🎥 APIs e Tokens Corretos para Vision - Análise Técnica

📊 Comparação: ESP32 Server vs PY-Xiaozhi

Com base na análise do repositório xiaozhi-esp32-server, descobri como a Vision API funciona corretamente. Vou descrever aqui.

🔧 Configuração Correta da Vision API

1. **Arquivo de Configuração**: `core/api/vision_handler.py`

O ESP32 Server usa um arquivo chamado `vision_handler.py` que processa imagens e as envia para análise via Vision API.

Localização: `xiaozhi-server/core/api/vision_handler.py`

Funcionalidade Principal:

- Recebe imagens em formato base64
- Valida tamanho máximo de arquivo (5MB)
- Envia para VLLM (Vision Large Language Model)
- Retorna análise via API

🌐 Estrutura da Vision API

Endpoint HTTP

```
POST /mcp/vision/explain
```

Parâmetros Esperados

```
{
    "image": "base64_encoded_image",  # Imagem em base64
    "question": "O que você vê?",     # Pergunta sobre a imagem
    "device_id": "device_123",        # ID do dispositivo (opcional)
    "client_id": "client_456"         # ID do cliente (opcional)
}
```

Resposta Esperada

```
{
    "code": 0,
    "msg": "Success",
    "data": {
        "content": "Descrição detalhada do conteúdo da imagem...",
        "tokens_used": 1234
    }
}
```

🔑 Configuração do Token e URL

Via `config.yaml` (Xiaozhi-Server)

```
server:
  # Porta do servidor HTTP
  port: 8003

  # URL da Vision API (Zhipu AI)
  vision_explain: "http://api.xiaozhi.me/vision/explain"

  # Token de autenticação JWT
  auth_key: "seu-jwt-token-aqui"

# Configuração do módulo VLLM (Vision Language Model)
selected_module:
  VLLM: "zhipu"  # Usar Zhipu AI como provider

# Detalhes do provider Zhipu
VLLM:
  zhipu:
    type: "zhipu"
    api_key: "seu-api-key-zhipu"
    model: "glm-4v-vision"  # Modelo de visão do Zhipu
    temperature: 0.7
    max_tokens: 2048
```

🛠️ Como Funciona no Código

1. **Inicialização** - `app.py`

```
# Ao iniciar, gera/carrega um auth_key
auth_key = config["server"]["auth_key"]

# Inicia o servidor HTTP que processa requisições Vision
SimpleHttpServer(config).start()
```

2. **Processamento** - `vision_handler.py`

```
class VisionHandler(BaseHandler):
    async def handle_post(self, request):
        # 1. Valida autenticação via JWT
        self.auth.validate_token(token)

        # 2. Lê arquivo de imagem (base64)
        image_data = request.get("image")

        # 3. Valida tamanho (máx 5MB)
        if len(image_data) > MAX_FILE_SIZE:
            return error_response()

        # 4. Carrega configuração
        config = await get_private_config_from_api()

        # 5. Obtém o VLLM provider configurado
        vllm_type = config["selected_module"]["VLLM"]

        # 6. Cria instância do provider
        vllm = create_instance(vllm_type, config)

        # 7. Processa a imagem
        result = await vllm.analyze_image(
            image=image_data,
            question=request.get("question"),
            context=request.get("context")
        )

        # 8. Retorna resultado
        return success_response(result)
```

3. **Provider Zhipu** - `core/providers/vllm/zhipu.py`

```
class ZhipuVisionProvider:
    def __init__(self, config):
        self.api_key = config["api_key"]
        self.model = config["model"]  # "glm-4v-vision"
        self.api_url = "https://open.bigmodel.cn/api/paas/v4/chat/completions"

    async def analyze_image(self, image, question, context=None):
        # Prepara payload
        payload = {
            "model": self.model,
            "messages": [
                {
```

```
                    "role": "user",
                    "content": [
                        {
                            "type": "image_url",
                            "image_url": {
                                "url": f"{image}"
                            }
                        },
                        {
                            "type": "text",
                            "text": question
                        }
                    ]
                }
            ],
            "temperature": self.temperature,
            "max_tokens": self.max_tokens
        }

        # Envia para Zhipu API
        response = await self.http_client.post(
            self.api_url,
            json=payload,
            headers={
                "Authorization": f"Bearer {self.api_key}",
                "Content-Type": "application/json"
            }
        )

        # Processa resposta
        return response.json()["choices"][0]["message"]["content"]
```

