PRIMARY KEY(session\_id, user\_id)

);

– Security scan results CREATE TABLE security\_scan\_results ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, version\_id UUID REFERENCES template\_versions(id), scan\_type VARCHAR(50) NOT NULL, – ‘pii’, ‘sensitive\_data’, ‘policy\_violation’ severity VARCHAR(20) NOT NULL, – ‘low’, ‘medium’, ‘high’, ‘critical’ finding\_type VARCHAR(100) NOT NULL, finding\_details JSONB NOT NULL, is\_resolved BOOLEAN DEFAULT FALSE, resolved\_by UUID, resolved\_at TIMESTAMP, scanned\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

INDEX(template\_id, severity),  
INDEX(scan\_type, scanned\_at)

);

– Workflow states CREATE TABLE workflow\_states ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, version\_id UUID REFERENCES template\_versions(id), workflow\_type VARCHAR(50) NOT NULL, – ‘review’, ‘approval’, ‘deployment’ current\_state VARCHAR(50) NOT NULL, assigned\_to UUID, assigned\_at TIMESTAMP, due\_date TIMESTAMP, completed\_at TIMESTAMP, workflow\_data JSONB DEFAULT ‘{}’, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

INDEX(assigned\_to, current\_state),  
INDEX(workflow\_type, current\_state)

);

– Audit log CREATE TABLE audit\_log ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), entity\_type VARCHAR(50) NOT NULL, – ‘template’, ‘version’, ‘experiment’ entity\_id UUID NOT NULL, action VARCHAR(100) NOT NULL, user\_id UUID NOT NULL, timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, ip\_address INET, user\_agent TEXT, changes JSONB, – Before/after values metadata JSONB DEFAULT ‘{}’,

INDEX(entity\_type, entity\_id),  
INDEX(user\_id, timestamp),  
INDEX(timestamp)

);

---  
  
## Pseudocode  
  
### Main Prompt Management Workflow

ALGORITHM AdvancedPromptManagementWorkflow INPUT: user\_request (action\_type, template\_data, user\_context) OUTPUT: management\_result (success, data, metadata)

BEGIN // Step 1: Authentication and authorization user\_auth = AUTHENTICATE\_USER(user\_request.auth\_token) IF NOT user\_auth.is\_valid THEN RETURN ERROR(“Authentication failed”) END IF

permissions = CHECK\_USER\_PERMISSIONS(user\_auth.user\_id, user\_request.action\_type, user\_request.template\_id)  
IF NOT permissions.has\_access THEN  
 RETURN ERROR("Insufficient permissions")  
END IF  
  
// Step 2: Route to appropriate handler based on action type  
SWITCH user\_request.action\_type  
 CASE "create\_template":  
 result = CREATE\_TEMPLATE\_WORKFLOW(user\_request.template\_data, user\_auth.user\_id)  
 CASE "update\_template":  
 result = UPDATE\_TEMPLATE\_WORKFLOW(user\_request.template\_data, user\_auth.user\_id)  
 CASE "start\_ab\_test":  
 result = START\_AB\_TEST\_WORKFLOW(user\_request.experiment\_config, user\_auth.user\_id)  
 CASE "deploy\_template":  
 result = DEPLOY\_TEMPLATE\_WORKFLOW(user\_request.deployment\_config, user\_auth.user\_id)  
 CASE "collaborate\_edit":  
 result = COLLABORATIVE\_EDIT\_WORKFLOW(user\_request.edit\_operation, user\_auth.user\_id)  
 DEFAULT:  
 RETURN ERROR("Unsupported action type")  
END SWITCH  
  
// Step 3: Log audit trail  
RECORD\_AUDIT\_EVENT(  
 entity\_type = "template\_management",  
 action = user\_request.action\_type,  
 user\_id = user\_auth.user\_id,  
 result = result,  
 metadata = user\_request.metadata  
)  
  
RETURN result

END

FUNCTION CREATE\_TEMPLATE\_WORKFLOW(template\_data, user\_id) BEGIN // Step 1: Validate template structure validation\_result = VALIDATE\_TEMPLATE\_STRUCTURE(template\_data) IF NOT validation\_result.is\_valid THEN RETURN ERROR(“Template validation failed”, validation\_result.errors) END IF

// Step 2: Security and compliance scanning  
security\_scan = PERFORM\_SECURITY\_SCAN(template\_data.content)  
compliance\_check = CHECK\_COMPLIANCE\_POLICIES(template\_data.content, template\_data.category)  
  
IF security\_scan.has\_violations OR compliance\_check.has\_violations THEN  
 violations = MERGE\_VIOLATIONS(security\_scan.violations, compliance\_check.violations)  
 RETURN ERROR("Security/Compliance violations detected", violations)  
END IF  
  
// Step 3: Extract and validate template variables  
template\_variables = EXTRACT\_TEMPLATE\_VARIABLES(template\_data.content)  
variable\_validation = VALIDATE\_TEMPLATE\_VARIABLES(template\_variables)  
  
IF NOT variable\_validation.is\_valid THEN  
 RETURN ERROR("Invalid template variables", variable\_validation.errors)  
END IF  
  
// Step 4: Create template record  
template = Template(  
 id = GENERATE\_UUID(),  
 name = template\_data.name,  
 description = template\_data.description,  
 category = template\_data.category,  
 content = template\_data.content,  
 variables = template\_variables,  
 tags = template\_data.tags,  
 folder\_id = template\_data.folder\_id,  
 created\_by = user\_id,  
 created\_at = CURRENT\_TIMESTAMP(),  
 status = "draft"  
)  
  
// Step 5: Save to repository  
saved\_template = SAVE\_TEMPLATE\_TO\_REPOSITORY(template)  
  
// Step 6: Create initial version  
initial\_version = CREATE\_INITIAL\_VERSION(  
 template\_id = saved\_template.id,  
 content = template.content,  
 user\_id = user\_id,  
 commit\_message = "Initial template creation"  
)  
  
// Step 7: Index for search  
INDEX\_TEMPLATE\_FOR\_SEARCH(saved\_template)  
  
// Step 8: Set up default workflow if applicable  
IF template\_data.requires\_approval THEN  
 INITIATE\_APPROVAL\_WORKFLOW(saved\_template.id, user\_id)  
END IF  
  
// Step 9: Send notifications  
SEND\_TEMPLATE\_CREATION\_NOTIFICATIONS(saved\_template, user\_id)  
  
RETURN TemplateCreationResult(  
 template = saved\_template,  
 version = initial\_version,  
 security\_scan\_results = security\_scan,  
 compliance\_status = compliance\_check  
)

END

FUNCTION START\_AB\_TEST\_WORKFLOW(experiment\_config, user\_id) BEGIN // Step 1: Validate experiment configuration validation\_result = VALIDATE\_EXPERIMENT\_CONFIG(experiment\_config) IF NOT validation\_result.is\_valid THEN RETURN ERROR(“Experiment configuration invalid”, validation\_result.errors) END IF

// Step 2: Check template permissions and status  
template = GET\_TEMPLATE(experiment\_config.template\_id)  
IF template.status != "active" THEN  
 RETURN ERROR("Template must be active to run experiments")  
END IF  
  
permissions = CHECK\_TEMPLATE\_EXPERIMENT\_PERMISSIONS(template.id, user\_id)  
IF NOT permissions.can\_run\_experiments THEN  
 RETURN ERROR("Insufficient permissions to run experiments")  
END IF  
  
// Step 3: Validate template versions  
control\_version = GET\_TEMPLATE\_VERSION(experiment\_config.control\_version\_id)  
treatment\_versions = []  
  
FOR version\_id IN experiment\_config.treatment\_version\_ids DO  
 treatment\_version = GET\_TEMPLATE\_VERSION(version\_id)  
 IF treatment\_version IS NULL THEN  
 RETURN ERROR(f"Treatment version {version\_id} not found")  
 END IF  
 treatment\_versions.APPEND(treatment\_version)  
END FOR  
  
// Step 4: Calculate required sample size  
sample\_size = CALCULATE\_REQUIRED\_SAMPLE\_SIZE(  
 baseline\_rate = experiment\_config.expected\_baseline\_rate,  
 minimum\_detectable\_effect = experiment\_config.minimum\_detectable\_effect,  
 statistical\_power = experiment\_config.statistical\_power,  
 significance\_level = experiment\_config.significance\_level  
)  
  
// Step 5: Validate traffic split configuration  
IF SUM(experiment\_config.traffic\_split.values()) != 100 THEN  
 RETURN ERROR("Traffic split must sum to 100%")  
END IF  
  
// Step 6: Check for conflicting experiments  
conflicting\_experiments = CHECK\_FOR\_CONFLICTING\_EXPERIMENTS(  
 template\_id = experiment\_config.template\_id,  
 start\_date = experiment\_config.start\_date,  
 end\_date = experiment\_config.end\_date  
)  
  
IF conflicting\_experiments.exist THEN  
 RETURN ERROR("Conflicting experiments detected", conflicting\_experiments.details)  
END IF  
  
// Step 7: Create experiment record  
experiment = Experiment(  
 id = GENERATE\_UUID(),  
 name = experiment\_config.name,  
 description = experiment\_config.description,  
 template\_id = experiment\_config.template\_id,  
 control\_version\_id = experiment\_config.control\_version\_id,  
 treatment\_versions = experiment\_config.treatment\_version\_ids,  
 traffic\_split = experiment\_config.traffic\_split,  
 success\_metrics = experiment\_config.success\_metrics,  
 start\_date = experiment\_config.start\_date,  
 end\_date = experiment\_config.end\_date,  
 required\_sample\_size = sample\_size,  
 status = "created",  
 created\_by = user\_id,  
 created\_at = CURRENT\_TIMESTAMP()  
)  
  
// Step 8: Save experiment  
saved\_experiment = SAVE\_EXPERIMENT\_TO\_REPOSITORY(experiment)  
  
