Chat-O-Tron Project Defense Report

Project Name: Chat-O-Tron

Presenter: G7

Presentation Date: December 04, 2025

Table of Contents

Project Overview: Background and Goals

Project Background

Growing demand for AI dialogue system applications

Technical challenges in multi-end collaboration and real-time interaction

Project Goals

Develop a high-performance, cross-platform intelligent dialogue system

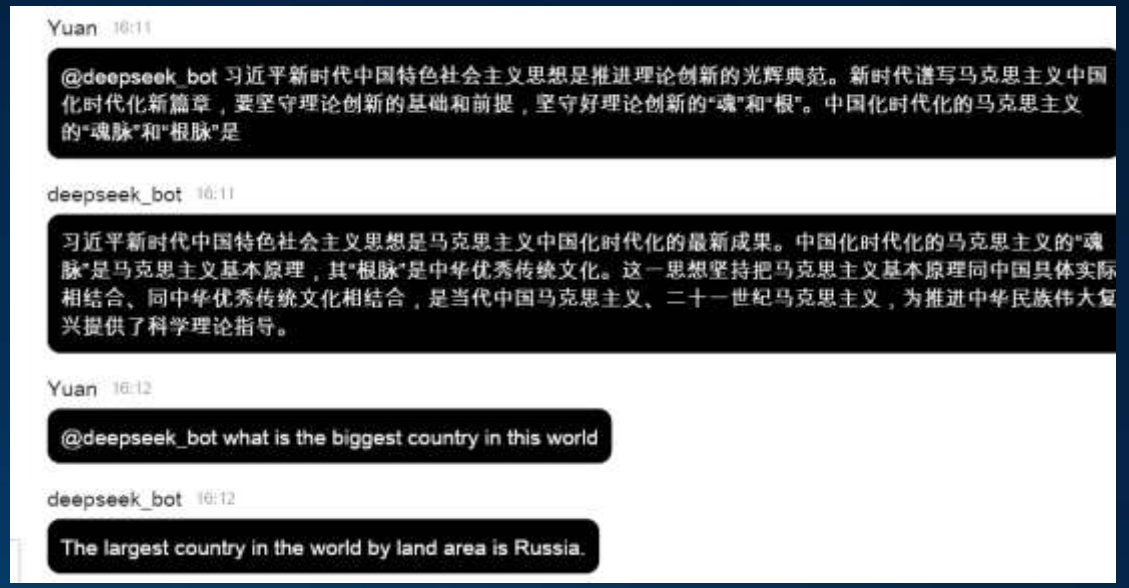
Support AI robot integration and third-party API extensions

Technology Stack

Backend: Rust (High-performance network services)

Frontend: JavaScript (Graphical user interface)

Integration: AI robot framework + API gateway



Project Overview: Overall Architecture

This project adopts a modular design to build an efficient and scalable intelligent interaction system.

System Architecture Diagram

Three-tier architecture: Client Layer, Server Layer, AI Integration Layer

Core Modules

- Network Service Module (Rust): Handles client connections and message forwarding
- Frontend Interface Module (JS): User interaction and visual display
- AI Robot Module: Dialogue logic and intelligent response
- API Integration Module: Third-party service integration

Network Server Team: Task Division

Team Division

01 XUEJIPEI: TCP/UDP Protocol Implementation

02 YUANWENJIE: Message Serialization and Deserialization

03 JIANGYAXING: Concurrent Connections and Performance Optimization

Core Goals

- Implement low-latency, high-concurrency network communication service
- Support multiple clients with simultaneous connections and message synchronization

Network Server Team: Rust Network Fundamentals

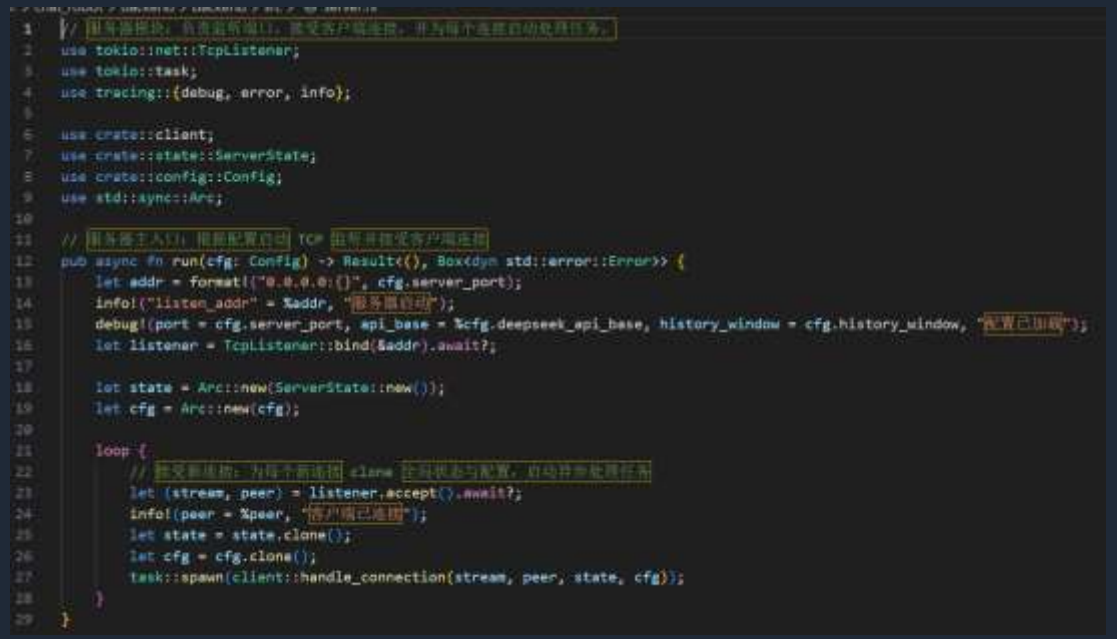
Advantages of Rust Network Programming

- Memory safety with zero-cost abstractions
- Efficiency of async IO framework (Tokio)