Configuração Passo-a-Passo no PY-Xiaozhi

Passo 1: Configurar `config.yaml`

Editar arquivo `config/config.yaml` do projeto:

```
server:
  port: 8003
  vision_explain: "http://api.xiaozhi.me/vision/explain"
  auth_key: "sua-chave-jwt"

selected_module:
  VLLM: "zhipu"

VLLM:
  zhipu:
    type: "zhipu"
```

```
    api_key: "d66ea037-1b07-4283-b49b-b629e005c074"  # Token fornecido
    model: "glm-4v-vision"
    temperature: 0.7
    max_tokens: 2048
    api_url: "https://open.bigmodel.cn/api/paas/v4/chat/completions"
```

Passo 2: Implementar Provider Zhipu

Criar arquivo `src/mcp/tools/providers/vllm_provider.py`:

```
import asyncio
import json
import base64
from typing import Optional
import httpx

class VisionAPIProvider:
    """Provider para análise de imagens via Zhipu Vision API"""

    def __init__(self, api_key: str, model: str = "glm-4v-vision"):
        self.api_key = api_key
        self.model = model
        self.api_url = "https://open.bigmodel.cn/api/paas/v4/chat/completions"
        self.temperature = 0.7
        self.max_tokens = 2048

    async def analyze_image(self, image_base64: str, question: str) -> str:
        """
        Analisa uma imagem usando Zhipu Vision API

        Args:
            image_base64: Imagem em base64
            question: Pergunta sobre a imagem

        Returns:
            Descrição/análise da imagem
        """
        payload = {
            "model": self.model,
            "messages": [
                {
                    "role": "user",
                    "content": [
                        {
                            "type": "image_url",
                            "image_url": {
                                "url": f"{image_base64}"
                            }
                        },
                        {
```

```
                            "type": "text",
                            "text": question
                        }
                    ]
                }
            ],
            "temperature": self.temperature,
            "max_tokens": self.max_tokens
        }

        async with httpx.AsyncClient() as client:
            response = await client.post(
                self.api_url,
                json=payload,
                headers={
                    "Authorization": f"Bearer {self.api_key}",
                    "Content-Type": "application/json"
                },
                timeout=30.0
            )

        result = response.json()

        if response.status_code == 200:
            return result["choices"][0]["message"]["content"]
        else:
            raise Exception(f"Vision API Error: {result}")

# Uso
async def test_vision():
    provider = VisionAPIProvider(
        api_key="d66ea037-1b07-4283-b49b-b629e005c074",
        model="glm-4v-vision"
    )

    # Simulando captura de câmera em base64
    with open("capture.jpg", "rb") as f:
        image_base64 = base64.b64encode(f.read()).decode("utf-8")

    result = await provider.analyze_image(
        image_base64=image_base64,
        question="O que você vê nesta imagem?"
    )

    print(f"Análise: {result}")

# Executar
asyncio.run(test_vision())
```

Passo 3: Integrar na Ferramenta MCP

Atualizar `src/mcp/tools/camera/camera.py`:

```
from src.mcp.tools.providers.vllm_provider import VisionAPIProvider

async def take_photo(arguments: dict) -> dict:
    """
    Ferramenta MCP para capturar e analisar foto via Vision API
    """
    try:
        question = arguments.get("question", "O que está nesta imagem?")

        # 1. Capturar imagem da câmera
        camera = Camera.get_instance()
        frame = camera.capture()

        # 2. Converter para base64
        _, buffer = cv2.imencode('.jpg', frame)
        image_base64 = base64.b64encode(buffer).decode('utf-8')

        # 3. Criar provider Vision
        provider = VisionAPIProvider(
            api_key="d66ea037-1b07-4283-b49b-b629e005c074",
            model="glm-4v-vision"
        )

        # 4. Analisar imagem
        analysis = await provider.analyze_image(
            image_base64=image_base64,
            question=question
        )

        return {
            "status": "success",
            "analysis": analysis,
            "timestamp": datetime.now().isoformat()
        }

    except Exception as e:
        return {
            "status": "error",
            "error": str(e)
        }
```