// Step 9: Set up traffic routing if experiment starts immediately  
IF experiment\_config.start\_immediately THEN  
 experiment\_start\_result = START\_EXPERIMENT\_EXECUTION(saved\_experiment.id)  
 saved\_experiment.status = "running"  
 saved\_experiment.actual\_start\_date = CURRENT\_TIMESTAMP()  
 UPDATE\_EXPERIMENT\_STATUS(saved\_experiment)  
END IF  
  
// Step 10: Set up monitoring and alerts  
SETUP\_EXPERIMENT\_MONITORING(  
 experiment\_id = saved\_experiment.id,  
 monitoring\_config = experiment\_config.monitoring\_config  
)  
  
RETURN ExperimentCreationResult(  
 experiment = saved\_experiment,  
 required\_sample\_size = sample\_size,  
 estimated\_duration = ESTIMATE\_EXPERIMENT\_DURATION(sample\_size, experiment\_config.expected\_traffic),  
 monitoring\_dashboard\_url = GET\_EXPERIMENT\_DASHBOARD\_URL(saved\_experiment.id)  
)

END

FUNCTION COLLABORATIVE\_EDIT\_WORKFLOW(edit\_operation, user\_id) BEGIN // Step 1: Validate collaboration session session = GET\_OR\_CREATE\_COLLABORATION\_SESSION(edit\_operation.template\_id)

// Step 2: Check if user is authorized to edit  
edit\_permissions = CHECK\_TEMPLATE\_EDIT\_PERMISSIONS(edit\_operation.template\_id, user\_id)  
IF NOT edit\_permissions.can\_edit THEN  
 RETURN ERROR("User not authorized to edit this template")  
END IF  
  
// Step 3: Add user to collaboration session if not already present  
IF user\_id NOT IN session.participants THEN  
 JOIN\_COLLABORATION\_SESSION(session.id, user\_id)  
 BROADCAST\_USER\_JOINED\_EVENT(session.id, user\_id)  
END IF  
  
// Step 4: Process the edit operation  
SWITCH edit\_operation.type  
 CASE "text\_edit":  
 result = PROCESS\_TEXT\_EDIT\_OPERATION(edit\_operation, session, user\_id)  
 CASE "cursor\_move":  
 result = PROCESS\_CURSOR\_MOVEMENT(edit\_operation, session, user\_id)  
 CASE "selection\_change":  
 result = PROCESS\_SELECTION\_CHANGE(edit\_operation, session, user\_id)  
 DEFAULT:  
 RETURN ERROR("Unsupported edit operation type")  
END SWITCH  
  
// Step 5: Update session activity  
UPDATE\_SESSION\_ACTIVITY(session.id, user\_id, CURRENT\_TIMESTAMP())  
  
// Step 6: Check for auto-save conditions  
IF SHOULD\_AUTO\_SAVE(session, edit\_operation) THEN  
 auto\_save\_result = PERFORM\_AUTO\_SAVE(session, user\_id)  
 result.auto\_save\_result = auto\_save\_result  
END IF  
  
RETURN result

END

FUNCTION PROCESS\_TEXT\_EDIT\_OPERATION(edit\_operation, session, user\_id) BEGIN // Step 1: Apply operational transformation current\_document\_state = GET\_CURRENT\_DOCUMENT\_STATE(session.template\_id)

transformed\_operation = APPLY\_OPERATIONAL\_TRANSFORMATION(  
 edit\_operation,  
 current\_document\_state,  
 session.pending\_operations  
)  
  
// Step 2: Validate the transformed operation  
validation\_result = VALIDATE\_EDIT\_OPERATION(transformed\_operation, current\_document\_state)  
IF NOT validation\_result.is\_valid THEN  
 RETURN OperationResult(  
 success = FALSE,  
 error = validation\_result.error,  
 operation\_id = edit\_operation.id  
 )  
END IF  
  
// Step 3: Apply operation to document  
new\_document\_state = APPLY\_OPERATION\_TO\_DOCUMENT(  
 current\_document\_state,  
 transformed\_operation  
)  
  
// Step 4: Update session document state  
session.document\_state = new\_document\_state  
session.last\_modified = CURRENT\_TIMESTAMP()  
session.last\_modified\_by = user\_id  
  
// Step 5: Add operation to pending operations queue  
session.pending\_operations.APPEND(transformed\_operation)  
  
// Step 6: Broadcast operation to other participants  
BROADCAST\_OPERATION\_TO\_PARTICIPANTS(  
 session,  
 transformed\_operation,  
 exclude\_user = user\_id  
)  
  
// Step 7: Track editing metrics  
TRACK\_EDIT\_METRICS(  
 template\_id = session.template\_id,  
 user\_id = user\_id,  
 operation\_type = transformed\_operation.type,  
 characters\_changed = transformed\_operation.content.length,  
 timestamp = CURRENT\_TIMESTAMP()  
)  
  
// Step 8: Check for security violations in new content  
IF CONTENT\_REQUIRES\_SECURITY\_SCAN(transformed\_operation.content) THEN  
 QUEUE\_SECURITY\_SCAN(session.template\_id, new\_document\_state.content, user\_id)  
END IF  
  
RETURN OperationResult(  
 success = TRUE,  
 transformed\_operation = transformed\_operation,  
 new\_document\_version = new\_document\_state.version,  
 operation\_id = edit\_operation.id,  
 participants\_notified = session.participants.keys().length - 1  
)

END

FUNCTION DEPLOY\_TEMPLATE\_WORKFLOW(deployment\_config, user\_id) BEGIN // Step 1: Validate deployment configuration validation\_result = VALIDATE\_DEPLOYMENT\_CONFIG(deployment\_config) IF NOT validation\_result.is\_valid THEN RETURN ERROR(“Deployment configuration invalid”, validation\_result.errors) END IF

// Step 2: Check deployment permissions  
deployment\_permissions = CHECK\_DEPLOYMENT\_PERMISSIONS(  
 deployment\_config.template\_id,  
 user\_id,  
 deployment\_config.target\_providers  
)  
IF NOT deployment\_permissions.authorized THEN  
 RETURN ERROR("Insufficient deployment permissions")  
END IF  
  
// Step 3: Get template and version to deploy  
template = GET\_TEMPLATE(deployment\_config.template\_id)  
version\_to\_deploy = GET\_TEMPLATE\_VERSION(deployment\_config.version\_id)  
  
IF template.status != "active" THEN  
 RETURN ERROR("Only active templates can be deployed")  
END IF  
  
// Step 4: Pre-deployment security and compliance checks  
final\_security\_scan = PERFORM\_COMPREHENSIVE\_SECURITY\_SCAN(  
 version\_to\_deploy.content,  
 deployment\_config.target\_providers  
)  
  
compliance\_check = PERFORM\_DEPLOYMENT\_COMPLIANCE\_CHECK(  
 template,  
 version\_to\_deploy,  
 deployment\_config  
)  
  
IF final\_security\_scan.blocks\_deployment OR compliance\_check.blocks\_deployment THEN  
 blocking\_issues = MERGE\_BLOCKING\_ISSUES(final\_security\_scan.issues, compliance\_check.issues)  
 RETURN ERROR("Deployment blocked by security/compliance issues", blocking\_issues)  
END IF  
  
// Step 5: Create deployment record  
deployment = Deployment(  
 id = GENERATE\_UUID(),  
 template\_id = deployment\_config.template\_id,  
 version\_id = deployment\_config.version\_id,  
 target\_providers = deployment\_config.target\_providers,  
 deployment\_config = deployment\_config.provider\_configs,  
 status = "pending",  
 created\_by = user\_id,  
 created\_at = CURRENT\_TIMESTAMP()  
)  
  
// Step 6: Save deployment record  
saved\_deployment = SAVE\_DEPLOYMENT\_TO\_REPOSITORY(deployment)  
  
deployment\_results = []  
  
// Step 7: Deploy to each target provider  
FOR provider\_config IN deployment\_config.target\_providers DO  
 provider\_deployment\_result = DEPLOY\_TO\_PROVIDER(  
 template = template,  
 version = version\_to\_deploy,  
 provider\_config = provider\_config,  
 deployment\_id = saved\_deployment.id  
 )  
   
 deployment\_results.APPEND(provider\_deployment\_result)  
   
 // Update deployment status based on individual provider results  
 IF provider\_deployment\_result.success THEN  
 RECORD\_SUCCESSFUL\_PROVIDER\_DEPLOYMENT(  
 saved\_deployment.id,  
 provider\_config.provider\_name,  
 provider\_deployment\_result  
 )  
 ELSE  
 RECORD\_FAILED\_PROVIDER\_DEPLOYMENT(  
 saved\_deployment.id,  
 provider\_config.provider\_name,  
 provider\_deployment\_result.error  
 )  
 END IF  
END FOR  
  
// Step 8: Update overall deployment status  
overall\_success = ALL(result.success FOR result IN deployment\_results)  
  
IF overall\_success THEN  
 saved\_deployment.status = "completed"  
 saved\_deployment.deployed\_at = CURRENT\_TIMESTAMP()  
ELSE  
 saved\_deployment.status = "partial\_failure"  
 saved\_deployment.error\_summary = SUMMARIZE\_DEPLOYMENT\_ERRORS(deployment\_results)  
END IF  
  
UPDATE\_DEPLOYMENT\_STATUS(saved\_deployment)  
  
// Step 9: Set up monitoring for deployed templates  
FOR successful\_result IN FILTER\_SUCCESSFUL\_DEPLOYMENTS(deployment\_results) DO  
 SETUP\_DEPLOYMENT\_MONITORING(  
 deployment\_id = saved\_deployment.id,  
 provider = successful\_result.provider,  
 endpoint = successful\_result.endpoint,  
 monitoring\_config = deployment\_config.monitoring\_config  
 )  
END FOR  
  
// Step 10: Send deployment notifications  
SEND\_DEPLOYMENT\_NOTIFICATIONS(  
 deployment = saved\_deployment,  
 results = deployment\_results,  
 recipients = GET\_DEPLOYMENT\_NOTIFICATION\_RECIPIENTS(template.id)  
)  
  
