**Text (sinh văn bản)**

| **Model** | **Mục** | **Giá gốc (USD / 1M tokens)** | **≈ $ / 1K** |
| --- | --- | --- | --- |
| **Gemini 2.5 Pro** | Input (prompt ≤200k) | 1.25 | 0.00125 |
|  | Input (>200k) | 2.50 | 0.00250 |
|  | Output (đã gồm “thinking tokens”) ≤200k | 10.00 | 0.01000 |
|  | Output >200k | 15.00 | 0.01500 |
| **Gemini 2.5 Flash** | Input (text/image/video) | 0.30 | 0.00030 |
|  | Input (audio) | 1.00 | 0.00100 |
|  | Output (text) | 2.50 | 0.00250 |

**Image (sinh hình ảnh)**

| **Model** | **Mục** | **Giá** | **Quy đổi token** |
| --- | --- | --- | --- |
| **Gemini 2.5 Flash Image** | Output 1 ảnh ≤1024×1024 | **$0.039 / ảnh** | 1 ảnh ≈ **1290 tokens** ⇒ tương đương **$30 / 1M tokens** ≈ **$0.00003 / token** |
|  | Input (mô tả bằng text) | **$0.30 / 1M tokens** (giống Flash input) | ≈ $0.00000030 / token |

**Video (sinh video – dùng Veo qua Gemini API)**

| **Model video** | **Giá (USD)** | **Đơn vị** |
| --- | --- | --- |
| **Veo 3.1 Standard** | **$0.40** | / giây video có âm thanh |
| **Veo 3.1 Fast** | **$0.15** | / giây video có âm thanh |

**Audio / TTS (sinh âm thanh giọng nói)**

| **Model** | **Mục** | **Giá gốc (USD / 1M “audio tokens”)** | **≈ $ / 1K** |
| --- | --- | --- | --- |
| **Gemini 2.5 Flash Preview TTS** | Input (text) | 0.50 | 0.00050 |
|  | Output (audio) | **10.00** | 0.01000 |
| **Gemini 2.5 Pro Preview TTS** | Input (text) | 1.00 | 0.00100 |
|  | Output (audio) | **20.00** | 0.02000 |

#### GENERATE AUDIO:

import os

import requests

import base64

import time

import io

from pathlib import Path

from typing import Optional, Tuple, List

import json

# ==================== TEXT-TO-SPEECH (V1) ====================

def text\_to\_speech\_v1(text: str, voice: str, base\_url: str, api\_key: str) -> Tuple[Optional[str], str]:

    """Convert text to speech using Gemini TTS (V1)"""

    try:

        url = f"{base\_url}/gemini/v1beta/models/gemini-2.5-flash-preview-tts:generateContent"

        headers = {

            "x-goog-api-key": api\_key,

            "Content-Type": "application/json"

        }

        payload = {

            "contents": [{

                "parts": [{"text": text}]

            }],

            "generationConfig": {

                "responseModalities": ["AUDIO"],

                "speechConfig": {

                    "voiceConfig": {

                        "prebuiltVoiceConfig": {

                            "voiceName": voice

                        }

                    }

                }

            }

        }

        response = requests.post(url, headers=headers, json=payload, timeout=120)

        response.raise\_for\_status()

        data = response.json()

        audio\_data = data["candidates"][0]["content"]["parts"][0]["inlineData"]["data"]

        # Decode base64 audio (PCM format)

        audio\_bytes = base64.b64decode(audio\_data)

        # Save as WAV file

        temp\_path = f"generated\_audio\_{int(time.time())}.wav"

        # Convert PCM to WAV format

        import wave

        with wave.open(temp\_path, 'wb') as wav\_file:

            wav\_file.setnchannels(1)  # Mono

            wav\_file.setsampwidth(2)  # 16-bit

            wav\_file.setframerate(24000)  # 24kHz

            wav\_file.writeframes(audio\_bytes)

        return temp\_path, "✅ Audio generated successfully!"

    except Exception as e:

        return None, f"❌ Error: {str(e)}"

# ==================== TEXT-TO-SPEECH (Gemini MULTI-SPEAKER) ====================

def text\_to\_speech\_multi\_speaker(text: str, speaker1\_name: str, speaker1\_voice: str,

                                  speaker2\_name: str, speaker2\_voice: str,

                                  base\_url: str, api\_key: str) -> Tuple[Optional[str], str]:

    """Convert text to speech with multiple speakers using Gemini TTS"""

    try:

        url = f"{base\_url}/gemini/v1beta/models/gemini-2.5-flash-preview-tts:generateContent"

        headers = {

            "x-goog-api-key": api\_key,

            "Content-Type": "application/json"

        }

        payload = {

            "contents": [{

                "parts": [{"text": text}]

            }],

            "generationConfig": {

                "responseModalities": ["AUDIO"],

                "speechConfig": {

                    "multiSpeakerVoiceConfig": {

                        "speakerVoiceConfigs": [

                            {

                                "speaker": speaker1\_name,

                                "voiceConfig": {

                                    "prebuiltVoiceConfig": {

                                        "voiceName": speaker1\_voice

                                    }

                                }

                            },

                            {

                                "speaker": speaker2\_name,

                                "voiceConfig": {

                                    "prebuiltVoiceConfig": {

                                        "voiceName": speaker2\_voice

                                    }

                                }

                            }

                        ]

                    }

                }

            }

        }

        response = requests.post(url, headers=headers, json=payload, timeout=120)

        response.raise\_for\_status()

        data = response.json()

        audio\_data = data["candidates"][0]["content"]["parts"][0]["inlineData"]["data"]

        # Decode base64 audio (PCM format)

        audio\_bytes = base64.b64decode(audio\_data)

        # Save as WAV file

        temp\_path = f"generated\_multi\_audio\_{int(time.time())}.wav"

        # Convert PCM to WAV format

        import wave

        with wave.open(temp\_path, 'wb') as wav\_file:

            wav\_file.setnchannels(1)  # Mono

            wav\_file.setsampwidth(2)  # 16-bit

            wav\_file.setframerate(24000)  # 24kHz

            wav\_file.writeframes(audio\_bytes)

        return temp\_path, "✅ Multi-speaker audio generated successfully!"

    except Exception as e:

        return None, f"❌ Error: {str(e)}"

# ==================== TEXT-TO-SPEECH (Gemini) ====================

def text\_to\_speech(text: str, voice: str, base\_url: str, api\_key: str) -> Tuple[Optional[str], str]:

    """Convert text to speech using /audio/speech endpoint"""

    try:

        url = f"{base\_url}/audio/speech"

        headers = {

            "Content-Type": "application/json",

            "Authorization": f"Bearer {api\_key}"

        }

        payload = {

            "model": "gemini-2.5-flash-preview-tts",

            "input": text,

            "voice": voice

        }

        response = requests.post(url, headers=headers, json=payload, timeout=120)

        response.raise\_for\_status()

        # Save audio file

        temp\_path = f"generated\_audio\_{int(time.time())}.mp3"

        with open(temp\_path, "wb") as f:

            f.write(response.content)

        return temp\_path, "✅ Audio generated successfully!"

    except Exception as e:

        return None, f"❌ Error: {str(e)}"

### IMAGEN-4

import requests

import json

import base64

from datetime import datetime

from IPython.display import display, Image as IPyImage

AI\_API\_BASE = "https://api.thucchien.ai/v1"

AI\_API\_KEY = "sk-97rJFHA-3LmTPPI74ahnZw"  # Khuyến nghị: dùng biến môi trường thay vì hard-code

url = f"{AI\_API\_BASE}/images/generations"

headers = {

    "Content-Type": "application/json",

    "Authorization": f"Bearer {AI\_API\_KEY}"

}

data = {

    # 'images': "",

    "model": "imagen-4",

    # "prompt": "Create a full-body photorealistic image of a virtual Asian TV news presenter, inspired by the professional style of Vietnamese national television. The presenter faces the camera (front view), standing tall with a gentle, graceful shift in posture to suggest slight movement. Shoulder-length straight black hair. Cheerful, friendly expression. Wearing a traditional Vietnamese áo dài in bright red with a single large five-point yellow star centered on the chest; flowing panels and simple matching pants. Seamless white studio background. Professional soft key light with subtle rim light, natural skin tones, crisp focus from head to toe. Vertical portrait composition, subject centered, entire body in frame. High dynamic range, ultra-detailed, 4K. No text, no logos, no watermark.",

