Proposal: Enhancing SBOannotator with LLM Integration & Dynamic Term Retrieval

Personal Information

Name: Jiahui Hu
Email: hu.jiahu@northeastern.edu
Alternative Email: lareinahuwork@gmail.com
GitHub:https://github.com/lareinahu-2023
University: Northeastern University, Master's in Computer Software Engineering
Prior Contribution: https://github.com/draeger-lab/SBOannotator/pull/2

Project Synopsis

This proposal aims to transform SBOannotator from a static, hard-coded tool into a dynamic, intelligent system for annotating SBML models with Systems Biology Ontology terms.After initial discussions with mentor Nantia Leonidou, who provided valuable feedback on my approach, I've developed a comprehensive plan and initial implementation (PR #2 https://github.com/draeger-lab/SBOannotator/pull/2).

By implementing real-time SBO term retrieval, integrating multiple enzymatic data sources, and incorporating LLM-based annotation assistance, this project will significantly enhance both the accuracy and usability of SBOannotator while maintaining its core classification strengths. The addition of a standalone desktop GUI with interactive visualizations will make these powerful annotation capabilities accessible to a broader range of systems biology researchers.

Project Link: https://github.com/nrnb/GoogleSummerOfCode/issues/261

Potential Mentor: Nantia Leonidou - nantia.leonidou@dkfz-heidelberg.de

Project Description

Background

SBOannotator is the first standalone tool that automatically assigns Systems Biology Ontology (SBO) terms to multiple entities of a given SBML (Systems Biology Markup Language) model. Its main strength lies in annotating biochemical reactions within metabolic models, as the correct assignment of precise SBO annotations requires extensive classification. SBO terms play a crucial role in precisely defining the functions of various components within biological models. By assigning these terms, the interpretability and overall understanding of the model are significantly improved.

However, in large-scale models containing thousands of reactions and chemical species, manually assigning SBO terms becomes a highly complex and time-consuming task. Currently, SBOannotator assigns terms that are hard-coded, making it unable to integrate newly introduced ontologies. Additionally, it derives enzymatic information needed for the correct classification of reactions from the BiGG database or a predefined static SQL database.

Problem Statement

The current SBOannotator has three key limitations:

1. **Static SBO Terms:** Hard-coded terms prevent the integration of newly introduced ontologies from the OLS server.
2. **Limited Data Sources:** Reliance on BiGG database restricts access to comprehensive enzymatic information.
3. **Complex User Experience:** The lack of an intuitive interface and visualization tools limits accessibility.

Project Goals

1. Develop a dynamic system to automatically fetch and update SBO terms from the OLS server
2. Expand enzymatic data integration to include KEGG and BRENDA databases
3. Implement an on-premise LLM-based annotation assistant to intelligently suggest SBO terms
4. Create interactive visualizations for reaction networks and annotation statistics
5. Build a standalone desktop GUI with an intuitive user interface

Benefits to the Community

This project will benefit the systems biology community by:

- **Improving Annotation Accuracy:** Access to up-to-date SBO terms and multiple enzymatic data sources will enable more precise annotations.
- **Enhancing Efficiency:** LLM-based suggestions and an intuitive GUI will significantly reduce the time required for annotation.
- **Expanding Accessibility:** The standalone application will make advanced annotation capabilities available to researchers with varying technical expertise.
- **Facilitating Model Comparison:** Visualization tools will allow for better understanding and comparison of model components.

- **Ensuring Sustainability:** Dynamic term retrieval will future-proof the tool against ontology changes.

Initial Implementation Progress

1. Description of What I Have Accomplished:

I've already laid the groundwork for integrating LLM capabilities into SBOannotator by creating the essential foundation components, as demonstrated in my PR #2: https://github.com/draeger-lab/SBOannotator/pull/2.

My initial implementation includes:

1. **Comprehensive Implementation Plan (LLM_ANNOTATION_PLAN.md):**
   - Defined a clear five-phase implementation strategy
   - Created detailed architectural diagram showing component relationships
   - Outlined specific technical approaches for reaction feature extraction
   - Developed prompt engineering strategies for biochemical classification
   - Established evaluation metrics for assessing annotation quality
2. **Modular LLM Provider Interface (llm_interface.py):**
   - Built abstract base class for provider-agnostic LLM integration
   - Implemented provider-specific classes for OpenAI and Anthropic
   - Created reaction feature extraction framework with placeholder methods
   - Designed SBOAnnotationAssistant class to orchestrate the annotation process
   - Developed methods for batch processing entire SBML models
3. **Template Management System (template_manager.py):**
   - Created a Jinja2-based template engine for dynamic prompt generation
   - Implemented reaction-specific template selection logic
   - Developed four specialized templates for different reaction types:
     - General biochemical reactions
     - Transport reactions
     - Enzymatic reactions
     - Exchange/boundary reactions
   - Built template caching system for improved performance

