modeling parental decisions and associated factors in Botswana, Mexico, and Vietnam', *Child Abuse & Neglect*, Vol. 33, No. 5, pp.312–323.

Sarddar, D. and Bose, R. (2014) 'Architecture of server virtualization technique based on VMware ESX sever in the private cloud for an organization', *International Journal of innovation and Scientific Research*, Vol. 12, No. 1, pp.284–294.

Xue, F. and Zhang, Y. (2021) 'Embedded multimedia terminal control system based on policy derivation', *Journal of University of Jinan (Natural Science Edition)*, Vol. 35, No. 5, pp.439–445.

Zhang, L., Wan, Y. and Donghui, F. (2021) 'Electrical data storage scheme based on blockchain', *Computer Applications and Software*, Vol. 38, No. 9, pp.21–27+52.

Zhang, Y., Li, Z., He, H. and Hu, Y. (2022) 'Design and implementation of ARM-based children's accompanying robot', *Forest Chemicals Review*, Vol. 2, No. 4, pp.1442–1463.

Zhang, Y., Li, Z., He, H. and Hu, Y. (2022) 'Design and implementation of ARM-based children's accompanying robot', *Forest Chemicals Review*, March–April, pp.1442–1463.

Zhiwei, L., Xin, J. and Songhao, Z. (2018) 'Development approaches of smart space service components for service robots', *Robot*, Vol. 32, No. 3, pp.337–343.

Zhu, L., Xie, J., Sun, L. and Xu, R. (2016) 'Sampling signal and switch quantity under the Wi-Fi module of hybrid coding transmission', *Journal of Foreign Electronic Measurement Technology*, Vol. 35, No. 5, pp.53–56.