Gemma Model Enhancements

Overview

This document describes the enhancements made to the Gemma model integration in the Cortex Bridge system, including multimodal capabilities, performance optimizations, and smart model management strategies.

Enhanced GemmaClient Features

1. Multimodal Image Support

The GemmaClient now supports image inputs for vision-language tasks:

```
client = GemmaClient("gemma3n:e4b")
response = client.generate_response(
    prompt="What's in this image?",
    image_path="/path/to/image.jpg"
)
```

Implementation Details: - Images are automatically encoded to base64 for API transmission - Supports common formats (JPG, PNG, etc.) - Uses Ollama's native multimodal API endpoint - Error handling for invalid image paths

2. Custom Prompt Templates

Added flexible prompt formatting with template support:

```
template = "System: You are a helpful assistant.\n{context}\nUser: {prompt}\nAssistant:"
response = client.generate_response(
    prompt="Explain photosynthesis",
    context="User is 10 years old",
    prompt_template=template
)
```

Features: - `{context}` and `{prompt}` placeholder substitution - Backward compatible with existing context formatting - Enables custom conversation formatting

3. Vector Context Integration

Added structured context support for vector database integration:

```
vector_data = {
    "user_preferences": {"style": "educational", "level": "beginner"},
    "related_topics": ["biology", "plants"],
    "conversation_metadata": {"session_type": "learning"}
}

response = client.generate_response(
```

```
    prompt="Tell me about plants",
    vector_context=vector_data
)
```

Benefits: - Separate from conversation history context - JSON structure for metadata and preferences - Enhanced context awareness for better responses

Performance Optimizations

Model Memory Management

For systems with limited VRAM (8GB), we've implemented several optimization strategies:

1. Smart Model Selection File: `smart_model_selector.py`

Intelligently chooses between models to minimize switching overhead:

- **Complexity Analysis**: Keywords and text length determine model needs
- **Switch Throttling**: Prevents frequent switching within 30-second windows
- **Persistence Logic**: Sticks with current model unless switch is critical

```
selector = SmartModelSelector()
optimal_model = selector.get_optimal_model(prompt, context)
```

2. Model Preloading & Warming File: `model_preloader.py`

Strategies to minimize cold-start loading times:

- **Warm-up Requests**: Minimal token generation to load models
- **Background Rotation**: Keep models warm with periodic requests
- **Load Time Benchmarking**: Measure and optimize loading performance

```
preloader = ModelPreloader()
load_time = preloader.warm_model("gemma3n:e4b")  # ~3-5 seconds
```

3. Optimized Client Implementation File: `optimized_gemma_client.py`

Enhanced client with automatic optimization:

- **Explicit Model Unloading**: Frees VRAM immediately when switching
- **Smart Switching Logic**: Combines selector and preloader
- **Performance Monitoring**: Tracks and reports switching times

```
client = OptimizedGemmaClient()
response = client.generate_response_optimized(prompt, context)
```

Model Switching Performance

VRAM Considerations

System VRAM	Model Loading Strategy	Switch Time
16GB+	Keep both models loaded	~0.1s (instant)
8-12GB	Smart switching with unload	~2-3s (optimized)
<8GB	Single model only	~4-5s (standard)

Model Specifications

Model	Size	VRAM Usage	Inference Speed	Best For
gemma3n:e2b	3GB	4.3GB	~1.3s	Quick responses, simple queries
gemma3n:e4b	6GB	5.6GB	~2.0s	Complex reasoning, detailed responses

System-Level Optimizations

Storage Optimizations

1. **NVMe SSD**: 50-80% faster model loading from disk
2. **RAM Caching**: Improved system cache settings for model files
3. **tmpfs Storage**: RAM disk for frequently accessed models

Ollama Configuration

```
# Optimize for memory-constrained systems
export OLLAMA_MAX_LOADED_MODELS=1  # Force single model
export OLLAMA_KEEP_ALIVE=30s       # Faster memory release
export OLLAMA_NUM_PARALLEL=1       # Single-threaded for memory
```

Quantization Strategies

- **Q4_K_M** (Current): Balanced quality/speed
- **Q3_K_M** (Recommended): 30% faster loading, minimal quality loss
- **Q2_K** (Ultra-fast): 50% faster loading, acceptable for simple tasks

Integration with Existing System

Backward Compatibility

All enhancements maintain full backward compatibility:

```
# Existing code continues to work unchanged
response = client.generate_response(prompt, context)

# New features are optional parameters
response = client.generate_response(
    prompt=prompt,
    context=context,  # Existing conversation history
    vector_context=vector_data,  # New structured context
    image_path=image_file,  # New multimodal support
    prompt_template=template  # New formatting control
)
```

Conversation Manager Integration

The existing conversation context system remains unchanged:

1. `conversation_manager.get_conversation_context()` → `context` parameter
2. New `vector_context` adds supplementary structured data
3. Both contexts are properly formatted and combined

Usage Examples

Basic Multimodal Query

```
client = GemmaClient("gemma3n:e4b")
response = client.generate_response(
    "Describe this image in detail",
    image_path="screenshot.png"
)
```

Advanced Context Integration

```
conversation_context = conversation_manager.get_conversation_context()
vector_context = {
    "user_profile": {"expertise": "beginner", "interest": "technology"},
    "session_data": {"previous_topics": ["AI", "programming"]}
}

response = client.generate_response(
    prompt="How does machine learning work?",
    context=conversation_context,  # Chat history
    vector_context=vector_context,  # User preferences & metadata
```

```
    prompt_template="Context: {context}\nUser: {prompt}\nExpert:"
)
```

Performance-Optimized Usage

```
optimized_client = OptimizedGemmaClient()
response = optimized_client.generate_response_optimized(
    prompt="Complex reasoning task",
    context=conversation_context
)
# Automatically selects gemma3n:e4b and manages loading
```

Future Improvements

Potential Enhancements

1. **Predictive Loading**: Analyze conversation patterns to preload likely models
2. **Dynamic Quantization**: Automatically adjust model precision based on VRAM
3. **Multi-GPU Support**: Distribute models across multiple GPUs
4. **Streaming Optimization**: Reduce perceived latency with streaming responses

Performance Targets

- **Target Switch Time**: <2 seconds for 8GB VRAM systems
- **Memory Efficiency**: <90% VRAM utilization with safety margins
- **Response Quality**: Maintain >95% quality vs unoptimized baseline

Testing

Use the provided test script to verify functionality:

```
python test_enhanced_gemma.py
```

This tests all new features: - Basic functionality (backward compatibility) - Custom prompt templates - Vector context integration - Combined parameter usage

Conclusion

These enhancements provide significant improvements to the Gemma model integration:

- **Multimodal capabilities** for vision-language tasks
- **Flexible context management** with templates and vector integration
- **Performance optimizations** for memory-constrained systems
- **Smart model selection** to minimize switching overhead

The system maintains full backward compatibility while providing powerful new capabilities for enhanced AI interactions.