RETURN DeploymentResult(  
 deployment = saved\_deployment,  
 provider\_results = deployment\_results,  
 overall\_success = overall\_success,  
 monitoring\_dashboard\_url = GET\_DEPLOYMENT\_MONITORING\_URL(saved\_deployment.id)  
)

END

### A/B Testing Statistical Analysis Workflow

ALGORITHM ComprehensiveABTestAnalysis INPUT: experiment\_id, analysis\_config OUTPUT: statistical\_analysis\_result

BEGIN // Step 1: Validate experiment status and data availability experiment = GET\_EXPERIMENT(experiment\_id)

IF experiment.status NOT IN ["running", "completed"] THEN  
 RETURN ERROR("Experiment must be running or completed for analysis")  
END IF  
  
// Step 2: Collect experiment data  
experiment\_data = COLLECT\_EXPERIMENT\_DATA(  
 experiment\_id = experiment\_id,  
 start\_date = experiment.actual\_start\_date,  
 end\_date = experiment.actual\_end\_date OR CURRENT\_TIMESTAMP()  
)  
  
// Step 3: Validate data quality and completeness  
data\_quality\_check = VALIDATE\_EXPERIMENT\_DATA\_QUALITY(experiment\_data)  
  
IF NOT data\_quality\_check.meets\_minimum\_requirements THEN  
 RETURN ERROR("Insufficient or poor quality data for analysis", data\_quality\_check.issues)  
END IF  
  
// Step 4: Perform statistical analysis for each success metric  
metric\_analyses = {}  
  
FOR metric IN experiment.success\_metrics DO  
 // Extract metric data for all variants  
 control\_data = EXTRACT\_METRIC\_DATA(experiment\_data, "control", metric.name)  
 treatment\_data = {}  
   
 FOR treatment\_version IN experiment.treatment\_versions DO  
 treatment\_data[treatment\_version] = EXTRACT\_METRIC\_DATA(  
 experiment\_data, treatment\_version, metric.name  
 )  
 END FOR  
   
 // Perform appropriate statistical test based on metric type  
 IF metric.type = "conversion\_rate" THEN  
 metric\_analysis = ANALYZE\_CONVERSION\_RATE\_METRIC(  
 control\_data, treatment\_data, metric  
 )  
 ELSE IF metric.type = "continuous" THEN  
 metric\_analysis = ANALYZE\_CONTINUOUS\_METRIC(  
 control\_data, treatment\_data, metric  
 )  
 ELSE IF metric.type = "count" THEN  
 metric\_analysis = ANALYZE\_COUNT\_METRIC(  
 control\_data, treatment\_data, metric  
 )  
 END IF  
   
 metric\_analyses[metric.name] = metric\_analysis  
END FOR  
  
// Step 5: Apply multiple comparison correction  
corrected\_analyses = APPLY\_MULTIPLE\_COMPARISON\_CORRECTION(  
 metric\_analyses,  
 correction\_method = analysis\_config.correction\_method OR "bonferroni"  
)  
  
// Step 6: Calculate overall experiment significance  
overall\_significance = CALCULATE\_OVERALL\_EXPERIMENT\_SIGNIFICANCE(corrected\_analyses)  
  
// Step 7: Perform power analysis  
power\_analysis = PERFORM\_POST\_HOC\_POWER\_ANALYSIS(  
 experiment\_data,  
 corrected\_analyses,  
 experiment.required\_sample\_size  
)  
  
// Step 8: Generate insights and recommendations  
insights = GENERATE\_EXPERIMENT\_INSIGHTS(  
 experiment,  
 corrected\_analyses,  
 power\_analysis  
)  
  
recommendations = GENERATE\_EXPERIMENT\_RECOMMENDATIONS(  
 experiment,  
 corrected\_analyses,  
 insights  
)  
  
// Step 9: Calculate business impact estimates  
business\_impact = ESTIMATE\_BUSINESS\_IMPACT(  
 corrected\_analyses,  
 analysis\_config.business\_metrics  
)  
  
// Step 10: Prepare comprehensive analysis report  
analysis\_result = StatisticalAnalysisResult(  
 experiment\_id = experiment\_id,  
 analysis\_date = CURRENT\_TIMESTAMP(),  
 sample\_sizes = experiment\_data.sample\_sizes,  
 metric\_analyses = corrected\_analyses,  
 overall\_significance = overall\_significance,  
 power\_analysis = power\_analysis,  
 insights = insights,  
 recommendations = recommendations,  
 business\_impact = business\_impact,  
 confidence\_level = analysis\_config.confidence\_level OR 0.95  
)  
  
// Step 11: Save analysis results  
SAVE\_EXPERIMENT\_ANALYSIS(analysis\_result)  
  
// Step 12: Update experiment status if analysis indicates completion  
IF ANALYSIS\_INDICATES\_COMPLETION(analysis\_result) THEN  
 UPDATE\_EXPERIMENT\_STATUS(experiment\_id, "completed")  
 SEND\_EXPERIMENT\_COMPLETION\_NOTIFICATIONS(experiment\_id, analysis\_result)  
END IF  
  
RETURN analysis\_result

END

This completes the comprehensive documentation for Problem Statement 35 - Advanced Prompt Template Management System. The solution provides a complete architecture for managing, versioning, optimizing, and deploying prompt templates across multiple LLM providers with advanced collaboration, A/B testing, security, and analytics capabilities.  
  
Would you like me to continue with Problem Statement 36 (AI Bias Detection and Mitigation Platform) or focus on any other specific problem statements from the remaining list? # Save tags  
 if template.tags:  
 for tag in template.tags:  
 await tx.execute(  
 "INSERT INTO template\_tags (template\_id, tag) VALUES ($1, $2)",  
 template.id, tag  
 )  
   
 # Index for search  
 await self.index\_template\_for\_search(template)  
   
 # Cache frequently accessed metadata  
 await self.cache\_template\_metadata(template)  
   
 return template  
   
 async def get\_template\_with\_content(self, template\_id: str, version: Optional[int] = None) -> Template:  
 # Try cache first  
 cache\_key = f"template:{template\_id}:{version or 'latest'}"  
 cached\_template = await self.cache.get(cache\_key)  
 if cached\_template:  
 return Template.from\_dict(cached\_template)  
   
 # Get template metadata from primary DB  
 template\_record = await self.primary\_db.fetchrow(  
 """  
 SELECT t.\*, array\_agg(tt.tag) as tags  
 FROM templates t  
 LEFT JOIN template\_tags tt ON t.id = tt.template\_id  
 WHERE t.id = $1 AND t.deleted\_at IS NULL  
 GROUP BY t.id  
 """,  
 template\_id  
 )  
   
 if not template\_record:  
 raise TemplateNotFoundError(f"Template {template\_id} not found")  
   
 # Get content from document store  
 content\_query = {'template\_id': template\_id}  
 if version:  
 content\_query['version'] = version  
   
 content\_doc = await self.document\_store.templates.find\_one(  
 content\_query,  
 sort=[('version', -1)] # Get latest version if not specified  
 )  
   
 if not content\_doc:  
 raise TemplateContentNotFoundError(f"Content for template {template\_id} not found")  
   
 # Construct template object  
 template = Template(  
 id=template\_record['id'],  
 name=template\_record['name'],  
 category=template\_record['category'],  
 content=content\_doc['content'],  
 variables=content\_doc.get('variables', []),  
 metadata=content\_doc.get('metadata', {}),  
 tags=template\_record['tags'] or [],  
 created\_by=template\_record['created\_by'],  
 created\_at=template\_record['created\_at'],  
 updated\_at=template\_record['updated\_at'],  
 version=content\_doc['version']  
 )  
   
 # Cache for future requests  
 await self.cache.setex(cache\_key, 3600, template.to\_dict())  
   
 return template  
   
 async def search\_templates(self, query: TemplateSearchQuery) -> SearchResults:  
 # Build Elasticsearch query  
 es\_query = {  
 "query": {  
 "bool": {  
 "must": [],  
 "filter": []  
 }  
 },  
 "sort": [],  
 "aggs": {  
 "categories": {"terms": {"field": "category.keyword"}},  
 "tags": {"terms": {"field": "tags.keyword"}},  
 "creators": {"terms": {"field": "created\_by.keyword"}}  
 }  
 }  
   
 # Add text search  
 if query.text:  
 es\_query["query"]["bool"]["must"].append({  
 "multi\_match": {  
 "query": query.text,  
 "fields": ["name^3", "content^2", "description", "tags"],  
 "fuzziness": "AUTO"  
 }  
 })  
   
 # Add filters  
 if query.categories:  
 es\_query["query"]["bool"]["filter"].append({  
 "terms": {"category.keyword": query.categories}  
 })  
   
 if query.tags:  
 es\_query["query"]["bool"]["filter"].append({  
 "terms": {"tags.keyword": query.tags}  
 })  
   
 if query.date\_range:  
 es\_query["query"]["bool"]["filter"].append({  
 "range": {  
 "created\_at": {  
 "gte": query.date\_range.start,  
 "lte": query.date\_range.end  
 }  
 }  
 })  
   
 # Add sorting  
 if query.sort\_by == "relevance":  
 es\_query["sort"].append({"\_score": {"order": "desc"}})  
 elif query.sort\_by == "created\_at":  
 es\_query["sort"].append({"created\_at": {"order": query.sort\_order}})  
 elif query.sort\_by == "performance":  
 es\_query["sort"].append({"performance\_score": {"order": "desc"}})  
   
 # Execute search  
 search\_result = await self.search\_index.search(  
 index="templates",  
 body=es\_query,  
 from\_=query.offset,  
 size=query.limit  
 )  
   