    "prompt": "Create an empty indoor TV news studio inspired by modern Vietnamese national television sets — no logos or text. Ultra-wide panoramic view that captures the entire studio in a 16:9 frame (3840×2160). Cinematic wide-angle lens (≈16–20 mm full-frame). A luxurious, contemporary stage made of modern materials (brushed metal, glass, polished stone, warm wood accents). A long, curved LED wall runs across the stage in a wide arc; its height is exactly half the stage height. Place a long anchor desk close to the LED wall; the LED wall is clearly taller than the desk. Leave a generous open floor area in front of the desk large enough for multiple people to stand. Clean broadcast lighting with soft overhead grids, subtle rim and floor reflections; neutral white balance. Composition centered and symmetrical with clear leading lines. Photorealistic, HDR, ultra-detailed, 4K. No people, no watermarks, no extra text.",

    "n": 3,                     # <-- int, không phải chuỗi

    "aspect\_ratio": "16:9"

}

response = requests.post(url, headers=headers, data=json.dumps(data))

if response.ok:

    result = response.json()

    items = result["data"]

    for idx, item in enumerate(items, start=1):

        # Lấy ảnh (b64 hoặc url)

        if "b64\_json" in item and item["b64\_json"]:

            img\_bytes = base64.b64decode(item["b64\_json"])

            revised\_prompt = item["revised\_prompt"]

        elif "url" in item and item["url"]:

            # fallback nếu server trả về URL thay vì base64

            img\_resp = requests.get(item["url"])

            img\_resp.raise\_for\_status()

            img\_bytes = img\_resp.content

        else:

            raise RuntimeError("No image content found in response.")

        # Tên file tự động theo timestamp

        ts = datetime.now().strftime("%Y%m%d\_%H%M%S")

        fname = f"generated\_image\_{ts}\_{idx}.png"

        with open(fname, "wb") as f:

            f.write(img\_bytes)

        print(f"Image saved to {fname}")

        # Hiển thị ngay trong notebook

        display(IPyImage(data=img\_bytes))

        print()

        print('revised\_prompt', revised\_prompt)

else:

    print(f"Error: {response.status\_code}")

    try:

        print(response.json())

    except Exception:

        print(response.text)

**### GEMINI 2.5 FLASH**

import os

import json

import base64

from datetime import datetime

from pathlib import Path

import requests

from IPython.display import display, Image as IPyImage

from PIL import Image

from io import BytesIO

from pathlib import Path

from PIL import Image, ImageOps

import matplotlib.pyplot as plt

import base64, io, hashlib

API\_KEY = os.getenv("THUCCHIEN\_API\_KEY", "sk-97rJFHA-3LmTPPI74ahnZw")

URL = "https://api.thucchien.ai/gemini/v1beta/models/gemini-2.5-flash-image-preview:generateContent"

# API\_KEY = os.getenv("THUCCHIEN\_API\_KEY", "AIzaSyA3yyChOko7uZJdU3M0v9d6zUqxvdaDC6Q")

# URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-flash-image:generateContent"

def image\_to\_base64(image: Image.Image, format="PNG") -> str:

    buffered = BytesIO()

    image.save(buffered, format=format)

    img\_str = base64.b64encode(buffered.getvalue()).decode("utf-8")

    return img\_str

def image\_to\_base64(img, fmt="PNG"):

    buf = io.BytesIO()

    img.save(buf, format=fmt)

    return base64.b64encode(buf.getvalue()).decode("utf-8")

def load\_image(path\_str):

    p = Path(path\_str)

    if not p.exists():

        raise FileNotFoundError(f"Not found: {p}")

    # giữ đúng xoay EXIF, chuyển RGBA cho hiển thị sắc nét

    img = ImageOps.exif\_transpose(Image.open(p)).convert("RGBA")

    raw = p.read\_bytes()

    sha12 = hashlib.sha256(raw).hexdigest()[:12]

    print(f"Loaded: {p.name} | size: {img.size} | bytes: {len(raw)/1024:.1f} KB | sha256[:12]={sha12}")

    return img

my\_prompt\_25 = """

Keep all details in photo 1. Only add the woman (image 2) standing in the center of the stage."""

p25 = r"E:\AI THỰC CHIẾN\ai\_ngoc\_linh\_ai\_thuc\_chien\_vong\_2\gemini\_image\_20251024\_031714.png"

image\_input\_p25 = load\_image(p25)

image\_input\_p25\_b64 = image\_to\_base64(image\_input\_p25)

p26 = r"E:\AI THỰC CHIẾN\ai\_ngoc\_linh\_ai\_thuc\_chien\_vong\_2\generated\_image\_20251024\_024055\_2.png"

image\_input\_p26 = load\_image(p26)

image\_input\_p26\_b64 = image\_to\_base64(image\_input\_p26)