Key Code Examples

- Tokio framework initialization code
- TCP listening and connection handling logic

TCP listening



```rust
// 服务器模块：负责监听端口，接受客户端连接，并为每个连接启动处理任务。
use tokio::net::TcpListener;
use tokio::task;
use tracing::{debug, error, info};

use crate::client;
use crate::state::ServerState;
use crate::config::Config;
use std::sync::Arc;

// 服务器主入口：根据配置启动 TCP 监听并接受客户端连接
pub async fn run(cfg: Config) -> Result<(), Box<dyn std::error::Error>> {
    let addr = format!("0.0.0.0:{}", cfg.server_port);
    info!("listen_addr" = %addr, "服务器启动");
    debug!(port = cfg.server_port, api_base = %cfg.deepseek_api_base, history_window = cfg.history_window, "配置已加载");
    let listener = TcpListener::bind(&addr).await?;

    let state = Arc::new(ServerState::new());
    let cfg = Arc::new(cfg);

    loop {
        // 接受新连接，为每个新连接 clone 共享状态与配置，启动异步处理任务
        let (stream, peer) = listener.accept().await?;
        info!(peer = %peer, "客户端已连接");
        let state = state.clone();
        let cfg = cfg.clone();
        task::spawn(client::handle_connection(stream, peer, state, cfg));
    }
}
```

Network Server Team: Message Protocol Design

Custom Message Format

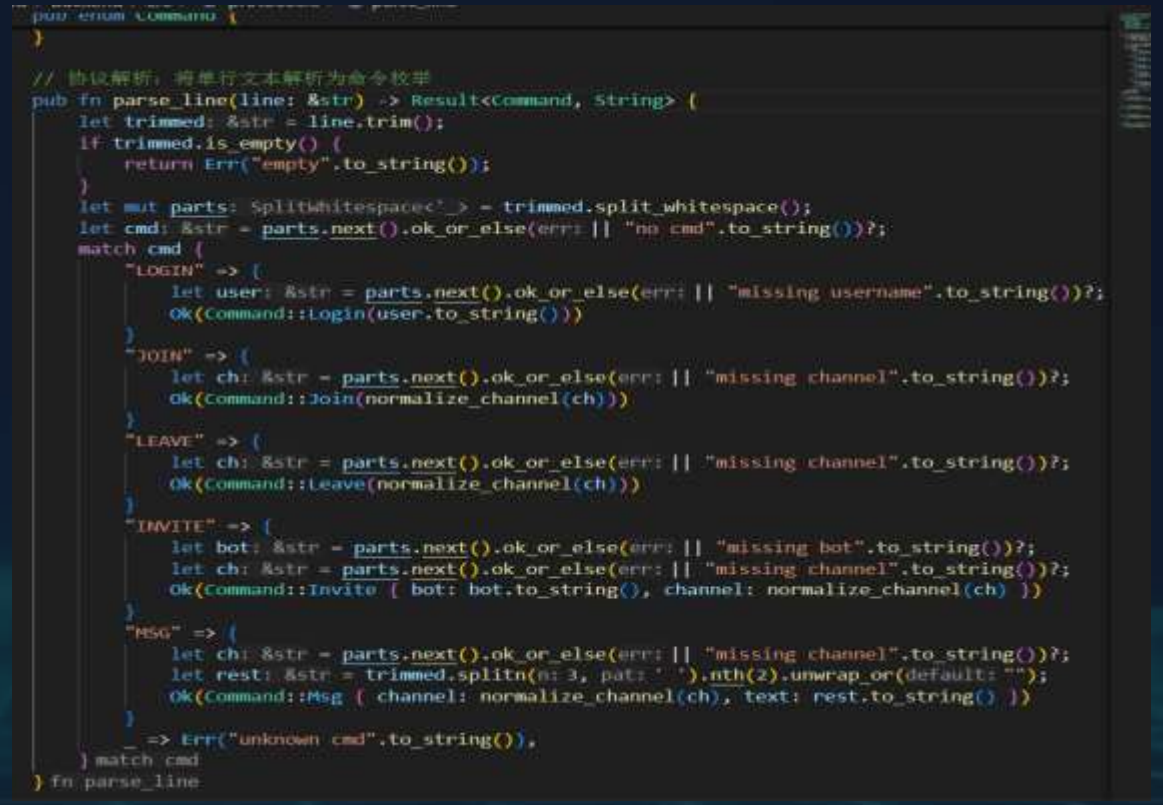
Message Header (Type + Length + Checksum)

Message Body (JSON/Protobuf Serialization)

Code Examples

Protobuf Definition File (.proto)