 # Process results  
 templates = []  
 for hit in search\_result['hits']['hits']:  
 template\_data = hit['\_source']  
 template\_data['relevance\_score'] = hit['\_score']  
 templates.append(Template.from\_search\_hit(template\_data))  
   
 return SearchResults(  
 templates=templates,  
 total\_count=search\_result['hits']['total']['value'],  
 facets=self.process\_aggregations(search\_result['aggregations']),  
 took\_ms=search\_result['took']  
 )  
  
class TemplateSearchEngine:  
 def \_\_init\_\_(self):  
 self.elasticsearch = ElasticsearchConnection()  
 self.query\_optimizer = QueryOptimizer()  
   
 async def index\_template(self, template: Template):  
 # Prepare document for indexing  
 doc = {  
 'id': template.id,  
 'name': template.name,  
 'category': template.category,  
 'content': template.content,  
 'description': template.metadata.get('description', ''),  
 'tags': template.tags,  
 'created\_by': template.created\_by,  
 'created\_at': template.created\_at,  
 'updated\_at': template.updated\_at,  
 'variables': [var['name'] for var in template.variables],  
 'complexity\_score': self.calculate\_complexity\_score(template),  
 'performance\_score': await self.get\_performance\_score(template.id),  
 'usage\_count': await self.get\_usage\_count(template.id),  
 'folder\_path': await self.get\_folder\_path(template.folder\_id)  
 }  
   
 # Index document  
 await self.elasticsearch.index(  
 index="templates",  
 id=template.id,  
 document=doc  
 )  
   
 def calculate\_complexity\_score(self, template: Template) -> float:  
 """Calculate template complexity based on various factors"""  
 complexity\_factors = {  
 'content\_length': len(template.content),  
 'variable\_count': len(template.variables),  
 'conditional\_logic': template.content.count('{{#if') + template.content.count('{{#each'),  
 'nested\_variables': template.content.count('{{') - template.content.count('}}')/2  
 }  
   
 # Normalize and weight factors  
 normalized\_score = (  
 min(complexity\_factors['content\_length'] / 10000, 1.0) \* 0.3 +  
 min(complexity\_factors['variable\_count'] / 20, 1.0) \* 0.3 +  
 min(complexity\_factors['conditional\_logic'] / 10, 1.0) \* 0.2 +  
 min(complexity\_factors['nested\_variables'] / 30, 1.0) \* 0.2  
 )  
   
 return normalized\_score

#### 2. Advanced A/B Testing Statistical Engine

##### Statistical Analysis Framework

class StatisticalEngine:  
 def \_\_init\_\_(self):  
 self.power\_analyzer = PowerAnalyzer()  
 self.effect\_estimator = EffectEstimator()  
 self.confidence\_calculator = ConfidenceCalculator()  
   
 def calculate\_sample\_size(  
 self,   
 baseline\_rate: float,   
 minimum\_detectable\_effect: float,  
 statistical\_power: float = 0.8,  
 significance\_level: float = 0.05  
 ) -> int:  
 """Calculate required sample size using power analysis"""  
 from scipy.stats import norm  
 import math  
   
 # Convert to effect size (Cohen's h for proportions)  
 p1 = baseline\_rate  
 p2 = baseline\_rate \* (1 + minimum\_detectable\_effect)  
   
 # Cohen's h formula  
 effect\_size = 2 \* (math.asin(math.sqrt(p1)) - math.asin(math.sqrt(p2)))  
   
 # Power analysis calculation  
 alpha = significance\_level  
 beta = 1 - statistical\_power  
   
 z\_alpha = norm.ppf(1 - alpha/2) # Two-tailed test  
 z\_beta = norm.ppf(statistical\_power)  
   
 # Sample size per group  
 n\_per\_group = ((z\_alpha + z\_beta) / effect\_size) \*\* 2  
   
 # Total sample size (both groups)  
 total\_sample\_size = int(math.ceil(n\_per\_group \* 2))  
   
 return max(total\_sample\_size, 100) # Minimum sample size  
   
 async def analyze\_experiment(  
 self,  
 control\_data: ExperimentData,  
 treatment\_data: List[ExperimentData],  
 success\_metrics: List[SuccessMetric]  
 ) -> StatisticalAnalysis:  
   
 analysis\_results = {}  
   
 for metric in success\_metrics:  
 metric\_results = {}  
   
 for i, treatment in enumerate(treatment\_data):  
 # Perform statistical test based on metric type  
 if metric.type == 'conversion\_rate':  
 test\_result = self.analyze\_conversion\_rate(  
 control\_data.get\_metric\_data(metric.name),  
 treatment.get\_metric\_data(metric.name)  
 )  
 elif metric.type == 'continuous':  
 test\_result = self.analyze\_continuous\_metric(  
 control\_data.get\_metric\_data(metric.name),  
 treatment.get\_metric\_data(metric.name)  
 )  
 elif metric.type == 'count':  
 test\_result = self.analyze\_count\_metric(  
 control\_data.get\_metric\_data(metric.name),  
 treatment.get\_metric\_data(metric.name)  
 )  
   
 metric\_results[f'treatment\_{i+1}'] = test\_result  
   
 analysis\_results[metric.name] = metric\_results  
   
 # Perform multiple comparison correction  
 corrected\_results = self.apply\_multiple\_comparison\_correction(  
 analysis\_results, method='bonferroni'  
 )  
   
 # Calculate overall experiment significance  
 overall\_significance = self.calculate\_overall\_significance(corrected\_results)  
   
 return StatisticalAnalysis(  
 metric\_results=corrected\_results,  
 overall\_significant=overall\_significance.is\_significant,  
 confidence\_level=overall\_significance.confidence\_level,  
 effect\_sizes=self.calculate\_effect\_sizes(analysis\_results),  
 recommendations=self.generate\_statistical\_recommendations(corrected\_results)  
 )  
   
 def analyze\_conversion\_rate(self, control\_data: MetricData, treatment\_data: MetricData) -> TestResult:  
 """Analyze conversion rate using Chi-square test or Fisher's exact test"""  
 from scipy.stats import chi2\_contingency, fisher\_exact  
 import numpy as np  
   
 # Prepare contingency table  
 control\_successes = control\_data.successes  
 control\_total = control\_data.total\_samples  
 control\_failures = control\_total - control\_successes  
   
 treatment\_successes = treatment\_data.successes  
 treatment\_total = treatment\_data.total\_samples  
 treatment\_failures = treatment\_total - treatment\_successes  
   
 contingency\_table = np.array([  
 [control\_successes, control\_failures],  
 [treatment\_successes, treatment\_failures]  
 ])  
   
 # Choose appropriate test  
 if min(contingency\_table.flatten()) < 5:  
 # Use Fisher's exact test for small samples  
 odds\_ratio, p\_value = fisher\_exact(contingency\_table)  
 test\_statistic = odds\_ratio  
 test\_name = "Fisher's Exact Test"  
 else:  
 # Use Chi-square test for larger samples  
 test\_statistic, p\_value, dof, expected = chi2\_contingency(contingency\_table)  
 test\_name = "Chi-square Test"  
   
 # Calculate confidence interval for difference in proportions  
 control\_rate = control\_successes / control\_total  
 treatment\_rate = treatment\_successes / treatment\_total  
   
 confidence\_interval = self.calculate\_proportion\_difference\_ci(  
 control\_rate, treatment\_rate, control\_total, treatment\_total  
 )  
   
 # Calculate effect size (relative improvement)  
 relative\_improvement = (treatment\_rate - control\_rate) / control\_rate if control\_rate > 0 else 0  
   
 return TestResult(  
 test\_name=test\_name,  
 test\_statistic=test\_statistic,  
 p\_value=p\_value,  
 is\_significant=p\_value < 0.05,  
 confidence\_interval=confidence\_interval,  
 effect\_size=relative\_improvement,  
 control\_metric=control\_rate,  
 treatment\_metric=treatment\_rate,  
 sample\_sizes={'control': control\_total, 'treatment': treatment\_total}  
 )  
   
 def analyze\_continuous\_metric(self, control\_data: MetricData, treatment\_data: MetricData) -> TestResult:  
 """Analyze continuous metrics using t-test or Mann-Whitney U test"""  
 from scipy.stats import ttest\_ind, mannwhitneyu, levene  
 import numpy as np  
   
 control\_values = np.array(control\_data.values)  
 treatment\_values = np.array(treatment\_data.values)  
   
 # Test for equal variances  
 levene\_stat, levene\_p = levene(control\_values, treatment\_values)  
 equal\_variances = levene\_p > 0.05  
   
 # Check for normality (simplified)  
 control\_normal = len(control\_values) >= 30 # Central limit theorem approximation  
 treatment\_normal = len(treatment\_values) >= 30  
   
 if control\_normal and treatment\_normal:  
 # Use t-test  
 t\_stat, p\_value = ttest\_ind(  
 control\_values,   
 treatment\_values,   
 equal\_var=equal\_variances  
 )  
 test\_name = "Student's t-test" if equal\_variances else "Welch's t-test"  
 test\_statistic = t\_stat  
 else:  
 # Use non-parametric test  
 u\_stat, p\_value = mannwhitneyu(  
 control\_values,   
 treatment\_values,   
 alternative='two-sided'  
 )  
 test\_name = "Mann-Whitney U test"  
 test\_statistic = u\_stat  
   
 # Calculate effect size (Cohen's d)  
 pooled\_std = np.sqrt((np.var(control\_values) + np.var(treatment\_values)) / 2)  
 cohens\_d = (np.mean(treatment\_values) - np.mean(control\_values)) / pooled\_std  
   
 # Calculate confidence interval for mean difference  
 confidence\_interval = self.calculate\_mean\_difference\_ci(  
 control\_values, treatment\_values  
 )  
   
 return TestResult(  
 test\_name=test\_name,  
 test\_statistic=test\_statistic,  
 p\_value=p\_value,  
 is\_significant=p\_value < 0.05,  
 confidence\_interval=confidence\_interval,  
 effect\_size=cohens\_d,  
 control\_metric=np.mean(control\_values),  
 treatment\_metric=np.mean(treatment\_values),  
 sample\_sizes={'control': len(control\_values), 'treatment': len(treatment\_values)}  
 )  
   
 def apply\_multiple\_comparison\_correction(self, analysis\_results: dict, method: str = 'bonferroni') -> dict:  
 """Apply correction for multiple comparisons"""  
 from statsmodels.stats.multitest import multipletests  
   
