### Enumerating Possible Types for AI Interactions

In PromptLogger (which logs prompts for LLM interactions like "shots" or "console"), we can extend the type parameter to a more comprehensive enumeration of AI interaction categories. This allows for better categorization in logs, especially as you expand to image generation, movie creation, multimodal processing, diffusion models, OCR, and beyond. I recommend defining an enum in PromptLogger.h for type safety and readability.

Extended enumeration of types, covering current text-based operations and future extensions:

* **text**: General text generation (e.g., script breakdown, console commands).
* **image\_generation**: Generating images from prompts (e.g., using Stable Diffusion or DALL-E).
* **movie**: Video or movie generation (e.g., text-to-video models like Sora or RunwayML).
* **multimodal**: Combined text/image/audio/video processing (e.g., models like GPT-4V or Gemini).
* **diffusion**: Specific for diffusion-based models (e.g., image editing or upscaling with Stable Diffusion).
* **ocr**: Optical Character Recognition (e.g., extracting text from images).
* **image\_analysis**: Analyzing images (e.g., captioning, object detection, sentiment analysis).
* **audio\_generation**: Generating audio or speech (e.g., text-to-speech with ElevenLabs or Whisper).
* **video\_analysis**: Analyzing videos (e.g., frame extraction, motion detection).
* **code\_generation**: Generating or analyzing code (e.g., AI-assisted coding).
* **data\_analysis**: Statistical or data processing tasks (e.g., summarizing datasets).
* **other**: Catch-all for uncategorized or custom interactions.

To implement this in code, add an enum to PromptLogger.h:

cpp  
enum class AiType {  
 Text,  
 ImageGeneration,  
 Movie,  
 Multimodal,  
 Diffusion,  
 Ocr,  
 ImageAnalysis,  
 AudioGeneration,  
 VideoAnalysis,  
 CodeGeneration,  
 DataAnalysis,  
 Other  
};

### Including Category/Type in Log Output

To include the category/type in your log output (the JSON file), it's already handled via the "type" key in logData. If you want to separate "category" (e.g., broad group like "generation" or "analysis") from "type" (specific like "image\_generation"), add a new parameter.

Suggested variable names:  
- For the enum/class type: logType (clear, follows "log" prefix for consistency with logData).  
- For a separate category string: logCategory (e.g., "generation" for image/movie, "analysis" for OCR/image\_analysis).

Update logPromptAndResult signature:  
cpp  
void logPromptAndResult(const std::string& category, LogType type, const std::string& prompt, const std::string& result,  
 int tokens, double cost, const QJsonObject& context = QJsonObject());

In the method:  
cpp  
logData["category"] = QString::fromStdString(category);  
logData["type"] = QString::fromStdString(toString(type));

This allows flexible tracking (e.g., category = "generation", type = ImageGeneration).

### Useful Information to Log: Standards and Best Practices for additional enhancements

**Standards and Best Practices**:  
- **Structured Logging**: Use JSON for logs (as you are), enabling easy parsing/searching with tools like ELK Stack or Splunk.  
- **Log Levels**: Use INFO for successful logs, ERROR for failures (as in your code). Add DEBUG for verbose details.  
- **Privacy/Security**: Anonymize sensitive data (e.g., strip user info from prompts). Use secure storage for tokens.  
- **Retention/Rotation**: Rotate logs by size/date (e.g., daily files). Delete old logs to comply with data retention policies (e.g., GDPR).  
- **Monitoring**: Integrate with monitoring tools (e.g., Prometheus for cost/token metrics).  
- **Correlation IDs**: Add unique IDs per session for tracing multi-prompt interactions.  
- **Error Handling**: Always log exceptions/network errors.  
- **Performance**: Log asynchronously to avoid blocking (your network POST is async—good).  
- **Auditing**: For production, log to a central system; include machine ID, app version.

**Useful Information added to Log DB**:  
- Session ID: For grouping related prompts (e.g., multi-shot breakdowns).  
- Correlation ID: UUID for tracing chains of interactions.  
- Environment: OS, hardware (e.g., GPU/CPU).  
- Error Details: If failed, include error code/message.  
- Input/Output Size: Prompt/response length (bytes or chars).  
- Latency Breakdown: Network time, inference time.  
- Model Parameters: Temperature, top-k, etc., if configurable.  
- User Agent: For remote requests (e.g., app name/version).

Variable Names:  
- sessionId (long or QString): For session tracking.  
- correlationId (QUuid): For linking related logs.  
- environment (QString): e.g., "local\_macOS".  
- errorDetails (QJsonObject): If failure occurs.  
- inputSize (int): Prompt length.  
- outputSize (int): Response length.  
- latencyBreakdown (QJsonObject): e.g., {"network": 0.5, "inference": 2.0}.  
- modelParams (QJsonObject): e.g., {"temperature": 0.7}.

Extended logPromptAndResult to include these optionally.