Protobuf Serialization/Deserialization Code in Rust



```
pub enum Command {
}

// 协议解析：将单行文本解析为命令枚举
pub fn parse_line(line: &str) -> Result<Command, String> {
    let trimmed: &str = line.trim();
    if trimmed.is_empty() {
        return Err("empty".to_string());
    }
    let mut parts: SplitWhitespace<'_> = trimmed.split_whitespace();
    let cmd: &str = parts.next().ok_or_else(err: || "no cmd".to_string())?;
    match cmd {
        "LOGIN" => {
            let user: &str = parts.next().ok_or_else(err: || "missing username".to_string())?;
            Ok(Command::Login(user.to_string()))
        }
        "JOIN" => {
            let ch: &str = parts.next().ok_or_else(err: || "missing channel".to_string())?;
            Ok(Command::Join(normalize_channel(ch)))
        }
        "LEAVE" => {
            let ch: &str = parts.next().ok_or_else(err: || "missing channel".to_string())?;
            Ok(Command::Leave(normalize_channel(ch)))
        }
        "INVITE" => {
            let bot: &str = parts.next().ok_or_else(err: || "missing bot".to_string())?;
            let ch: &str = parts.next().ok_or_else(err: || "missing channel".to_string())?;
            Ok(Command::Invite { bot: bot.to_string(), channel: normalize_channel(ch) })
        }
        "MSG" => {
            let ch: &str = parts.next().ok_or_else(err: || "missing channel".to_string())?;
            let rest: &str = trimmed.splitn(n: 3, pat: ' ').nth(2).unwrap_or(default: "");
            Ok(Command::Msg { channel: normalize_channel(ch), text: rest.to_string() })
        }
        _ => Err("unknown cmd".to_string()),
    } match cmd
} fn parse_line
```

Protobuf Message Structure Definition

```
/消息类型定义
num MessageType {
HEARTBEAT = 0;
DATA_REQUEST = 1;
DATA_RESPONSE = 2;
ERROR  3;
```

Network Server Team: Concurrent Connection Management

Concurrency Model

- Each connection corresponds to an asynchronous task
- Shared state management (Arc+Mutex)

Code Examples

- Connection pool implementation code
- Message broadcasting logic (forward to all clients)

```rust
use crate::config::Config;

// 客户端连接处理，解析指令并执行业务逻辑
pub async fn handle_connection(stream: TcpStream, peer: SocketAddr, state: Arc<ServerState>, c
    let (read_half: OwnedReadHalf, write_half: OwnedWriteHalf) = stream.into_split();
    let mut reader: BufReader<OwnedReadHalf> = BufReader::new(inner: read_half);
    let (tx: UnboundedSender<String>, mut rx: UnboundedReceive…) = unbounded_channel::<String>
    let mut writer: Writer = Writer::new(inner: write_half);
    tokio::spawn(future: async move {
        // 写入循环，从通道接收要返回给客户端的文本行
        while let Some(line: String) = rx.recv().await {
            let _ = writer.write_line(&line).await;
        }
    });

    info!(peer = %peer, "发送欢迎消息");
    let _ = tx.send(message: encode_info(text: "Welcome to Chat-O-Tron"));

    let mut username: Option<String> = None;
    let mut buf: String = String::new();
    loop {
        buf.clear();
        let n: Result<usize, Error> = reader.read_line(&mut buf).await;
        match n {
            Ok(0) => break,
            Ok(_) => {
                debug!(raw = buf.trim_end(), "收到一行");
                match parse_line(&buf) {
                    Ok(cmd: Command) => {
                        debug!(?cmd, "指令解析成功");
                        if let Err(e: String) = process_command(cmd, &state, &tx, &mut usernam
                            let _ = tx.send(message: encode_error(text: &e));
                            warn!(error = %e, "指令处理失败");
                        }
                    }
                    Err(e: String) => {
```

```rust
use tracing::debug;

// 广播文本行到指定频道的所有成员
pub fn broadcast(state: &ServerState, channel: &str, line: String) {
    if let Some(members: Ref<'_, String, Vec<ClientHandle>>) = state.channels.get(key: channel
        debug!(channel = %channel, recipients = members.len(), "广播: 发送");
        for c: &ClientHandle in members.iter() {
            let _ = c.tx.send(message: line.clone());
        }
    }
}
```

Network Server Team: Key File Description

Core File List

1. src/main.rs

   Service entry and initialization

2. src/network/tcp.rs

   TCP connection handling

3. src/message/proto.rs

   Protobuf message definitions

4. src/utils/logger.rs

   Logging system integration

Dependency Graph

File Dependency Relationships

Graphical Interface Team: Task Division

Team Division

- ZHENGJINHAO: Frontend interface design and implementation
- XIEZIJING: Client-server communication

Core Goals

- Develop an intuitive and smooth user interaction interface
- Implement real-time message synchronization with backend services

Graphical Interface Team: Frontend Technology Stack

Technology Selection

Framework: React (Component-based Development)

State Management: Redux (Global State Synchronization)