 # Collect all p-values  
 p\_values = []  
 result\_mapping = []  
   
 for metric\_name, metric\_results in analysis\_results.items():  
 for treatment\_name, test\_result in metric\_results.items():  
 p\_values.append(test\_result.p\_value)  
 result\_mapping.append((metric\_name, treatment\_name))  
   
 # Apply correction  
 if method == 'bonferroni':  
 corrected\_p\_values = [p \* len(p\_values) for p in p\_values]  
 corrected\_p\_values = [min(p, 1.0) for p in corrected\_p\_values] # Cap at 1.0  
 elif method == 'benjamini\_hochberg':  
 \_, corrected\_p\_values, \_, \_ = multipletests(p\_values, method='fdr\_bh')  
   
 # Update results with corrected p-values  
 corrected\_results = {}  
 for i, (metric\_name, treatment\_name) in enumerate(result\_mapping):  
 if metric\_name not in corrected\_results:  
 corrected\_results[metric\_name] = {}  
   
 original\_result = analysis\_results[metric\_name][treatment\_name]  
 corrected\_result = original\_result.copy()  
 corrected\_result.corrected\_p\_value = corrected\_p\_values[i]  
 corrected\_result.is\_significant\_corrected = corrected\_p\_values[i] < 0.05  
   
 corrected\_results[metric\_name][treatment\_name] = corrected\_result  
   
 return corrected\_results

#### 3. Collaborative Editing Engine

##### Real-Time Collaboration System

class CollaborationEngine:  
 def \_\_init\_\_(self):  
 self.websocket\_manager = WebSocketManager()  
 self.operational\_transform = OperationalTransform()  
 self.conflict\_resolver = ConflictResolver()  
 self.activity\_tracker = ActivityTracker()  
   
 async def join\_collaboration\_session(self, template\_id: str, user\_id: str) -> CollaborationSession:  
 # Get or create collaboration session  
 session = await self.get\_or\_create\_session(template\_id)  
   
 # Add user to session  
 participant = CollaborationParticipant(  
 user\_id=user\_id,  
 joined\_at=datetime.utcnow(),  
 cursor\_position=0,  
 is\_active=True  
 )  
   
 session.participants[user\_id] = participant  
   
 # Get current document state  
 current\_state = await self.get\_document\_state(template\_id)  
   
 # Send current state to new participant  
 await self.websocket\_manager.send\_to\_user(user\_id, {  
 'type': 'document\_state',  
 'content': current\_state.content,  
 'version': current\_state.version,  
 'participants': [p.to\_dict() for p in session.participants.values()]  
 })  
   
 # Notify other participants  
 await self.broadcast\_to\_session(session.id, {  
 'type': 'user\_joined',  
 'user\_id': user\_id,  
 'participant\_count': len(session.participants)  
 }, exclude\_user=user\_id)  
   
 return session  
   
 async def handle\_text\_operation(self, operation: TextOperation) -> OperationResult:  
 session = await self.get\_session(operation.template\_id)  
   
 # Apply operational transformation  
 transformed\_operation = await self.operational\_transform.transform\_operation(  
 operation, session.document\_state  
 )  
   
 # Validate operation  
 validation\_result = self.validate\_operation(transformed\_operation, session)  
 if not validation\_result.is\_valid:  
 return OperationResult(  
 success=False,  
 error=validation\_result.error,  
 operation\_id=operation.id  
 )  
   
 # Apply operation to document  
 new\_document\_state = self.apply\_operation\_to\_document(  
 session.document\_state, transformed\_operation  
 )  
   
 # Update session state  
 session.document\_state = new\_document\_state  
 session.last\_modified = datetime.utcnow()  
 session.last\_modified\_by = operation.user\_id  
   
 # Broadcast operation to other participants  
 await self.broadcast\_operation\_to\_participants(  
 session, transformed\_operation, exclude\_user=operation.user\_id  
 )  
   
 # Track activity  
 await self.activity\_tracker.track\_edit\_activity(  
 template\_id=operation.template\_id,  
 user\_id=operation.user\_id,  
 operation\_type=operation.type,  
 operation\_size=len(operation.content)  
 )  
   
 return OperationResult(  
 success=True,  
 transformed\_operation=transformed\_operation,  
 new\_version=new\_document\_state.version,  
 operation\_id=operation.id  
 )  
   
 async def handle\_cursor\_movement(self, cursor\_update: CursorUpdate):  
 session = await self.get\_session(cursor\_update.template\_id)  
   
 # Update participant cursor position  
 if cursor\_update.user\_id in session.participants:  
 participant = session.participants[cursor\_update.user\_id]  
 participant.cursor\_position = cursor\_update.position  
 participant.selection\_range = cursor\_update.selection\_range  
 participant.last\_activity = datetime.utcnow()  
   
 # Broadcast cursor update to other participants  
 await self.broadcast\_to\_session(session.id, {  
 'type': 'cursor\_update',  
 'user\_id': cursor\_update.user\_id,  
 'position': cursor\_update.position,  
 'selection\_range': cursor\_update.selection\_range  
 }, exclude\_user=cursor\_update.user\_id)  
  
class OperationalTransform:  
 """Implements operational transformation for conflict-free collaborative editing"""  
   
 def \_\_init\_\_(self):  
 self.operation\_queue = OperationQueue()  
   
 async def transform\_operation(self, operation: TextOperation, document\_state: DocumentState) -> TextOperation:  
 # Get all operations that happened after this operation's base version  
 concurrent\_operations = await self.get\_concurrent\_operations(  
 operation.base\_version,   
 document\_state.version,  
 operation.template\_id  
 )  
   
 transformed\_operation = operation  
   
 # Transform against each concurrent operation  
 for concurrent\_op in concurrent\_operations:  
 transformed\_operation = self.transform\_against\_operation(  
 transformed\_operation, concurrent\_op  
 )  
   
 return transformed\_operation  
   
 def transform\_against\_operation(self, op1: TextOperation, op2: TextOperation) -> TextOperation:  
 """Transform op1 against op2 using operational transformation rules"""  
   
 if op1.type == 'insert' and op2.type == 'insert':  
 return self.transform\_insert\_insert(op1, op2)  
 elif op1.type == 'insert' and op2.type == 'delete':  
 return self.transform\_insert\_delete(op1, op2)  
 elif op1.type == 'delete' and op2.type == 'insert':  
 return self.transform\_delete\_insert(op1, op2)  
 elif op1.type == 'delete' and op2.type == 'delete':  
 return self.transform\_delete\_delete(op1, op2)  
 else:  
 return op1  
   
 def transform\_insert\_insert(self, op1: TextOperation, op2: TextOperation) -> TextOperation:  
 """Transform insert operation against another insert operation"""  
 if op1.position <= op2.position:  
 # op1 position is before or at op2 position - no change needed  
 return op1  
 else:  
 # op1 position is after op2 position - adjust by length of op2 insertion  
 transformed\_op = op1.copy()  
 transformed\_op.position += len(op2.content)  
 return transformed\_op  
   
 def transform\_insert\_delete(self, op1: TextOperation, op2: TextOperation) -> TextOperation:  
 """Transform insert operation against delete operation"""  
 if op1.position <= op2.position:  
 # Insert is before delete - no change needed  
 return op1  
 elif op1.position >= op2.position + op2.length:  
 # Insert is after delete - adjust position  
 transformed\_op = op1.copy()  
 transformed\_op.position -= op2.length  
 return transformed\_op  
 else:  
 # Insert is within delete range - adjust to delete position  
 transformed\_op = op1.copy()  
 transformed\_op.position = op2.position  
 return transformed\_op  
   
 def transform\_delete\_insert(self, op1: TextOperation, op2: TextOperation) -> TextOperation:  
 """Transform delete operation against insert operation"""  
 if op2.position <= op1.position:  
 # Insert is before delete - adjust delete position  
 transformed\_op = op1.copy()  
 transformed\_op.position += len(op2.content)  
 return transformed\_op  
 elif op2.position >= op1.position + op1.length:  
 # Insert is after delete - no change needed  
 return op1  
 else:  
 # Insert is within delete range - split delete operation  
 # This is a complex case that may require splitting into multiple operations  
 # For simplicity, we'll adjust the length  
 transformed\_op = op1.copy()  
 transformed\_op.length += len(op2.content)  
 return transformed\_op  
   
 def transform\_delete\_delete(self, op1: TextOperation, op2: TextOperation) -> TextOperation:  
 """Transform delete operation against another delete operation"""  
 if op1.position + op1.length <= op2.position:  
 # op1 is completely before op2 - no change needed  
 return op1  
 elif op1.position >= op2.position + op2.length:  
 # op1 is completely after op2 - adjust position  
 transformed\_op = op1.copy()  
 transformed\_op.position -= op2.length  
 return transformed\_op  
 else:  
 # Overlapping deletes - complex case  
 # Calculate the intersection and adjust accordingly  
 overlap\_start = max(op1.position, op2.position)  
 overlap\_end = min(op1.position + op1.length, op2.position + op2.length)  
 overlap\_length = max(0, overlap\_end - overlap\_start)  
   
 transformed\_op = op1.copy()  
   
 if op2.position <= op1.position:  
 # op2 starts before or at op1  
 transformed\_op.position = op2.position  
 transformed\_op.length = max(0, op1.length - overlap\_length)  
 else:  
 # op2 starts within op1  
 transformed\_op.length = max(0, op1.length - overlap\_length)  
   