# ==== HIỂN THỊ SIDE-BY-SIDE ====

fig, axes = plt.subplots(1, 3, figsize=(12, 4))

for i, image\_input in enumerate([image\_input\_p25, image\_input\_p26]):

    axes[i].imshow(image\_input)

    axes[i].set\_title(f"Image {i}")

    axes[i].axis("off")

plt.tight\_layout()

plt.show()

payload = {

    "contents": [

        {

            "parts": [

                {"text": my\_prompt\_25},

                {

                    "inlineData": {

                        "mimeType": "image/png",

                        "data": image\_input\_p25\_b64,

                    },

                },

                {

                    "inlineData": {

                        "mimeType": "image/png",

                        "data": image\_input\_p26\_b64,

                    },

                },

            ]

        }

    ],

    "generationConfig": {

        'responseModalities': ["Image"],

        'candidateCount': 1,

        "imageConfig": {"aspectRatio": "16:9"}

    },

}

headers = {

    "x-goog-api-key": API\_KEY,  # native Gemini dùng header này

    "Content-Type": "application/json",

}

# Gọi API

resp = requests.post(URL, headers=headers, data=json.dumps(payload))

print("HTTP", resp.status\_code)

# Helper: lưu + hiển thị ảnh từ inlineData

def save\_and\_show\_inline\_image(

    b64\_data: str, mime\_type: str = "image/png", prefix: str = "gemini\_image"

):

    # Chọn phần mở rộng dựa vào mime

    ext\_map = {

        "image/png": "png",

        "image/jpeg": "jpg",

        "image/jpg": "jpg",

        "image/webp": "webp",

    }

    ext = ext\_map.get(mime\_type.lower(), "png")

    img\_bytes = base64.b64decode(b64\_data)

    # Tên file theo thời gian thực

    ts = datetime.now().strftime("%Y%m%d\_%H%M%S")

    fname = f"{prefix}\_{ts}.{ext}"

    Path(fname).write\_bytes(img\_bytes)

    print(f"Saved: {fname}")

    display(IPyImage(data=img\_bytes))

if resp.ok:

    data = resp.json()

    # Trích tất cả inlineData trong candidates -> content.parts

    found = 0

    for cand in data.get("candidates", []):

        content = cand.get("content", {})

        for part in content.get("parts", []):

            inline = part.get("inlineData") or part.get(

                "inline\_data"

            )  # phòng khi viết khác case

            if inline and inline.get("data"):

                save\_and\_show\_inline\_image(

                    b64\_data=inline["data"],

                    mime\_type=inline.get("mimeType", "image/png"),

                    prefix="gemini\_image",

                )

                found += 1

    if found == 0:

        print("No inline image found in response.")

        print(json.dumps(data, indent=2))

else:

    # In lỗi chi tiết để chẩn đoán

    try:

        print(json.dumps(resp.json(), indent=2))

    except Exception:

        print(resp.text)

**### GENERATE VEO 3**

#!/usr/bin/env python3

import json

import os

import time

from typing import Optional

import requests

import utils

BASE\_URL = "https://generativelanguage.googleapis.com/v1beta"

API\_KEY = "<YOUR\_API\_KEY>"

MODEL = "veo-3.1-generate-preview:predictLongRunning"

PROMPT = """A reverent and respectful scene of people lining up to offer incense (Dâng hương) at the memorial site.

The atmosphere is solemn and quiet, focusing on flowers and offerings. Warm, soft lighting, 16:9, documentary style.

Professional Vietnamese female news anchor MC speaks in background: "Các hoạt động dâng hương tại khu di tích các anh hùng liệt sỹ tại Quảng Trị đã thể hiện lòng biết ơn sâu sắc của toàn dân với các anh hùng liệt sỹ đã hy sinh vì độc lập tự do của Tổ quốc..."

"""

REFERENCE\_IMAGES = [

    # "ref/dieubinh.jpg",

    "ref/thoisu.jpg",

]

class VeoVideoGenerator:

    """Complete Veo video generation client using LiteLLM proxy."""

    def \_\_init\_\_(

        self,

        base\_url: str = BASE\_URL,

        api\_key: str = "sk-1234",

    ):

        self.base\_url = base\_url

        self.api\_key = api\_key

        self.headers = {"x-goog-api-key": api\_key, "Content-Type": "application/json"}

    def generate\_video(self, prompt: str) -> Optional[str]:

        print(f"🎬 Generating video with prompt: '{prompt}'")

        url = f"{self.base\_url}/models/{MODEL}"

        payload = {

            "instances": [

                {

                    "prompt": prompt,

                }

            ]