🔐 Segurança e Autenticação

JWT Token

```python
# Gerado ou carregado em app.py
from uuid import uuid4

auth_key = config.get("server", {}).get("auth_key")
if not auth_key:
    auth_key = str(uuid4())  # Gerar se não existir

# Validação em vision_handler.py
class AuthToken:
    def validate_token(self, token):
        # Valida JWT token
        if token != self.expected_token:
            raise Exception("Token inválido")
```

Headers Necessários

```
Authorization: Bearer d66ea037-1b07-4283-b49b-b629e005c074
Content-Type: application/json
```

📊 Arquitetura da Solução

```
┌─────────────────────────────────────────────────────────┐
│              Usuário fala: "Tire uma foto"              │
└─────────────────────────┬───────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│       LLM interpreta comando → Chama take_photo         │
└─────────────────────────┬───────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│    MCP Tool: take_photo(question="...")                 │
│    1. Captura frame da câmera (OpenCV)                  │
│    2. Converte para base64                              │
└─────────────────────────┬───────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│    VisionAPIProvider.analyze_image()                    │
│    - Monta payload JSON                                 │
│    - Envia para Zhipu AI Vision API                     │
│    - Recebe análise detalhada                           │
└─────────────────────────┬───────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│    HTTP POST a https://open.bigmodel.cn/...             │
│    Headers:                                             │
│    - Authorization: Bearer API_KEY                      │
```

```
│    - Content-Type: application/json                    │
│                                                        │
│    Body:                                               │
│    {                                                   │
│      "model": "glm-4v-vision",                         │
│      "messages": [...]                                 │
│    }                                                   │
└──────────────────────┬─────────────────────────────────┘
                       ↓
┌──────────────────────┴─────────────────────────────────┐
│    Zhipu AI processa e retorna análise                 │
│    "Na imagem vejo: <descrição>"                       │
└──────────────────────┬─────────────────────────────────┘
                       ↓
┌──────────────────────┴─────────────────────────────────┐
│    LLM gera resposta para o usuário                    │
│    "Vi uma ..., parece ser ..."                        │
└──────────────────────┬─────────────────────────────────┘
                       ↓
┌──────────────────────┴─────────────────────────────────┐
│    TTS sintetiza voz e envia ao usuário                │
└────────────────────────────────────────────────────────┘
```

☑ Checklist de Implementação

- ☐ Configurar `api_key`: `d66ea037-1b07-4283-b49b-b629e005c074`
- ☐ Configurar `model`: `glm-4v-vision`
- ☐ Configurar `api_url`: `https://open.bigmodel.cn/api/paas/v4/chat/completions`
- ☐ Implementar `VisionAPIProvider` com suporte a async/await
- ☐ Adicionar autenticação JWT em `vision_handler`
- ☐ Testar captura de câmera com OpenCV
- ☐ Testar envio de imagem para Vision API
- ☐ Integrar resposta na ferramenta MCP `take_photo`
- ☐ Testar fluxo completo: câmera → Vision API → resposta de voz

---

🧪 Teste Rápido

```python
import asyncio
import cv2
import base64
from src.mcp.tools.providers.vllm_provider import VisionAPIProvider

async def test():
    # Capturar imagem
    cap = cv2.VideoCapture(0)
    ret, frame = cap.read()
    cap.release()
```

```
    # Converter para base64
    _, buffer = cv2.imencode('.jpg', frame)
    img_base64 = base64.b64encode(buffer).decode('utf-8')

    # Analisar
    provider = VisionAPIProvider(
        api_key="d66ea037-1b07-4283-b49b-b629e005c074"
    )

    result = await provider.analyze_image(
        image_base64=img_base64,
        question="Descreva tudo que você vê em detalhes"
    )

    print("Resultado:", result)

asyncio.run(test())
```

Referências

- **Repositório ESP32**: xiaozhi-esp32-server
- **Arquivo Vision**: `main/xiaozhi-server/core/api/vision_handler.py`
- **API Zhipu**: https://open.bigmodel.cn
- **Modelo**: GLM-4V-Vision

Status: ☑ Configuração correta identificada e documentada
Próximo Passo: Implementar provider e integrar com MCP tools