 return transformed\_op

#### 4. Database Schema Implementation

##### PostgreSQL Schema

```sql – Core templates table CREATE TABLE templates ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), name VARCHAR(255) NOT NULL, description TEXT, category VARCHAR(100) NOT NULL, folder\_id UUID REFERENCES template\_folders(id), created\_by UUID NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_by UUID, status VARCHAR(50) DEFAULT ‘draft’, – ‘draft’, ‘active’, ‘archived’, ‘deprecated’ deleted\_at TIMESTAMP, performance\_score FLOAT DEFAULT 0.0, usage\_count INTEGER DEFAULT 0,

CONSTRAINT valid\_status CHECK (status IN ('draft', 'active', 'archived', 'deprecated'))

);

– Template folders for organization CREATE TABLE template\_folders ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), name VARCHAR(255) NOT NULL, parent\_folder\_id UUID REFERENCES template\_folders(id), created\_by UUID NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, folder\_path TEXT, – Materialized path for efficient queries

UNIQUE(parent\_folder\_id, name)

);

– Template tags CREATE TABLE template\_tags ( template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, tag VARCHAR(100) NOT NULL,

PRIMARY KEY(template\_id, tag)

);

– Template versions CREATE TABLE template\_versions ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, version\_number INTEGER NOT NULL, content TEXT NOT NULL, variables JSONB DEFAULT ‘[]’, metadata JSONB DEFAULT ‘{}’, commit\_message TEXT, created\_by UUID NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, parent\_version\_id UUID REFERENCES template\_versions(id), branch\_name VARCHAR(100) DEFAULT ‘main’,

UNIQUE(template\_id, version\_number)

);

– A/B Testing experiments CREATE TABLE ab\_experiments ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), name VARCHAR(255) NOT NULL, description TEXT, template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, control\_version\_id UUID REFERENCES template\_versions(id), treatment\_versions JSONB NOT NULL, – Array of version IDs traffic\_split JSONB NOT NULL, – Traffic allocation percentages success\_metrics JSONB NOT NULL, start\_date TIMESTAMP, end\_date TIMESTAMP, actual\_start\_date TIMESTAMP, actual\_end\_date TIMESTAMP, status VARCHAR(50) DEFAULT ‘created’, required\_sample\_size INTEGER, current\_sample\_size INTEGER DEFAULT 0, statistical\_significance FLOAT, created\_by UUID NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT valid\_experiment\_status CHECK (  
 status IN ('created', 'running', 'paused', 'completed', 'cancelled')  
)

);

– Experiment results CREATE TABLE ab\_experiment\_results ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), experiment\_id UUID REFERENCES ab\_experiments(id) ON DELETE CASCADE, version\_id UUID REFERENCES template\_versions(id), metric\_name VARCHAR(100) NOT NULL, metric\_value FLOAT NOT NULL, sample\_size INTEGER NOT NULL, recorded\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, session\_id VARCHAR(255), – For grouping related metrics

INDEX(experiment\_id, version\_id, metric\_name),  
INDEX(recorded\_at)

);

– Template executions for performance tracking CREATE TABLE template\_executions ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, version\_id UUID REFERENCES template\_versions(id), provider VARCHAR(50) NOT NULL, model VARCHAR(100) NOT NULL, execution\_time\_ms INTEGER, tokens\_used INTEGER, cost\_usd DECIMAL(10, 6), success BOOLEAN NOT NULL, error\_message TEXT, quality\_score FLOAT, executed\_by UUID, executed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

INDEX(template\_id, executed\_at),  
INDEX(provider, model, executed\_at)

);

– Collaboration sessions CREATE TABLE collaboration\_sessions ( id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), template\_id UUID REFERENCES templates(id) ON DELETE CASCADE, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, last\_activity TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, participant\_count INTEGER DEFAULT 0, document\_version INTEGER NOT NULL, session\_state JSONB DEFAULT ‘{}’ );

– Collaboration participants CREATE TABLE collaboration\_participants ( session\_id UUID REFERENCES collaboration\_sessions(id) ON DELETE CASCADE, user\_id UUID NOT NULL, joined\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, left\_at TIMESTAMP, cursor\_position INTEGER DEFAULT 0, is\_active BOOLEAN DEFAULT TRUE,

PRIMARY# 140509\_35.md - Advanced Prompt Template Management System

## README

**Summary:** Create a comprehensive system for managing, versioning, and optimizing prompt templates across different proprietary LLM providers.

**Problem Statement:** Managing prompts across different providers and use cases requires systematic organization and optimization. Your task is to build a prompt management system that stores, versions, and optimizes prompt templates for various proprietary models. The system should enable A/B testing, performance tracking, and collaborative prompt development while maintaining security and compliance.

**Steps:** - Design prompt template storage with versioning and metadata management - Implement A/B testing framework for prompt optimization - Create collaborative editing and review workflows for prompt development - Build performance tracking and quality metrics across different providers - Develop prompt security scanning and compliance checking - Include automated prompt generation and improvement suggestions

**Suggested Data Requirements:** - Prompt template libraries with performance metadata - A/B testing results and optimization histories - Security and compliance requirements for prompt content - User collaboration and editing logs

**Themes:** Proprietary models, GenAI & its techniques

## PRD (Product Requirements Document)

### Product Vision

Create a comprehensive prompt management platform that enables organizations to systematically develop, optimize, and deploy prompt templates across multiple LLM providers while ensuring security, compliance, and optimal performance through data-driven insights and collaborative workflows.

### Target Users

* **Primary:** AI Engineers, Prompt Engineers, ML Operations Teams
* **Secondary:** Product Managers, Content Teams, AI Researchers
* **Tertiary:** Compliance Officers, Security Teams, Business Analysts

### Core Value Propositions

1. **Centralized Management:** Single source of truth for all prompt templates across providers
2. **Performance Optimization:** Data-driven prompt improvement through A/B testing and analytics
3. **Collaboration Excellence:** Streamlined workflows for team-based prompt development
4. **Security & Compliance:** Automated scanning and policy enforcement
5. **Cross-Provider Compatibility:** Seamless deployment across different LLM providers

### Key Features

1. **Prompt Template Library:** Hierarchical organization with tagging and search capabilities
2. **Version Control System:** Git-like versioning with branching and merging
3. **A/B Testing Platform:** Automated experimentation with statistical significance testing
4. **Performance Analytics:** Comprehensive metrics tracking across providers and use cases
5. **Collaborative Editing:** Real-time collaboration with review and approval workflows
6. **Security Scanner:** Automated detection of sensitive content and compliance violations
7. **Multi-Provider Deployment:** Unified interface for deploying to various LLM APIs

### Success Metrics

* Prompt performance improvement: >25% average improvement in task success rates
* Development velocity: 50% reduction in prompt development time
* Collaboration efficiency: >90% of prompts reviewed within 24 hours
* Security compliance: 100% automated detection of policy violations
* Provider coverage: Support for 10+ major LLM providers within 6 months

## FRD (Functional Requirements Document)

### Core Functional Requirements

#### F1: Prompt Template Storage and Organization

* **F1.1:** Hierarchical folder structure with unlimited nesting depth
* **F1.2:** Rich metadata support (tags, categories, use cases, performance metrics)
* **F1.3:** Template inheritance and composition capabilities
* **F1.4:** Cross-reference linking between related prompts
* **F1.5:** Bulk import/export functionality for template libraries

#### F2: Version Control and Change Management

* **F2.1:** Git-style versioning with commit history and messages
* **F2.2:** Branching and merging capabilities for parallel development
* **F2.3:** Diff visualization showing changes between versions
* **F2.4:** Rollback functionality to previous versions
* **F2.5:** Automated backup and recovery systems

#### F3: A/B Testing and Experimentation Framework

* **F3.1:** Automated test setup with control and variant groups
* **F3.2:** Statistical significance calculation and early stopping
* **F3.3:** Multi-variate testing support for complex experiments
* **F3.4:** Custom success metrics definition and tracking
* **F3.5:** Integration with external analytics and monitoring systems

#### F4: Performance Tracking and Analytics

* **F4.1:** Real-time performance metrics collection across providers
* **F4.2:** Cost tracking and optimization recommendations
* **F4.3:** Latency and throughput monitoring
* **F4.4:** Quality metrics (accuracy, relevance, coherence) assessment
* **F4.5:** Comparative analysis between providers and models

#### F5: Collaborative Development Workflows

* **F5.1:** Real-time collaborative editing with conflict resolution
* **F5.2:** Review and approval processes with customizable workflows
* **F5.3:** Comment and annotation systems for feedback
* **F5.4:** Role-based access control and permissions management
* **F5.5:** Integration with project management and communication tools

#### F6: Security and Compliance Management

* **F6.1:** Automated scanning for PII and sensitive data