Communication: TCP stream

Advantages

Component Reusability and Development Efficiency Improvement

Responsive Design for Multiple Devices

Graphical Interface Team: Interface Design

Core Interface Modules

01 Login Interface: User authentication and connection establishment

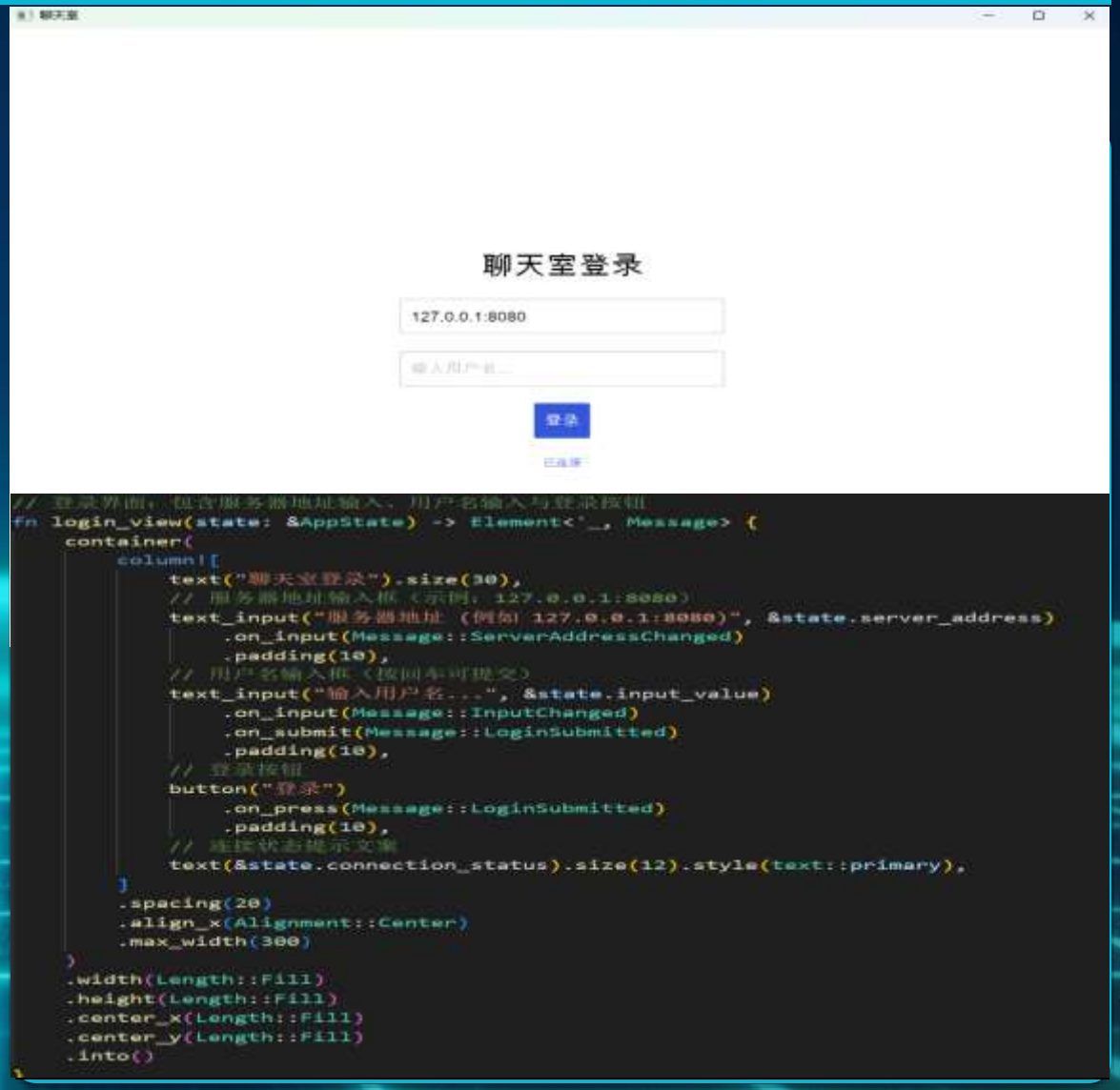
02 Chat Interface: Message list + input box + send button

03 Settings Interface: Theme switching + notification configuration

UI/UX Design Principles

Simple and clear with short operation paths

Real-time feedback (message sending status prompts)




```
// 登录界面：包含服务器地址输入、用户名输入与登录按钮
fn login_view(state: &AppState) -> Element<'_, Message> {
    container(
        column![
            text("聊天室登录").size(30),
            // 服务器地址输入框（示例：127.0.0.1:8080）
            text_input("服务器地址（例如 127.0.0.1:8080）", &state.server_address)
                .on_input(Message::ServerAddressChanged)
                .padding(10),
            // 用户名输入框（按回车可提交）
            text_input("输入用户名...", &state.input_value)
                .on_input(Message::InputChanged)
                .on_submit(Message::LoginSubmitted)
                .padding(10),
            // 登录按钮
            button("登录")
                .on_press(Message::LoginSubmitted)
                .padding(10),
            // 连接状态提示文案
            text(&state.connection_status).size(12).style(text::primary),
        ]
        .spacing(20)
        .align_x(Alignment::Center)
        .max_width(300)
    )
    .width(Length::Fill)
    .height(Length::Fill)
    .center_x(Length::Fill)
    .center_y(Length::Fill)
    .into()
```