These components establish a solid architectural foundation for the LLM integration, following software engineering best practices including:
- Clear separation of concerns
- Extensible interfaces
- Comprehensive documentation
- Type hints throughout the codebase

2. ScreenShots of My Key Files:

1. LLM_ANNOTATION_PLAN.md

```
# LLM-Based SBO Term Annotation: Implementation Plan

h2: ## Overview

## Project Objectives

1. Create a modular, extensible interface for interacting with different LLM providers
2. Develop intelligent reaction feature extraction for improved term suggestions
3. Design effective prompting strategies for accurate SBO term assignment
4. Integrate the LLM assistant with the existing SBOannotator workflow
5. Evaluate and benchmark the accuracy of LLM-based annotations against existing methods

```

```
## Architecture

The solution follows a modular design with clear separation of concerns:

```
                    ┌──────────────────┐
                    │   SBOannotator   │
                    └─────────┬────────┘
                              │
                              ▼
┌───┐
│   SBOAnnotationAssistant   │
│   │
│ ┌────────────────┐   ┌────────────────┐   │
│ │ Feature   │   │ Response   │   │
│ │ Extraction   │   │ Processing   │   │
│ └────────────────┘   └────────────────┘   │
│   │
│ ┌────────────────┐   ┌────────────────┐   │
│ │ Prompt   │   │ Suggestion   │   │
│ │ Engineering   │   │ Validation   │   │
│ └────────────────┘   └────────────────┘   │
└─────────────────────┬───────────────────────────┘
                      │
                      ▼
┌───┐
│   LLMProvider (Interface)   │
└───────────────┬───────────────┬───────────────┘
                │   │
    ┌───────────▼───┐   ┌───────▼──────────┐
    │ OpenAIProvider│   │ AnthropicProvider │
    └───────────────┘   └──────────────────┘
```
```

Implementation Phases

Phase 1: Foundation (Current PR)

- [x] Design the abstract LLM provider interface
- [x] Create the SBOAnnotationAssistant class with core functionality
- [x] Define placeholder methods for reaction feature extraction and prompt generation
- [x] Outline initial provider implementations for OpenAI and Anthropic

Phase 2: Feature Extraction

- [ ] Implement comprehensive reaction feature extraction:
    - [ ] Identify metabolite patterns (e.g., ATP/ADP for phosphorylation)
    - [ ] Detect compartment changes for transport reactions
    - [ ] Extract EC numbers from reaction annotations
    - [ ] Analyze reaction reversibility and stoichiometry
    - [ ] Process reaction names for keyword indicators

Phase 3: Prompt Engineering

- [ ] Develop effective prompting strategies:
    - [ ] Research optimal prompts for biochemical reaction classification
    - [ ] Include relevant SBO term definitions in context
    - [ ] Structure prompts to encourage specific, accurate responses
    - [ ] Optimize for token efficiency
    - [ ] Format instructions for structured output

Phase 4: Response Processing

- [ ] Create robust response parsing:
    - [ ] Extract recommended SBO terms from LLM output
    - [ ] Handle various response formats gracefully
    - [ ] Parse confidence scores and explanations
    - [ ] Implement fallback logic for ambiguous responses

Phase 5: Integration & Validation

- [ ] Integrate with SBOannotator:
    - [ ] Add LLM-based annotation as optional enhancement
    - [ ] Implement configuration options for LLM settings
    - [ ] Create hybrid approach that combines rule-based and LLM methods
    - [ ] Develop validation metrics to compare LLM suggestions against existing annotations
    - [ ] Implement benchmark tooling to evaluate different LLM models and prompts

Technical Details

Reaction Feature Extraction

The quality of LLM suggestions depends heavily on effective feature extraction. Key features include:

- **Metabolite Patterns**: Identifying characteristic metabolites (ATP/ADP, NAD/NADH, etc.)
- **Compartment Analysis**: Detecting cross-compartment transport
- **EC Number Integration**: Using existing EC annotations to inform suggestions
- **Reaction Properties**: Analyzing reversibility, stoichiometry, and other properties
- **Naming Analysis**: Extracting insights from reaction names (e.g., "kinase" indicating phosphorylation)

Prompt Engineering

Effective prompts must:

1. Provide sufficient context about SBO terms and their meanings
2. Present reaction details in a structured format
3. Guide the LLM toward precise ontological classification
4. Specify response format for easier parsing
5. Include few-shot examples of correct classifications

````
Example prompt structure:
```
You are a Systems Biology expert tasked with assigning SBO terms to biochemical reactions.