        }

        if REFERENCE\_IMAGES:

            ref = []

            for img in REFERENCE\_IMAGES:

                print(f"🖼️  Using reference image: {img}")

                base64\_img = utils.image\_to\_base64(img)

                ref.append(

                    {

                        "image": {

                            "bytesBase64Encoded": base64\_img,

                            "mimeType": "image/jpeg",

                        },

                        # "referenceType": "asset",

                    }

                )

            payload["instances"][0]["referenceImages"] = ref

        print(f"Payload: {json.dumps(payload, indent=2)[:500]}")

        try:

            response = requests.post(url, headers=self.headers, json=payload)

            response.raise\_for\_status()

            data = response.json()

            operation\_name = data.get("name")

            if operation\_name:

                print(f"✅ Video generation started: {operation\_name}")

                return operation\_name

            else:

                print("❌ No operation name returned")

                print(f"Response: {json.dumps(data, indent=2)}")

                return None

        except requests.RequestException as e:

            print(f"❌ Failed to start video generation: {e}")

            if hasattr(e, "response") and e.response is not None:

                try:

                    error\_data = e.response.json()

                    print(f"Error details: {json.dumps(error\_data, indent=2)}")

                except:

                    print(f"Error response: {e.response.text}")

            return None

    def wait\_for\_completion(

        self, operation\_name: str, max\_wait\_time: int = 600

    ) -> Optional[str]:

        print("⏳ Waiting for video generation to complete...")

        operation\_url = f"{self.base\_url}/{operation\_name}"

        start\_time = time.time()

        poll\_interval = 10  # Start with 10 seconds

        while time.time() - start\_time < max\_wait\_time:

            try:

                print(

                    f"🔍 Polling status... ({int(time.time() - start\_time)}s elapsed)"

                )

                response = requests.get(operation\_url, headers=self.headers)

                response.raise\_for\_status()

                data = response.json()

                # Check for errors

                if "error" in data:

                    print("❌ Error in video generation:")

                    print(json.dumps(data["error"], indent=2))

                    return None

                # Check if operation is complete

                is\_done = data.get("done", False)

                if is\_done:

                    print("🎉 Video generation complete!")

                    try:

                        # Extract video URI from nested response

                        video\_uri = data["response"]["generateVideoResponse"][

                            "generatedSamples"

                        ][0]["video"]["uri"]

                        print(f"📹 Video URI: {video\_uri}")

                        return video\_uri

                    except KeyError as e:

                        print(f"❌ Could not extract video URI: {e}")

                        print("Full response:")

                        print(json.dumps(data, indent=2))

                        return None

                # Wait before next poll, with exponential backoff

                time.sleep(poll\_interval)

                poll\_interval = min(poll\_interval \* 1.2, 30)  # Cap at 30 seconds

            except requests.RequestException as e:

                print(f"❌ Error polling operation status: {e}")

                time.sleep(poll\_interval)

        print(f"⏰ Timeout after {max\_wait\_time} seconds")

        return None

    def download\_video(

        self, video\_uri: str, output\_filename: str = "generated\_video.mp4"

    ) -> bool:

        """

        Download the generated video file.

        Args:

            video\_uri: URI of the video to download (from Google's response)

            output\_filename: Local filename to save the video

        Returns:

            True if download successful, False otherwise

        """

        print(f"⬇️  Downloading video...")

        print(f"Original URI: {video\_uri}")

        # Convert Google URI to LiteLLM proxy URI

        # Example: https://generativelanguage.googleapis.com/v1beta/files/abc123 -> /gemini/download/v1beta/files/abc123:download?alt=media

        if video\_uri.startswith("https://generativelanguage.googleapis.com/"):

            relative\_path = video\_uri.replace(

                "https://generativelanguage.googleapis.com/", ""

            )

        else:

            relative\_path = video\_uri

        # base\_url: https://api.thucchien.ai/gemini/v1beta

        if self.base\_url.endswith("/v1beta"):

            base\_path = self.base\_url.replace("/v1beta", "/download")

        else:

            base\_path = self.base\_url

        litellm\_download\_url = f"{base\_path}/{relative\_path}"

        print(f"Download URL: {litellm\_download\_url}")

        try:

            # Download with streaming and redirect handling

            response = requests.get(

                litellm\_download\_url,

                headers=self.headers,

                stream=True,

                allow\_redirects=True,  # Handle redirects automatically

            )

            response.raise\_for\_status()

            # Save video file

            with open(output\_filename, "wb") as f:

                downloaded\_size = 0

                for chunk in response.iter\_content(chunk\_size=8192):

                    if chunk:

                        f.write(chunk)

                        downloaded\_size += len(chunk)

                        # Progress indicator for large files

                        if downloaded\_size % (1024 \* 1024) == 0:  # Every MB

                            print(

                                f"📦 Downloaded {downloaded\_size / (1024\*1024):.1f} MB..."