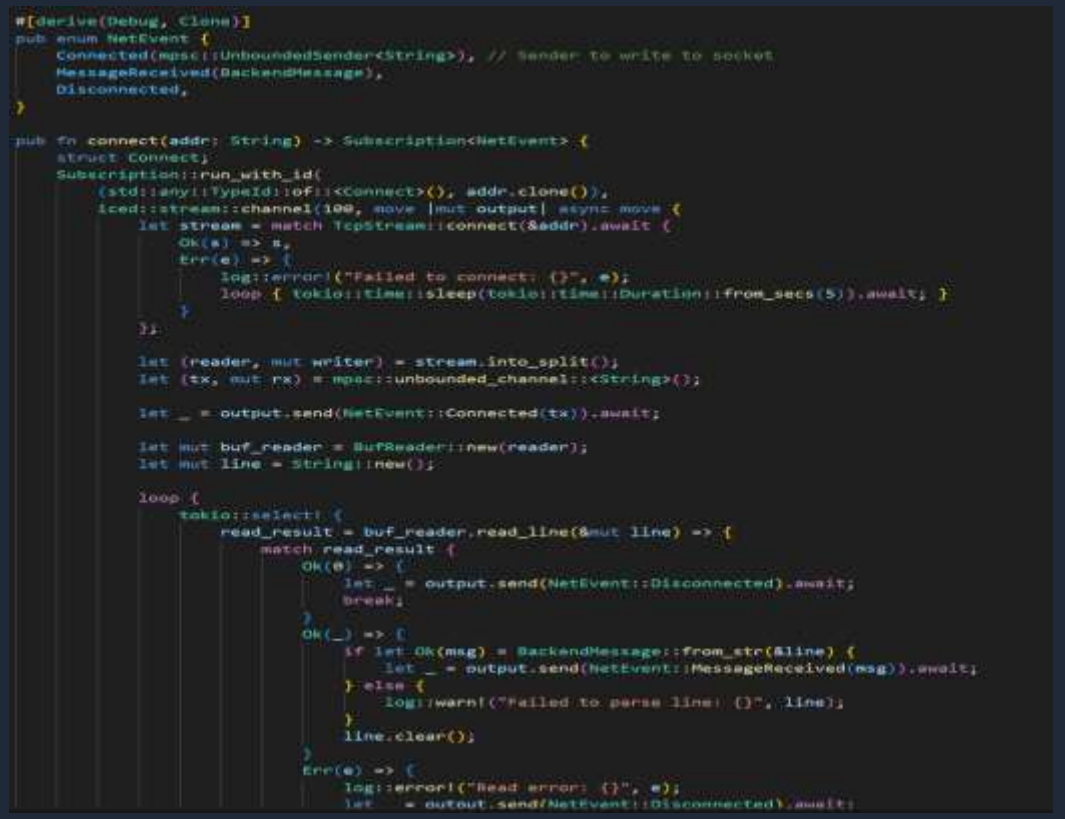
Reserved code image space : Chat interface prototype

Graphical Interface Team: TCP STREAM

Advantages

Reliable Transmission: Features a built-in retransmission and acknowledgment mechanism to ensure the ultimate delivery of data.
Ordered Delivery: The data sequence received by the receiver is consistent with that sent by the sender.
Error Detection and Recovery: Mechanisms such as checksum and ACK detect and handle errors or packet loss.
Flow Control: Prevents the sender from overwhelming the receiver (via the sliding window mechanism).
Congestion Control: Automatically adjusts the transmission rate when network congestion occurs to safeguard network stability.
Connection-Oriented: Establishes a session through handshaking, facilitating the maintenance of end-to-end states and graceful termination.
Byte Stream Abstraction: Provides a continuous byte stream, making it easy to implement various upper-layer protocols (such as HTTP, SSH, etc.).
Wide Compatibility: Natively supported by nearly all operating systems and network libraries, boasting strong interoperability.

Code Example



Graphical Interface Team: State Management

Redux Core Concepts

- Action: Message type definition
- Reducer: State update logic
- Store: Global state storage

Code Examples

- Reducer code for chat message state
- Asynchronous Action (WebSocket message receiving)

```rust
#[derive(Debug, Clone)]
pub struct ChatMessage {
    pub sender: String,
    pub content: String,
    pub timestamp: DateTime<Local>,
    pub is_system: bool,
}

impl ChatMessage {
    // Helper to suppress unused warning if we want to keep the field
    pub fn is_system(&self) -> bool {
        self.is_system
    }
}

#[derive(Debug, Default)]
pub struct AppState {
    pub username: Option<String>,
    pub active_channel: Option<String>,
    pub joined_channels: HashSet<String>,
    pub chat_logs: HashMap<String, Vec<ChatMessage>>,
    pub system_log: Vec<ChatMessage>, // For general INFO/ERROR when no channel is active or global
    pub input_value: String,
    pub server_address: String,
    pub connection_status: String,
}

impl AppState {
    pub fn new() -> Self {
        Self {
            connection_status: "未连接".to_string(),
            server_address: "127.0.0.1:8080".to_string(),
            ..Default::default()
        }
    }

    pub fn add_message(&mut self, channel: &str, sender: &str, content: &str) {
        let log = self.chat_logs.entry(channel.to_string()).or_default();
        log.push(ChatMessage {
            sender: sender.to_string(),
            content: content.to_string(),
            timestamp: Local::now(),
            is_system: false,
        });
    }

    pub fn add_system_message(&mut self, content: &str, is_error: bool) {
```