Reaction details:
- ID: {reaction_id}
- Name: {reaction_name}
- Reactants: {reactants}
- Products: {products}
- [Additional extracted features]

Relevant SBO terms include:
- SBO:0000176 (Biochemical reaction): General biochemical transformation
- SBO:0000200 (Redox reaction): Involves electron transfer (e.g., NAD/NADH)
- [Additional relevant terms]

Based on these details, what is the most appropriate SBO term for this reaction?
Provide your answer in JSON format with fields: sbo_term, confidence, explanation.
```

### Provider Implementation

For each LLM provider, we need to:

1. Implement authentication and API handling
2. Optimize request parameters (temperature, max tokens, etc.)
3. Manage rate limiting and error handling
4. Process provider-specific response formats

## Future Enhancements

1. **Fine-tuning**: Create specialized models fine-tuned on SBO classification
2. **Batch Processing**: Optimize for efficient batch annotation
3. **Confidence Thresholds**: Implement adaptive confidence thresholds
4. **User Feedback Loop**: Incorporate user corrections to improve suggestions
5. **Term Expansion**: Expand beyond reactions to other model components

## Evaluation

The LLM-based annotation will be evaluated on:

1. **Accuracy**: Agreement with expert-annotated models
2. **Coverage**: Percentage of reactions receiving specific (non-generic) SBO terms
3. **Consistency**: Consistency of annotations across similar reactions
4. **Performance**: Annotation speed and resource usage
5. **Explainability**: Quality of explanations for suggested terms

## Conclusion

This LLM-based annotation assistant represents a significant enhancement to SBOannotator, enabling more accurate and
comprehensive annotation of SBML models. The modular design ensures adaptability to different LLM providers
and future ontological developments.

The current PR establishes the foundation for this work, demonstrating an understanding of
both the SBOannotator codebase and the requirements for effective LLM integration.
````

2. llm_interface.py - Next show the core interface implementation

```python
"""LLM Interface for SBO term annotation assistance.

This module provides an abstract interface for using Large Language Models (LLMs)
to assist in the annotation of SBML models with appropriate SBO terms.
"""



import abc
from typing import Dict, List, Tuple, Optional, Union, Any
from libsbml import SBMLDocument, Model, Reaction


```

```python
class LLMProvider(abc.ABC):  3 usages  lareinahu-2023
    """Abstract base class for LLM provider implementations."""

    @abc.abstractmethod  lareinahu-2023
    def initialize(self, **kwargs) -> None:
        """Initialize the LLM provider with necessary credentials and settings.

        Args:
            **kwargs: Provider-specific initialization parameters
        """
        pass

    @abc.abstractmethod  1 usage  lareinahu-2023
    def generate_completion(self, prompt: str, **kwargs) -> str:
        """Generate a completion from the LLM based on the given prompt.

        Args:
            prompt: The input prompt for the LLM
            **kwargs: Additional provider-specific parameters

        Returns:
            The generated completion text
        """
        pass

    @property  lareinahu-2023
    @abc.abstractmethod
    def provider_name(self) -> str:
        """Return the name of the LLM provider."""
        pass


```

```python
class SBOAnnotationAssistant:  lareinahu-2023
    """High-level interface for LLM-based SBO term annotation assistance."""

    def __init__(self, llm_provider: LLMProvider):  lareinahu-2023
        """Initialize the annotation assistant with an LLM provider.

        Args:
            llm_provider: An implementation of the LLMProvider interface
        """
        self.llm_provider = llm_provider
        self.sbo_cache = {}  # Cache for SBO terms info

    def extract_reaction_features(self, reaction: Reaction) -> Dict[str, Any]:  1 usage  lareinahu-2023
        """Extract relevant features from a reaction for annotation.

        Args:
            reaction: The SBML reaction to extract features from

        Returns:
            Dictionary of extracted features
        """
        # TODO: Implement feature extraction from reaction
        # This should include:
        # - Reaction ID and name
        # - Reactants and products (including compartments)
        # - Presence of specific metabolites (ATP, NAD, etc.)
        # - Compartment changes (for transport reactions)
        # - EC numbers from annotations
        # - Other relevant metadata

        features = {
            "id": reaction.getId(),
            "name": reaction.getName() or reaction.getId(),
            "reversible": reaction.getReversible(),
            # Add more features
        }

        return features

    def format_prompt(self, reaction_features: Dict[str, Any]) -> str:  1 usage  lareinahu-2023
        """Format a prompt for the LLM based on reaction features.

        Args:
            reaction_features: Dictionary of reaction features

        Returns:
            Formatted prompt string
        """
        # TODO: Implement prompt formatting
        # This should create a detailed prompt that:
        # - Provides context about SBO and its importance
        # - Describes the reaction with relevant features
        # - Asks for specific SBO term recommendations
        # - Specifies response format

        prompt = f"""
        [Prompt would be constructed based on reaction_features]
        """

        return prompt
```

```python
    def parse_llm_response(self, response: str) -> Dict[str, Any]:  1 usage  lareinahu-2023
        """Parse the LLM response into structured annotation suggestions.

        Args:
            response: Raw response from the LLM

        Returns:
            Dictionary containing structured annotation suggestions
        """
        # TODO: Implement response parsing
        # This should extract:
        # - Recommended SBO term ID(s)
        # - Confidence scores
        # - Explanations/reasoning
        # - Alternative suggestions

        result = {
            "sbo_term": "SBO:0000000",  # Placeholder
            "confidence": 0.0,
            "alternatives": [],
            "explanation": ""
        }

        return result