* **F6.2:** Policy enforcement and compliance checking
* **F6.3:** Audit trail for all prompt modifications and deployments
* **F6.4:** Data residency and regional compliance controls
* **F6.5:** Integration with enterprise security systems

#### F7: Multi-Provider Deployment and Management

* **F7.1:** Unified API interface for multiple LLM providers
* **F7.2:** Provider-specific optimization and formatting
* **F7.3:** Failover and load balancing across providers
* **F7.4:** Cost optimization through intelligent provider routing
* **F7.5:** Performance monitoring across all deployed instances

## NFRD (Non-Functional Requirements Document)

### Performance Requirements

* **NFR-P1:** Prompt retrieval latency: <50ms for cached templates
* **NFR-P2:** A/B test result calculation: <5 seconds for standard experiments
* **NFR-P3:** Collaborative editing response time: <200ms for real-time updates
* **NFR-P4:** Bulk operations: Process 10,000+ templates in <10 minutes
* **NFR-P5:** Analytics dashboard loading: <3 seconds for standard reports

### Scalability Requirements

* **NFR-S1:** Support for 1M+ prompt templates with full-text search
* **NFR-S2:** Concurrent users: 1000+ simultaneous collaborative editors
* **NFR-S3:** A/B test scale: 100+ concurrent experiments
* **NFR-S4:** Provider API calls: Handle 1M+ requests per hour
* **NFR-S5:** Storage scaling: Petabyte-scale template and analytics data

### Reliability Requirements

* **NFR-R1:** System availability: 99.9% uptime for core services
* **NFR-R2:** Data durability: 99.999% for prompt templates and version history
* **NFR-R3:** Backup and recovery: RPO 15 minutes, RTO 30 minutes
* **NFR-R4:** Cross-region replication for disaster recovery
* **NFR-R5:** Graceful degradation during provider API outages

### Security Requirements

* **NFR-SE1:** End-to-end encryption for sensitive prompt content
* **NFR-SE2:** Multi-factor authentication and SSO integration
* **NFR-SE3:** Role-based access control with fine-grained permissions
* **NFR-SE4:** Audit logging with immutable records
* **NFR-SE5:** Compliance with SOC 2, GDPR, and industry-specific regulations

### Usability Requirements

* **NFR-U1:** Intuitive interface: <5 minutes learning curve for basic operations
* **NFR-U2:** Mobile responsiveness for review and approval workflows
* **NFR-U3:** Accessibility compliance (WCAG 2.1 AA)
* **NFR-U4:** Multi-language support for global teams
* **NFR-U5:** Comprehensive API documentation and SDKs

### Integration Requirements

* **NFR-I1:** REST API with OpenAPI 3.0 specification
* **NFR-I2:** Webhook support for real-time notifications
* **NFR-I3:** CI/CD pipeline integration for automated deployments
* **NFR-I4:** Enterprise tool integration (Slack, Teams, Jira, GitHub)
* **NFR-I5:** Third-party analytics and monitoring platform integration

## AD (Architecture Diagram)

graph TB  
 subgraph "Client Layer"  
 WEB\_UI[Web Application]  
 MOBILE[Mobile App]  
 IDE\_PLUGINS[IDE Plugins]  
 API\_CLIENTS[API Clients]  
 end  
   
 subgraph "API Gateway & Security"  
 LB[Load Balancer]  
 API\_GW[API Gateway]  
 AUTH\_SVC[Authentication Service]  
 RATE\_LIMIT[Rate Limiter]  
 end  
   
 subgraph "Core Services"  
 TEMPLATE\_MGR[Template Manager]  
 VERSION\_CTRL[Version Control Service]  
 COLLAB\_ENGINE[Collaboration Engine]  
 AB\_TEST\_MGR[A/B Testing Manager]  
 ANALYTICS\_SVC[Analytics Service]  
 DEPLOY\_MGR[Deployment Manager]  
 end  
   
 subgraph "Specialized Services"  
 SECURITY\_SCANNER[Security Scanner]  
 COMPLIANCE\_CHECKER[Compliance Checker]  
 PERFORMANCE\_MONITOR[Performance Monitor]  
 OPTIMIZATION\_ENGINE[Optimization Engine]  
 WORKFLOW\_ENGINE[Workflow Engine]  
 end  
   
 subgraph "LLM Provider Integration"  
 PROVIDER\_GATEWAY[Provider Gateway]  
 OPENAI\_ADAPTER[OpenAI Adapter]  
 ANTHROPIC\_ADAPTER[Anthropic Adapter]  
 GOOGLE\_ADAPTER[Google Adapter]  
 AZURE\_ADAPTER[Azure Adapter]  
 AWS\_ADAPTER[AWS Adapter]  
 end  
   
 subgraph "Data Layer"  
 POSTGRES[PostgreSQL - Core Data]  
 MONGODB[MongoDB - Templates]  
 REDIS[Redis - Cache & Sessions]  
 ELASTICSEARCH[Elasticsearch - Search]  
 TIMESERIES[InfluxDB - Metrics]  
 BLOB\_STORAGE[Object Storage - Assets]  
 end  
   
 subgraph "Message Queue & Processing"  
 MESSAGE\_QUEUE[Message Queue]  
 TASK\_PROCESSOR[Task Processor]  
 NOTIFICATION\_SVC[Notification Service]  
 WEBHOOK\_DISPATCHER[Webhook Dispatcher]  
 end  
   
 subgraph "External Integrations"  
 SLACK\_API[Slack Integration]  
 TEAMS\_API[Teams Integration]  
 JIRA\_API[Jira Integration]  
 GITHUB\_API[GitHub Integration]  
 ANALYTICS\_TOOLS[External Analytics]  
 end  
   
 WEB\_UI --> LB  
 MOBILE --> LB  
 IDE\_PLUGINS --> LB  
 API\_CLIENTS --> LB  
   
 LB --> API\_GW  
 API\_GW --> AUTH\_SVC  
 API\_GW --> RATE\_LIMIT  
   
 API\_GW --> TEMPLATE\_MGR  
 API\_GW --> VERSION\_CTRL  
 API\_GW --> COLLAB\_ENGINE  
 API\_GW --> AB\_TEST\_MGR  
 API\_GW --> ANALYTICS\_SVC  
 API\_GW --> DEPLOY\_MGR  
   
 TEMPLATE\_MGR --> SECURITY\_SCANNER  
 TEMPLATE\_MGR --> COMPLIANCE\_CHECKER  
 DEPLOY\_MGR --> PERFORMANCE\_MONITOR  
 AB\_TEST\_MGR --> OPTIMIZATION\_ENGINE  
 COLLAB\_ENGINE --> WORKFLOW\_ENGINE  
   
 DEPLOY\_MGR --> PROVIDER\_GATEWAY  
 PROVIDER\_GATEWAY --> OPENAI\_ADAPTER  
 PROVIDER\_GATEWAY --> ANTHROPIC\_ADAPTER  
 PROVIDER\_GATEWAY --> GOOGLE\_ADAPTER  
 PROVIDER\_GATEWAY --> AZURE\_ADAPTER  
 PROVIDER\_GATEWAY --> AWS\_ADAPTER  
   
 TEMPLATE\_MGR --> POSTGRES  
 VERSION\_CTRL --> MONGODB  
 COLLAB\_ENGINE --> REDIS  
 ANALYTICS\_SVC --> ELASTICSEARCH  
 PERFORMANCE\_MONITOR --> TIMESERIES  
 TEMPLATE\_MGR --> BLOB\_STORAGE  
   
 WORKFLOW\_ENGINE --> MESSAGE\_QUEUE  
 MESSAGE\_QUEUE --> TASK\_PROCESSOR  
 TASK\_PROCESSOR --> NOTIFICATION\_SVC  
 NOTIFICATION\_SVC --> WEBHOOK\_DISPATCHER  
   
 NOTIFICATION\_SVC --> SLACK\_API  
 NOTIFICATION\_SVC --> TEAMS\_API  
 WORKFLOW\_ENGINE --> JIRA\_API  
 VERSION\_CTRL --> GITHUB\_API  
 ANALYTICS\_SVC --> ANALYTICS\_TOOLS

## HLD (High Level Design)

### System Architecture Overview

The Advanced Prompt Template Management System uses a microservices architecture optimized for collaborative development, performance tracking, and multi-provider deployment with enterprise-grade security and compliance features.

#### 1. Template Management Core

##### Hierarchical Template Organization

class TemplateManager:  
 def \_\_init\_\_(self):  
 self.template\_repository = TemplateRepository()  
 self.metadata\_manager = MetadataManager()  
 self.search\_engine = TemplateSearchEngine()  
 self.security\_scanner = SecurityScanner()  
   
 async def create\_template(self, template\_data: TemplateCreationRequest) -> Template:  
 # Validate template structure and content  
 validation\_result = await self.validate\_template\_structure(template\_data)  
 if not validation\_result.is\_valid:  
 raise TemplateValidationError(validation\_result.errors)  
   
 # Security and compliance scanning  
 security\_result = await self.security\_scanner.scan\_template(template\_data.content)  
 if security\_result.has\_violations:  
 raise SecurityViolationError(security\_result.violations)  
   
 # Create template with metadata  
 template = Template(  
 id=generate\_uuid(),  
 name=template\_data.name,  
 content=template\_data.content,  
 category=template\_data.category,  
 tags=template\_data.tags,  
 metadata=self.extract\_template\_metadata(template\_data),  
 created\_by=template\_data.user\_id,  
 created\_at=datetime.utcnow(),  
 version=1  
 )  
   
 # Store in repository  
 stored\_template = await self.template\_repository.save\_template(template)  
   
 # Index for search  
 await self.search\_engine.index\_template(stored\_template)  
   
 # Track creation event  
 await self.track\_template\_event('template\_created', stored\_template.id, template\_data.user\_id)  
   
 return stored\_template  
   
 async def search\_templates(self, search\_query: TemplateSearchQuery) -> SearchResults:  
 # Build search criteria  
 search\_criteria = self.build\_search\_criteria(search\_query)  
   
 # Execute search with filters  
 search\_results = await self.search\_engine.search(  
 query=search\_query.text,  
 filters=search\_criteria.filters,  
 sort=search\_criteria.sort\_criteria,  
 pagination=search\_query.pagination  
 )  
   
 # Enhance results with performance data  
 enhanced\_results = await self.enhance\_results\_with\_performance(search\_results)  
   
 return SearchResults(  
 templates=enhanced\_results,  
 total\_count=search\_results.total\_count,  
 facets=search\_results.facets,  
 suggestions=search\_results.suggestions  
 )

#### 2. Version Control System

##### Git-Style Versioning for Prompts

class VersionControlService:  
 def \_\_init\_\_(self):  
 self.version\_repository = VersionRepository()  
 self.diff\_calculator = DiffCalculator()  
 self.merge\_resolver = MergeResolver()  
 self.branch\_manager = BranchManager()  
   
 async def create\_version(self, template\_id: str, changes: TemplateChanges, commit\_message: str) -> Version:  
 # Get current version  
 current\_version = await self.version\_repository.get\_latest\_version(template\_id)  
   
 # Calculate diff  
 diff = self.diff\_calculator.calculate\_diff(  
 current\_version.content,  
 changes.new\_content  
 )  
   
 # Create new version  
 new\_version = Version(  
 id=generate\_uuid(),  
 template\_id=template\_id,  
 version\_number=current\_version.version\_number + 1,  
 content=changes.new\_content,  
 diff=diff,  
 commit\_message=commit\_message,  
 author=changes.user\_id,  
 parent\_version=current\_version.id,  
 created\_at=datetime.utcnow(),  
 metadata=changes.metadata\_updates  
 )  
   
 # Validate version integrity  
 validation\_result = await self.validate\_version\_integrity(new\_version)  