Graphical Interface Team: Key File Documentation

Core Files List

`src/App.js`: Routing and global layout

`src/components/ChatWindow.js`: Chat interface component

`src/services/WebSocketService.js`: Communication service encapsulation

`src/store/reducers/messageReducer.js`: State management

Component Hierarchy Diagram

AI Robot Integration Team: Task Assignment

AI Robot Conversation Logic

Team Division

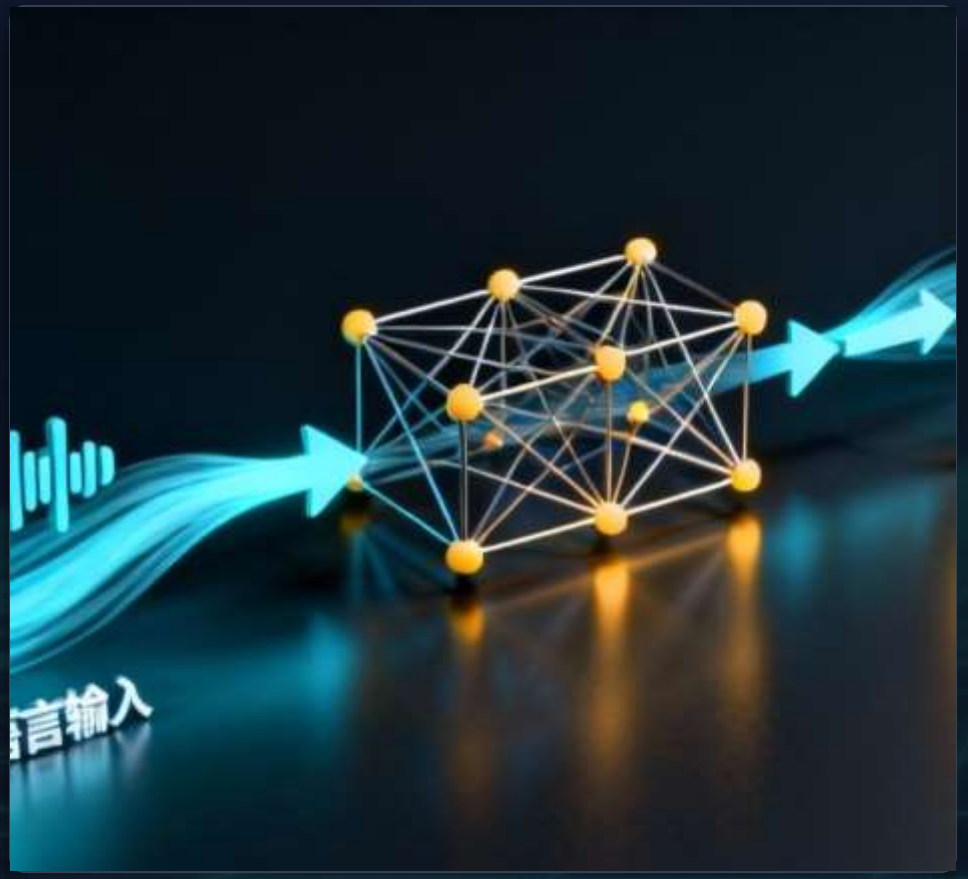
ZHOUKAIQI: AI Robot Conversation Logic
TONGWEI: API Gateway and Third-party Service Integration

Core Objectives

Implement intelligent conversation responses
Support third-party API calls (e.g., weather, news)

API Gateway and Service Integration

AI Robot Integration Team: Conversation Logic Design



Rasa Intent Recognition Rules

AI Robot Framework

Rule-based dialogue engine

Intent recognition and entity extraction

Code Examples

Rasa configuration file (domain.yml)

Custom action code (Rust)

AI Robot Integration Team: Rust and AI Interaction

Rust Calling AI Services

- HTTP client (Reqwest) sending requests
- response parsing and processing

Code Example

- Reqwest asynchronous request code
- AI response result encapsulation

```rust
// 调用 DeepSeek Chat 完成接口，返回回复文本
pub async fn chat(history: VecDeque<Message>, cfg: &Config) -> Result<String, String> {
    let key: _ = cfg.deepseek_api_key.clone().ok_or_else(|| "missing api key".to_string())?;
    let base: String = build_endpoint(&cfg.deepseek_api_base);

    let mut msgs: Vec<DSMessage> = Vec::new();
    msgs.push(DSMessage { role: "system".to_string(), content: "你是一名在频道聊天中参与对话的助手。只用简洁中文回复，不使用 emoji 或表情符号。".to_string() });
    for m: &{unknown} in history.iter() {
        let role: &str = if m.sender == "deepseek_bot" { "assistant" } else { "user" };
        msgs.push(DSMessage { role: role.to_string(), content: format!("[{}] {}", m.sender, m.text) });
    }
    let req: DSRequest = DSRequest { model: "deepseek-chat".to_string(), messages: msgs, stream: Some(false) };
    debug!(endpoint = %base, "DeepSeek 请求构建");
```

```rust
// Bot 可能回复，检查配置与上下文，调用 DeepSeek 获取回复
pub async fn maybe_respond(state: Arc<ServerState>, cfg: Arc<Config>, channel: String) {
    if cfg.deepseek_api_key.is_none() {
        warn!("Bot 未启用，缺少 API Key");
        broadcast(&state, &channel, line: encode_msg(&channel, user: "deepseek_bot", text: "Dee
        return;
    }
    state.mark_bot_trigger(&channel);
    let suppress_fallback: bool = state.within_cooldown(&channel, cfg.bot_cooldown_ms);
    let history: VecDeque<Message> = state.history.get(key: &channel).map(|q: Ref<'_, String,
    debug!(channel = %channel, history_len = history.len(), "bot 上下文就绪");
    match deepseek::chat(history, &cfg).await {
        Ok(r: String) => {
            info!(channel = %channel, reply_len = r.len(), "bot 回复");
            broadcast(&state, &channel, line: encode_msg(&channel, user: "deepseek_bot", text:
        }
        Err(e: String) => {
            warn!(error = %e, "bot 调用失败");
            if !suppress_fallback {
                let reply: String = match e.as_str() {
                    "INSUFFICIENT_BALANCE" => "DeepSeek 余额不足或配额用尽".to_string(),
                    "INVALID_API_KEY" => "DeepSeek API Key 无效或未授权".to_string(),
                    "RATE_LIMITED" => "DeepSeek 接口频率限制，请稍后重试".to_string(),
                    _ => "DeepSeek 服务异常，请稍后再试".to_string(),
                };
                info!(channel = %channel, reply_len = reply.len(), "bot 回复");
                broadcast(&state, &channel, line: encode_msg(&channel, user: "deepseek_bot", te
            }
        }
    }
} async fn maybe_respond
```