```

```python
    def suggest_sbo_term(self, reaction: Reaction) -> Dict[str, Any]:  1 usage  lareinahu-2023
        """Suggest an SBO term for a reaction using the LLM.

        Args:
            reaction: The SBML reaction to annotate

        Returns:
            Dictionary with SBO term suggestion and metadata
        """
        # Extract features
        features = self.extract_reaction_features(reaction)

        # Format prompt
        prompt = self.format_prompt(features)

        # Generate completion
        response = self.llm_provider.generate_completion(prompt)

        # Parse response
        result = self.parse_llm_response(response)

        return result

```

```python
    def batch_process_model(self, model: Model) -> Dict[str, Dict[str, Any]]:  lareinahu-2023
        """Process all reactions in a model and suggest SBO terms.

        Args:
            model: The SBML model to process

        Returns:
            Dictionary mapping reaction IDs to annotation suggestions
        """
        results = {}

        for i in range(model.getNumReactions()):
            reaction = model.getReaction(i)
            suggestion = self.suggest_sbo_term(reaction)
            results[reaction.getId()] = suggestion

        return results

    def validate_suggestion(self, sbo_term: str) -> bool:  lareinahu-2023
        """Validate that an SBO term ID is valid.

        Args:
            sbo_term: SBO term ID to validate

        Returns:
            Boolean indicating if the term is valid
        """
        # TODO: Implement validation against SBO ontology
        return sbo_term.startswith("SBO:")
```

```python
# Example provider implementations (to be implemented in separate modules)

class OpenAIProvider(LLMProvider):
    """OpenAI LLM provider implementation."""

    def initialize(self, api_key: Optional[str] = None, model: str = "gpt-3.5-turbo", **kwargs) -> None:
        """Initialize OpenAI provider.

        Args:
            api_key: OpenAI API key
            model: Model name to use
            **kwargs: Additional parameters
        """
        # TODO: Implement OpenAI provider initialization
        self.api_key = api_key
        self.model = model

    def generate_completion(self, prompt: str, **kwargs) -> str:
        """Generate completion using OpenAI API.

        Args:
            prompt: The input prompt
            **kwargs: Additional parameters

        Returns:
            Generated completion text
        """
        # TODO: Implement OpenAI API call
        return "[OpenAI response placeholder]"

    @property
    def provider_name(self) -> str:
        """Return the provider name."""
        return "OpenAI"

```

```python
class AnthropicProvider(LLMProvider):
    """Anthropic Claude provider implementation."""

    def initialize(self, api_key: Optional[str] = None, model: str = "claude-3.5", **kwargs) -> None:
        """Initialize Anthropic provider.

        Args:
            api_key: Anthropic API key
            model: Model name to use
            **kwargs: Additional parameters
        """
        # TODO: Implement Anthropic provider initialization
        self.api_key = api_key
        self.model = model

    def generate_completion(self, prompt: str, **kwargs) -> str:
        """Generate completion using Anthropic API.

        Args:
            prompt: The input prompt
            **kwargs: Additional parameters

        Returns:
            Generated completion text
        """
        # TODO: Implement Anthropic API call
        return "[Anthropic response placeholder]"

    @property
    def provider_name(self) -> str:
        """Return the provider name."""
        return "Anthropic"

```

```python
# Example usage
if __name__ == "__main__":
    print("LLM Interface for SBO term annotation")
    print("This module defines interfaces for LLM-assisted annotation")

    # Example usage would be:
    # provider = OpenAIProvider()
    # provider.initialize(api_key="your-api-key")
    # assistant = SBOAnnotationAssistant(provider)
    # model = readSBML("path/to/model.xml").getModel()
    # results = assistant.batch_process_model(model)
```