                            )

            # Verify file was created and has content

            if os.path.exists(output\_filename):

                file\_size = os.path.getsize(output\_filename)

                if file\_size > 0:

                    print(f"✅ Video downloaded successfully!")

                    print(f"📁 Saved as: {output\_filename}")

                    print(f"📏 File size: {file\_size / (1024\*1024):.2f} MB")

                    return True

                else:

                    print("❌ Downloaded file is empty")

                    os.remove(output\_filename)

                    return False

            else:

                print("❌ File was not created")

                return False

        except requests.RequestException as e:

            print(f"❌ Download failed: {e}")

            if hasattr(e, "response") and e.response is not None:

                print(f"Status code: {e.response.status\_code}")

                print(f"Response headers: {dict(e.response.headers)}")

            return False

    def generate\_and\_download(self, prompt: str, output\_filename: str = None) -> bool:

        # Auto-generate filename if not provided

        if output\_filename is None:

            timestamp = int(time.time())

            safe\_prompt = "".join(

                c for c in prompt[:30] if c.isalnum() or c in (" ", "-", "\_")

            ).rstrip()

            output\_filename = (

                f"result/veo\_video\_{safe\_prompt.replace(' ', '\_')}\_{timestamp}.mp4"

            )

        print("=" \* 60)

        print("🎬 VEO VIDEO GENERATION WORKFLOW")

        print("=" \* 60)

        # Step 1: Generate video

        operation\_name = self.generate\_video(PROMPT)

        if not operation\_name:

            return False

        # Step 2: Wait for completion

        video\_uri = self.wait\_for\_completion(operation\_name)

        if not video\_uri:

            return False

        # Step 3: Download video

        success = self.download\_video(video\_uri, output\_filename)

        if success:

            print("=" \* 60)

            print("🎉 SUCCESS! Video generation complete!")

            print(f"📁 Video saved as: {output\_filename}")

            print("=" \* 60)

        else:

            print("=" \* 60)

            print("❌ FAILED! Video generation or download failed")

            print("=" \* 60)

        return success

def main():

    # Configuration from environment or defaults

    base\_url = os.getenv("LITELLM\_BASE\_URL", BASE\_URL)

    api\_key = os.getenv("LITELLM\_API\_KEY", API\_KEY)

    print("🚀 Starting Veo Video Generation Example")

    print(f"📡 Using LiteLLM proxy at: {base\_url}")

    # Initialize generator

    generator = VeoVideoGenerator(base\_url=base\_url, api\_key=api\_key)

    print(f"🎬 Using prompt: '{PROMPT}'")

    # Generate and download video

    success = generator.generate\_and\_download(PROMPT)

    if success:

        print("✅ Example completed successfully!")

    else:

        print("❌ Example failed!")

        print("🔧 Check your API Configuration")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

### PROMPT 1

A cute 3D cartoon pig character inspired by Peppa Pig. The pig has a simple rounded head with a long pink snout, two small nostrils, and two circular black eyes placed on the same side of the face. It has small round ears on top of the head and rosy pink cheeks. The mouth forms a soft, cheerful smile. The body is small and slightly triangular, wearing a plain red dress. The arms and legs are thin and pink, and there is a short curly pink tail on the back. The style is smooth, toy-like 3D modeling with soft plastic material, bright pastel colors, and clean surface details. Lighting is even and studio-style with gentle reflections. The overall design keeps the simplicity and charm of the original 2D cartoon, translated into a high-quality 3D model suitable for animation or figurine rendering.

### PROMPT 2

A 3D animated Dalmatian puppy with a natural dog body, standing on four legs in a relaxed pose. The puppy wears a bright red firefighter uniform with yellow accents and a small flame badge on the chest, plus a red rescue helmet with a paw emblem. The animation style is simple and stylized — smooth plastic-like texture, no visible fur strands, clean edges, and bright cartoon lighting. The character looks happy and alert, suitable for a children’s 3D cartoon series. The scene has warm daylight and vivid colors, rendered in medium quality, non-photorealistic style.