AI Robot Integration Team: API Gateway Design

API Gateway Functions

- Request Routing and Load Balancing
- Authentication and Rate Limiting

Supported Third-party APIs

- Deepseek API

Code Examples

- API Gateway Routing Configuration
- Request Forwarding Logic

```rust
impl Config {
    /// 加载配置：从文件与环境变量合并得到最终配置
    pub fn load() -> Self {
        let file_cfg: Option<FileConfig> = std::fs::read_to_string(path: "config.toml").ok().a
        let env_port: Option<u16> = std::env::var(key: "SERVER_PORT").ok().and_then(|s: String
        let env_key: Option<String> = std::env::var(key: "DEEPSEEK_API_KEY").ok();
        let env_base: Option<String> = std::env::var(key: "DEEPSEEK_API_BASE").ok();
        let env_hist: Option<usize> = std::env::var(key: "HISTORY_WINDOW").ok().and_then(|s: S
        let env_cool: Option<u64> = std::env::var(key: "BOT_COOLDOWN_MS").ok().and_then(|s: St

        let server_port: u16 = env_port Option<u16>
            .or_else(|| file_cfg.as_ref().and_then(|c: &FileConfig| c.server.as_ref().and_the
            .unwrap_or(default: 8080);
        let deepseek_api_key: Option<String> = env_key.or_else(|| file_cfg.as_ref().and_then(
        let deepseek_api_base: String = env_base Option<String>
            .or_else(|| file_cfg.as_ref().and_then(|c: &FileConfig| c.deepseek.as_ref().and_t
            .unwrap_or_else(|| "https://api.deepseek.com/chat/completions".to_string());
        let history_window: usize = env_hist Option<usize>
            .or_else(|| file_cfg.as_ref().and_then(|c: &FileConfig| c.history_window)) Option<
            .unwrap_or(default: 20);

        let bot_cooldown_ms: u64 = env_cool Option<u64>
            .or_else(|| file_cfg.as_ref().and_then(|c: &FileConfig| c.bot_cooldown_ms)) Option
            .unwrap_or(default: 2000);

        Self { server_port, deepseek_api_key, deepseek_api_base, history_window, bot_cooldown_
    } fn load
} impl Config
```

AI Robot Integration Team: API Call Example


```
// 天气API响应JSON示例
{
  "coord": { "lon": 139, "lat": 35 },
  "weather": [{ "id": 800, "main": "Clear",
"description": "clear sky" }],
  "main": { "temp": 282.55, "feels_like": 281.86,
"temp_min": 280.37, "temp_max": 284.26, "pressure":
1023, "humidity": 100 },
  "name": "Tokyo"
}
```

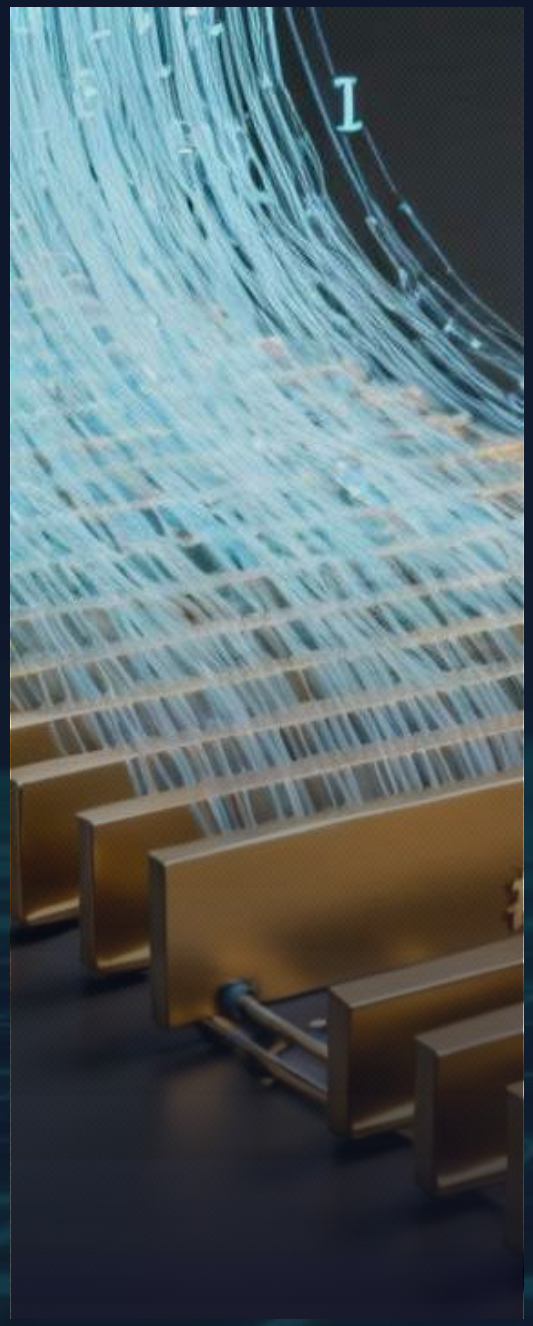
Weather API Call Flow

1. Client sends weather query request
2. Server forwards to API gateway
3. Gateway calls OpenWeatherMap API
4. Result returns to client

Code Examples

Weather API request parameter construction

Response data parsing and formatting



AI Robot Integration Team: Key File Description

Core Files List

1. src/ai/rasa_client.rs
   Rasa Service Client
2. src/api_gateway/router.rs
   API Routing Configuration
3. src/api_providers/weather.rs
   Weather API Wrapper
4. src/utils/config.rs
   API Key Management

```
// 邀请机器人：当前支持 deepseek_bot
Message::InviteBot => {
    if let Some(channel) = &self.state.active_channel {
        if let Some(sender) = &self.net_sender {
            let _ = sender.send(cmd_invite("deepseek_bot", channel));
        }
    }
}
```