3. template_manager.py - Finally show the template system

```python
"""Template Manager for LLM-based SBO Annotation.

This module provides a template management system for loading and rendering
prompt templates used in LLM-based annotation.
"""

__author__ = 'Jiahui Hu'

import os
import json
from typing import Dict, Any, Optional
from pathlib import Path
import jinja2


class TemplateManager:  1 usage  lareinahu-2023
    """Manager for loading and rendering templates for LLM prompts."""

    def __init__(self, templates_dir: Optional[str] = None) -> None:  lareinahu-2023
        """Initialize the template manager.

        Args:
            templates_dir: Directory containing template files (default: 'templates')
        """
        # Use provided directory or default to a 'templates' directory
        self.templates_dir = templates_dir or os.path.join(
            os.path.dirname(os.path.abspath(__file__)), 'templates'
        )

        # Ensure templates directory exists
        os.makedirs(self.templates_dir, exist_ok=True)

        # Set up Jinja environment
        self.env = jinja2.Environment(
            loader=jinja2.FileSystemLoader(self.templates_dir),
            trim_blocks=True,
            lstrip_blocks=True,
            autoescape=False
        )

        # Cache for loaded templates
        self._template_cache = {}

```

```python
    def list_templates(self) -> list:  1 usage  lareinahu-2023
        """List available templates in the templates directory.

        Returns:
            List of template names
        """
        return [f for f in os.listdir(self.templates_dir)
                if f.endswith(('.txt', '.j2', '.jinja', '.tmpl'))]

    def render_template(self, template_name: str, context: Dict[str, Any]) -> str:  lareinahu-2023
        """Render a template with the given context.

        Args:
            template_name: Name of the template file
            context: Dictionary of variables to use in template rendering

        Returns:
            The rendered template string

        Raises:
            FileNotFoundError: If the template doesn't exist
        """
        # Load template (with caching)
        if template_name not in self._template_cache:
            try:
                self._template_cache[template_name] = self.env.get_template(template_name)
            except jinja2.exceptions.TemplateNotFound:
                raise FileNotFoundError(f"Template not found: {template_name}")

        template = self._template_cache[template_name]

        # Render with provided context
        return template.render(**context)

```

```python
    def load_template_from_string(self, template_string: str) -> jinja2.Template:  1 usage  lareinahu-2023
        """Load a template from a string.

        Args:
            template_string: The template string to load

        Returns:
            Jinja Template object
        """
        return self.env.from_string(template_string)

    def render_string_template(self, template_string: str, context: Dict[str, Any]) -> str:  lareinahu-2023
        """Render a template string with the given context.

        Args:
            template_string: Template string to render
            context: Dictionary of variables to use in template rendering

        Returns:
            The rendered template string
        """
        template = self.load_template_from_string(template_string)
        return template.render(**context)

```

```python
    def save_template(self, template_name: str, content: str) -> str:  lareinahu-2023
        """Save a template to the templates directory.

        Args:
            template_name: Name to save the template as
            content: Template content

        Returns:
            Path to the saved template
        """
        # Add extension if not present
        if not any(template_name.endswith(ext) for ext in ('.txt', '.j2', '.jinja', '.tmpl')):
            template_name += '.j2'

        template_path = os.path.join(self.templates_dir, template_name)

        with open(template_path, 'w') as f:
            f.write(content)

        # Clear cache for this template if it exists
        if template_name in self._template_cache:
            del self._template_cache[template_name]

        return template_path

```

```python
    def create_default_templates(self) -> None:  1 usage  lareinahu-2023
        """Create default templates if they don't exist."""
        # Define default templates for different reaction types
        default_templates = {
            'reaction_base.j2': """
You are a Systems Biology expert tasked with assigning SBO (Systems Biology Ontology) terms to biochemical reactions.

Reaction details:
- ID: {{ reaction.id }}
- Name: {{ reaction.name }}
- Reversible: {{ reaction.reversible }}
- Reactants: {{ reaction.reactants|join(', ') }}
- Products: {{ reaction.products|join(', ') }}
{% if reaction.ec_numbers %}
- EC Numbers: {{ reaction.ec_numbers|join(', ') }}
{% endif %}
{% if reaction.compartments %}
- Compartments: {{ reaction.compartments|join(', ') }}
{% endif %}

Relevant SBO terms include:
- SBO:0000176 (Biochemical reaction): General biochemical transformation
- SBO:0000200 (Redox reaction): Involves electron transfer (e.g., NAD/NADH)
- SBO:0000216 (Phosphorylation): Transfer of phosphate groups (look for ATP/ADP)
- SBO:0000655 (Transport reaction): Movement across compartments
- SBO:0000627 (Exchange reaction): Exchange with environment
- SBO:0000629 (Biomass production): Overall cell growth

Based on these details, what is the most appropriate SBO term for this reaction?
Provide your answer in JSON format with fields: sbo_term, confidence, explanation.
""",

            'transport_reaction.j2': """
You are a Systems Biology expert tasked with classifying transport reactions in the Systems Biology Ontology.

```

```
Transport Reaction details:
- ID: {{ reaction.id }}
- Name: {{ reaction.name }}
- Reversible: {{ reaction.reversible }}
- Reactants: {{ reaction.reactants|join(', ') }}
- Products: {{ reaction.products|join(', ') }}
- Source compartment: {{ reaction.source_compartment }}
- Target compartment: {{ reaction.target_compartment }}
{% if reaction.metabolites_transported %}
- Metabolites transported: {{ reaction.metabolites_transported|join(', ') }}
{% endif %}

Transport reaction SBO terms include:
- SBO:0000655 (Transport reaction): General transport
- SBO:0000657 (Active transport): Energy-requiring transport (ATP, GTP involved)
- SBO:0000658 (Passive transport): No energy required
- SBO:0000659 (Antiporter): Exchange of molecules in opposite directions
- SBO:0000660 (Symporter): Transport of molecules in same direction
- SBO:0000654 (Co-transport): Transport of multiple species

Based on these details, what is the most appropriate SBO term for this reaction?
Provide your answer in JSON format with fields: sbo_term, confidence, explanation.
""",

            'enzymatic_reaction.j2': """
You are a Systems Biology expert tasked with classifying enzymatic reactions using the Systems Biology Ontology.

Enzymatic Reaction details:
- ID: {{ reaction.id }}
- Name: {{ reaction.name }}
- EC Numbers: {{ reaction.ec_numbers|join(', ') }}
- Reversible: {{ reaction.reversible }}
- Reactants: {{ reaction.reactants|join(', ') }}
- Products: {{ reaction.products|join(', ') }}

Relevant SBO terms based on EC classification:
- SBO:0000200 (Oxidoreductase): EC 1.*
- SBO:0000402 (Transferase): EC 2.*
- SBO:0000376 (Hydrolase): EC 3.*
- SBO:0000211 (Lyase): EC 4.*
- SBO:0000377 (Isomerase): EC 5.*
- SBO:0000695 (Ligase): EC 6.*
- SBO:0000185 (Translocase): EC 7.*

More specific SBO terms include:
- SBO:0000216 (Phosphorylation): Transfer of phosphate groups
- SBO:0000217 (Glycosylation): Addition of glycosyl groups
- SBO:0000399 (Decarboxylation): Removal of carboxyl group
- SBO:0000400 (Decarbonylation): Removal of carbonyl group
- SBO:0000401 (Deamination): Removal of amino group

Based on these details, what is the most appropriate SBO term for this reaction?
Provide your answer in JSON format with fields: sbo_term, confidence, explanation.
""",

            'exchange_reaction.j2': """
You are a Systems Biology expert tasked with classifying exchange and boundary reactions in the Systems Biology Ontology
```

```
Exchange Reaction details:
- ID: {{ reaction.id }}
- Name: {{ reaction.name }}
- Reversible: {{ reaction.reversible }}
- Reactants: {{ reaction.reactants|join(', ') }}
- Products: {{ reaction.products|join(', ') }}

Boundary reaction SBO terms include:
- SBO:0000627 (Exchange reaction): Exchange with environment (often prefixed with EX_)
- SBO:0000628 (Demand reaction): Pure consumption (often prefixed with DM_)
- SBO:0000632 (Sink reaction): Pure production (often prefixed with SK_)
- SBO:0000629 (Biomass production): Overall cell growth (contains "biomass" in name)
- SBO:0000630 (ATP energy): Non-growth associated maintenance

Based on these details, what is the most appropriate SBO term for this reaction?
Provide your answer in JSON format with fields: sbo_term, confidence, explanation.
"""
        }

        for template_name, content in default_templates.items():
            template_path = os.path.join(self.templates_dir, template_name)
            if not os.path.exists(template_path):
                with open(template_path, 'w') as f:
                    f.write(content.strip())