Security Measures

- Encrypted Storage of API Keys

Project Achievements and Value

Functional Achievements

- Implemented cross-platform clients (Web/Desktop)
- Supports 1000+ concurrent connections
- AI dialogue accuracy rate of 85%

Technical Value

- Verified high-performance advantages of Rust in network services
- Explored multi-language collaborative development model

Application Scenarios

- Internal enterprise intelligent customer service
- Virtual teaching assistant in education
- Smart home control center

Future Outlook: Feature Expansion

Short-term Plans

Mobile client development (Flutter)

Voice interaction feature integration

Long-term Plans

Distributed service deployment (K8s)

Multimodal conversation support (text + image + voice)

Technical Challenges

Cross-platform compatibility optimization

Performance bottlenecks in multimodal data processing

Future Outlook: Technical Optimization

Performance Optimization Directions

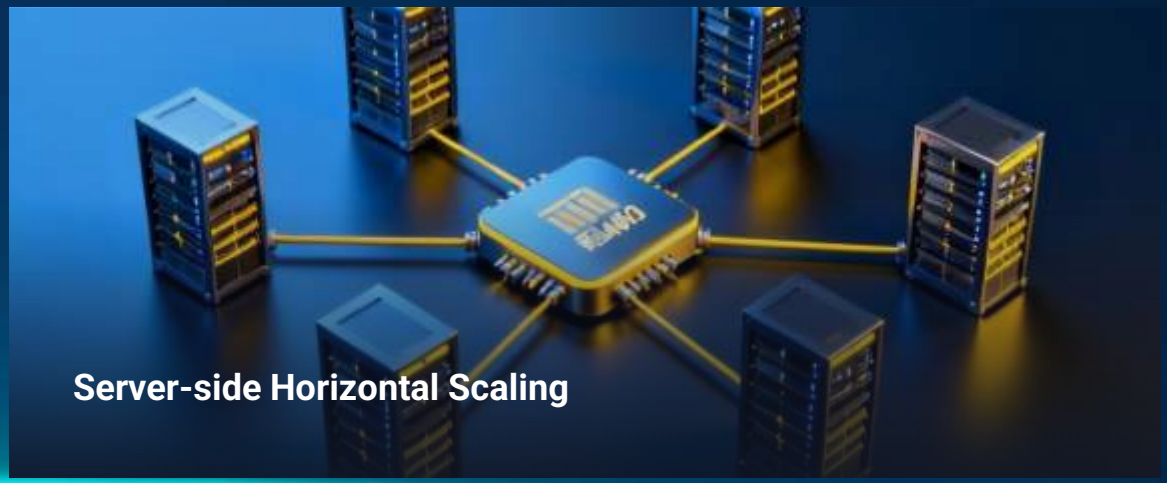
Server-side Horizontal Scaling (Load Balancing)

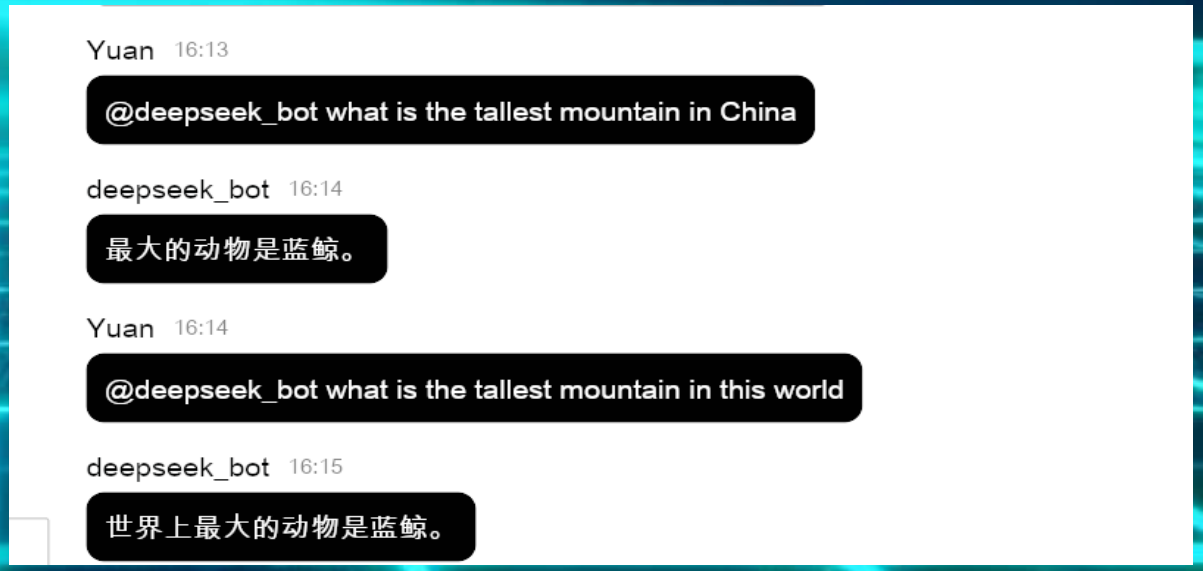
Frontend Rendering Optimization (WebAssembly)

Security Enhancement

End-to-End Encryption (TLS 1.3)

User Authentication Enhancement (JWT+OAuth2)





Team Summary and Acknowledgements

Team Collaboration Experience

Efficiency of Agile Development (Scrum)

Code review and knowledge sharing mechanisms

Acknowledgements

Professional guidance from instructors

Dedicated efforts from team members

Technical support from the open-source community

Thank You for Listening!

Chat-O-Tron Project Defense Report

Date: December 04, 2025