```

```python
    def get_template_for_reaction(self, reaction_features: Dict[str, Any]) -> str:  1 usage  lareinahu-2023
        """Select the most appropriate template for a reaction based on its features.

        Args:
            reaction_features: Dictionary of reaction features

        Returns:
            Template name to use for this reaction
        """
        # Determine the appropriate template based on reaction features
        if 'exchange' in reaction_features.get('id', '').lower() or 'ex_' in reaction_features.get('id', '').lower():
            return 'exchange_reaction.j2'
        elif len(reaction_features.get('compartments', [])) > 1:
            return 'transport_reaction.j2'
        elif reaction_features.get('ec_numbers'):
            return 'enzymatic_reaction.j2'
        else:
            return 'reaction_base.j2'
```

```python
# Example usage
if __name__ == "__main__":
    # Example usage
    template_manager = TemplateManager()

    # Create default templates if they don't exist
    template_manager.create_default_templates()

    print("Available templates:")
    for template in template_manager.list_templates():
        print(f"- {template}")

    # Example reaction features
    example_reaction = {
        "id": "R_GAPD",
        "name": "Glyceraldehyde-3-phosphate dehydrogenase",
        "reversible": True,
        "reactants": ["M_g3p_c", "M_nad_c", "M_pi_c"],
        "products": ["M_13dpg_c", "M_h_c", "M_nadh_c"],
        "ec_numbers": ["1.2.1.12"],
        "compartments": ["c"]
    }

    # Get appropriate template for this reaction
    template_name = template_manager.get_template_for_reaction(example_reaction)
    print(f"\nSelected template for reaction {example_reaction['id']}: {template_name}")

    # Example rendering (commented out since files might not exist yet)
    # rendered = template_manager.render_template(template_name, {"reaction": example_reaction})
    # print("\nRendered template:")
    # print(rendered)
```

Technical Implementation

1. Dynamic SBO Term Retrieval System

- Implement Python API client using the requests library to fetch SBO terms from OLS server
- Develop SQLite-based caching mechanism for offline access with automatic update checking at startup
- Create differential update system to minimize bandwidth and processing overhead
- Implement local cache fallback for reliability when server access is unavailable

2. Enzymatic Data Integration Framework

- Build modular data provider system using the factory pattern for extensibility
- Develop adapters for KEGG and BRENDA databases
- Implement unified query interface via adapter pattern to standardize data access
- Create data normalization layer to resolve inconsistencies between sources

3. LLM-based Annotation Assistant

- Integrate pre-trained DistilBERT model (distilbert-base-uncased) using Hugging Face's Transformers library
- Fine-tune on a dataset of manually annotated SBML models provided by the mentor team
- Implement confidence scoring system for annotation suggestions
- Provide dual modes: automatic application of high-confidence suggestions and user selection from multiple options

4. Visualization Component

- Implement interactive reaction network visualization using Plotly and NetworkX
- Develop before/after annotation comparison views
- Create coverage analysis dashboard for model quality assessment

5. Standalone Desktop GUI

- Develop cross-platform application using PyQt6
- Design tabbed interface with drag-and-drop functionality
- Create annotation review interface with filtering capabilities

Detailed Technical Approach

LLM Implementation Details

I will implement the annotation assistant using DistilBERT (distilbert-base-uncased), a lightweight yet powerful language model that balances performance with resource requirements. Key implementation details include:

1. **Training Data:**
    - Use a corpus of 20-30 manually annotated SBML models (to be provided by mentor)
    - Supplement with existing SBO term definitions and relationships

    - Implement data augmentation techniques to enhance training set size
2. **Fine-tuning Approach:**

    - Focus on text classification rather than full model retraining
    - Use a sequence classification head on top of DistilBERT
    - Train on a modest-sized GPU (e.g., Google Colab or university resources)
3. **Resource Management:**

    - Implement model quantization to reduce memory footprint
    - Use gradient accumulation to enable training with limited GPU memory
    - Cache inference results to improve runtime performance
4. **Fallback Strategy:**

    - If fine-tuning proves too resource-intensive, implement a simpler semantic matching approach using pre-trained embeddings
    - Use cosine similarity between SBO term descriptions and reaction characteristics

Testing Methodology

I will implement a comprehensive testing approach using:

1. **Test Data:**

    - Cross-validation set of 5-10 manually annotated SBML models (separate from training data)
    - Synthetic test cases for edge-case handling
    - Large-scale models to test performance and scalability
2. **Validation Metrics:**

    - Precision, recall, and F1 score for annotation accuracy
    - Coverage percentage of model elements
    - Processing time for various model sizes
    - User satisfaction metrics from beta testers
3. **Continuous Testing:**

    - Unit tests for each component
    - Integration tests for the complete workflow
    - UI testing for the desktop application

Risk Assessment and Mitigation

Risk	Probability	Impact	Mitigation Strategy
LLM training requires excessive resources	Medium	High	Use smaller pre-trained models; implement simpler semantic matching as fallback

OLS API integration is more complex than anticipated	Low	Medium	Focus on core term retrieval first; implement incremental features
Integration of multiple enzymatic databases creates conflicting data	Medium	Medium	Develop robust normalization rules; provide clear conflict resolution UI
GUI development exceeds time allocation	Medium	Medium	Start with minimal viable UI and enhance incrementally; prioritize functionality over aesthetics
Poor performance with large-scale models	Low	High	Implement lazy loading and processing; optimize critical path algorithms

Implementation Timeline (12 Weeks)

Community Bonding Period (2 weeks before official start)

- Set up development environment and communication channels
- Review existing codebase in detail
- Collect and analyze test SBML models
- Define API specifications in collaboration with mentor

Week 1-2: Dynamic SBO Term Retrieval System

- Implement Python API client for OLS server
- Develop SQLite caching mechanism
- Create automatic update checker
- **Milestone:** Working term retrieval with caching

Week 3-4: Enzymatic Data Integration Framework

- Build modular data provider architecture
- Implement KEGG database adapter
- Develop BRENDA database adapter
- **Milestone:** Integrated enzymatic data retrieval
- **Mentor Checkpoint:** Review data integration approach and results

Week 5-6: LLM-based Annotation Assistant (Phase 1)

- Set up Hugging Face Transformers integration
- Prepare training data with mentor assistance
- Implement model training pipeline
- **Milestone:** Initial LLM model with basic prediction capability
- **Mentor Checkpoint:** Evaluate initial model performance

Week 7-8: LLM-based Annotation Assistant (Phase 2) & Visualization

- Refine model based on mentor feedback
- Implement confidence scoring system
- Create basic reaction network visualization
- **Milestone:** Working LLM suggestion system with visualization
- **Mentor Checkpoint:** Review annotation accuracy and visualization

Week 9-10: Standalone Desktop GUI

- Develop cross-platform application shell
- Implement tabbed interface
- Create annotation review interface
- **Milestone:** Functional GUI with core features
- **Mentor Checkpoint:** Usability review and feedback

Week 11-12: Integration, Testing & Documentation

- Integrate all components
- Implement comprehensive testing
- Create user documentation
- Fix bugs and optimize performance
- **Final Milestone:** Complete working application
- **Final Mentor Review:** Comprehensive project evaluation

Mentor Collaboration Plan

I plan to maintain regular communication with my mentor throughout the project:

- **Weekly Progress Reports:** Brief written updates on completed tasks and current challenges
- **Bi-weekly Video Meetings:** Detailed discussion of progress, demonstrations, and technical guidance
- **Code Review Process:** Submit PRs for each completed component for mentor review
- **Critical Decision Points:** Specific checkpoints (noted in timeline) where mentor input is crucial for next steps

Contingency Planning

If time constraints become an issue, I will prioritize features in this order:

1. **Core Functionality:**
    - Dynamic SBO term retrieval (highest priority)
    - Basic enzymatic data integration
    - Simple GUI for accessibility

2. **Enhanced Features:**
    - Advanced LLM integration
    - Multiple data source integration
    - Comprehensive visualization

If the LLM component proves particularly challenging, I will implement a simplified version using pre-trained embeddings and semantic matching rather than full model fine-tuning.

Deliverables

Required Deliverables (Will Definitely Complete)

1. Dynamic SBO term retrieval system with caching mechanism
2. KEGG and BRENDA database adapters for enzymatic data
3. Basic standalone GUI application
4. Comprehensive documentation and tests

Stretch Deliverables (Will Complete If Time Permits)

1. Advanced LLM-based annotation with high-confidence prediction
2. Interactive visualization dashboard
3. Performance optimizations for large-scale models
4. Video tutorials and demonstrations

Qualifications

As a Master's student in Computer Software Engineering at Northeastern University, I bring relevant expertise to this project:

- Proven Contribution to SBOannotator: I've already submitted a functional PR (#2: https://github.com/draeger-lab/SBOannotator/pull/2) implementing the LLM integration foundation
- Experience in NLP system development with fuzzy matching techniques
- Proficiency in API integration and distributed data processing
- UI development skills with PyQt
- Coursework in "Theory & Practice of AI Gen Model" providing insights into effective LLM implementation
- Strong Python programming skills with experience in scientific computing libraries

Related Work

My proposed enhancements build upon existing work in systems biology annotation tools while addressing specific gaps:

- Existing SBO term annotation tools lack dynamic updating and intelligent suggestions

- Current SBML editors provide limited visualization capabilities for annotation coverage
- Few tools integrate multiple enzymatic data sources for comprehensive annotation

Conclusion

This project represents a significant advancement for SBOannotator, transforming it from a tool with static, hard-coded functionality into a dynamic, intelligent system capable of adapting to evolving ontologies and leveraging multiple data sources. I am committed to delivering a realistic, well-tested implementation that maintains SBOannotator's core classification strengths while addressing its current limitations, with clear prioritization and contingency plans to ensure successful completion within the GSoC timeframe.