 if not validation\_result.is\_valid:  
 raise VersionIntegrityError(validation\_result.errors)  
   
 # Save version  
 saved\_version = await self.version\_repository.save\_version(new\_version)  
   
 # Update template current version  
 await self.template\_repository.update\_current\_version(template\_id, saved\_version.id)  
   
 return saved\_version  
   
 async def create\_branch(self, template\_id: str, branch\_name: str, base\_version\_id: str) -> Branch:  
 # Validate branch name  
 if not self.is\_valid\_branch\_name(branch\_name):  
 raise InvalidBranchNameError(f"Invalid branch name: {branch\_name}")  
   
 # Check if branch already exists  
 existing\_branch = await self.branch\_manager.get\_branch(template\_id, branch\_name)  
 if existing\_branch:  
 raise BranchAlreadyExistsError(f"Branch {branch\_name} already exists")  
   
 # Get base version  
 base\_version = await self.version\_repository.get\_version(base\_version\_id)  
   
 # Create branch  
 branch = Branch(  
 id=generate\_uuid(),  
 template\_id=template\_id,  
 name=branch\_name,  
 base\_version\_id=base\_version\_id,  
 head\_version\_id=base\_version\_id,  
 created\_by=base\_version.author,  
 created\_at=datetime.utcnow(),  
 status='active'  
 )  
   
 # Save branch  
 saved\_branch = await self.branch\_manager.save\_branch(branch)  
   
 return saved\_branch  
   
 async def merge\_branches(self, template\_id: str, source\_branch: str, target\_branch: str, merge\_strategy: str = 'auto') -> MergeResult:  
 # Get branch information  
 source\_branch\_info = await self.branch\_manager.get\_branch(template\_id, source\_branch)  
 target\_branch\_info = await self.branch\_manager.get\_branch(template\_id, target\_branch)  
   
 # Get versions to merge  
 source\_version = await self.version\_repository.get\_version(source\_branch\_info.head\_version\_id)  
 target\_version = await self.version\_repository.get\_version(target\_branch\_info.head\_version\_id)  
   
 # Detect conflicts  
 merge\_analysis = await self.analyze\_merge\_conflicts(source\_version, target\_version)  
   
 if merge\_analysis.has\_conflicts and merge\_strategy == 'auto':  
 return MergeResult(  
 success=False,  
 conflicts=merge\_analysis.conflicts,  
 requires\_manual\_resolution=True  
 )  
   
 # Perform merge  
 merged\_content = await self.merge\_resolver.resolve\_merge(  
 source\_version.content,  
 target\_version.content,  
 merge\_strategy=merge\_strategy  
 )  
   
 # Create merge commit  
 merge\_commit = await self.create\_merge\_commit(  
 template\_id=template\_id,  
 merged\_content=merged\_content,  
 source\_version=source\_version,  
 target\_version=target\_version,  
 target\_branch=target\_branch  
 )  
   
 return MergeResult(  
 success=True,  
 merge\_commit=merge\_commit,  
 merged\_content=merged\_content  
 )

#### 3. A/B Testing Framework

##### Statistical Experimentation Engine

class ABTestingManager:  
 def \_\_init\_\_(self):  
 self.experiment\_repository = ExperimentRepository()  
 self.statistical\_engine = StatisticalEngine()  
 self.traffic\_splitter = TrafficSplitter()  
 self.metrics\_collector = MetricsCollector()  
   
 async def create\_experiment(self, experiment\_config: ExperimentConfig) -> Experiment:  
 # Validate experiment configuration  
 validation\_result = self.validate\_experiment\_config(experiment\_config)  
 if not validation\_result.is\_valid:  
 raise ExperimentConfigError(validation\_result.errors)  
   
 # Calculate required sample size  
 sample\_size = self.statistical\_engine.calculate\_sample\_size(  
 baseline\_rate=experiment\_config.expected\_baseline\_rate,  
 minimum\_detectable\_effect=experiment\_config.minimum\_detectable\_effect,  
 statistical\_power=experiment\_config.statistical\_power,  
 significance\_level=experiment\_config.significance\_level  
 )  
   
 # Create experiment  
 experiment = Experiment(  
 id=generate\_uuid(),  
 name=experiment\_config.name,  
 description=experiment\_config.description,  
 template\_id=experiment\_config.template\_id,  
 control\_version\_id=experiment\_config.control\_version\_id,  
 treatment\_versions=experiment\_config.treatment\_versions,  
 traffic\_split=experiment\_config.traffic\_split,  
 success\_metrics=experiment\_config.success\_metrics,  
 required\_sample\_size=sample\_size,  
 start\_date=experiment\_config.start\_date,  
 end\_date=experiment\_config.end\_date,  
 status='created',  
 created\_by=experiment\_config.user\_id,  
 created\_at=datetime.utcnow()  
 )  
   
 # Save experiment  
 saved\_experiment = await self.experiment\_repository.save\_experiment(experiment)  
   
 return saved\_experiment  
   
 async def start\_experiment(self, experiment\_id: str) -> ExperimentStartResult:  
 experiment = await self.experiment\_repository.get\_experiment(experiment\_id)  
   
 # Pre-flight checks  
 preflight\_result = await self.run\_preflight\_checks(experiment)  
 if not preflight\_result.passed:  
 raise ExperimentPreflightError(preflight\_result.issues)  
   
 # Configure traffic splitting  
 await self.traffic\_splitter.configure\_experiment\_routing(  
 experiment\_id=experiment.id,  
 template\_id=experiment.template\_id,  
 traffic\_split=experiment.traffic\_split,  
 treatment\_versions=experiment.treatment\_versions  
 )  
   
 # Start metrics collection  
 await self.metrics\_collector.start\_experiment\_tracking(experiment)  
   
 # Update experiment status  
 experiment.status = 'running'  
 experiment.actual\_start\_date = datetime.utcnow()  
 await self.experiment\_repository.update\_experiment(experiment)  
   
 return ExperimentStartResult(  
 experiment\_id=experiment.id,  
 started\_at=experiment.actual\_start\_date,  
 estimated\_completion=self.estimate\_completion\_date(experiment)  
 )  
   
 async def analyze\_experiment\_results(self, experiment\_id: str) -> ExperimentAnalysis:  
 experiment = await self.experiment\_repository.get\_experiment(experiment\_id)  
   
 # Collect experiment data  
 experiment\_data = await self.metrics\_collector.get\_experiment\_data(experiment\_id)  
   
 # Perform statistical analysis  
 statistical\_analysis = await self.statistical\_engine.analyze\_experiment(  
 control\_data=experiment\_data.control\_metrics,  
 treatment\_data=experiment\_data.treatment\_metrics,  
 success\_metrics=experiment.success\_metrics  
 )  
   
 # Generate insights and recommendations  
 insights = await self.generate\_experiment\_insights(  
 experiment, experiment\_data, statistical\_analysis  
 )  
   
 return ExperimentAnalysis(  
 experiment\_id=experiment.id,  
 statistical\_results=statistical\_analysis,  
 insights=insights,  
 recommendation=self.generate\_recommendation(statistical\_analysis),  
 confidence\_level=statistical\_analysis.confidence\_level,  
 sample\_size\_achieved=experiment\_data.total\_samples  
 )

#### 4. Multi-Provider Integration

##### Universal LLM Provider Gateway

class ProviderGateway:  
 def \_\_init\_\_(self):  
 self.provider\_adapters = {  
 'openai': OpenAIAdapter(),  
 'anthropic': AnthropicAdapter(),  
 'google': GoogleAdapter(),  
 'azure': AzureAdapter(),  
 'aws': AWSAdapter()  
 }  
 self.load\_balancer = ProviderLoadBalancer()  
 self.cost\_optimizer = CostOptimizer()  
 self.performance\_monitor = ProviderPerformanceMonitor()  
   
 async def execute\_prompt(self, request: PromptExecutionRequest) -> PromptResponse:  
 # Select optimal provider  
 provider\_selection = await self.select\_optimal\_provider(request)  
   
 # Get provider adapter  
 adapter = self.provider\_adapters[provider\_selection.provider\_name]  
   
 # Transform request for provider-specific format  
 provider\_request = await adapter.transform\_request(request)  
   
 # Execute with performance monitoring  
 start\_time = time.time()  
   
 try:  
 response = await adapter.execute\_prompt(provider\_request)  
 execution\_time = time.time() - start\_time  
   
 # Record successful execution metrics  
 await self.performance\_monitor.record\_success(  
 provider=provider\_selection.provider\_name,  
 model=provider\_request.model,  
 execution\_time=execution\_time,  
 token\_usage=response.token\_usage,  
 cost=self.calculate\_cost(response.token\_usage, provider\_selection.provider\_name)  
 )  
   
 return self.transform\_provider\_response(response, request.template\_id)  
   
 except ProviderError as e:  
 # Record failure metrics  
 await self.performance\_monitor.record\_failure(  
 provider=provider\_selection.provider\_name,  
 error\_type=type(e).\_\_name\_\_,  
 error\_message=str(e)  
 )  
   
 # Attempt failover if configured  
 if request.enable\_failover:  
 return await self.execute\_with\_failover(request, provider\_selection.provider\_name)  
   
 raise e  
   
 async def select\_optimal\_provider(self, request: PromptExecutionRequest) -> ProviderSelection:  
 # Get available providers for template  
 available\_providers = await self.get\_available\_providers(request.template\_id)  
   
 # Filter by requirements (model capabilities, cost constraints, etc.)  
 suitable\_providers = self.filter\_suitable\_providers(  
 available\_providers,   
 request.requirements  
 )  
   
 if not suitable\_providers:  
 raise NoSuitableProviderError("No providers match the specified requirements")  
   
 # Multi-criteria decision making  
 selection\_criteria = {  
 'cost': request.optimization\_preferences.cost\_weight,  
 'latency': request.optimization\_preferences.latency\_weight,  
 'quality': request.optimization\_preferences.quality\_weight,  
 'reliability': request.optimization\_preferences.reliability\_weight  
 }  
   
 optimal\_provider = await self.cost\_optimizer.select\_optimal\_provider(  
 suitable\_providers,  
 selection\_criteria,  
 request.expected\_usage  
 )  
   
 return ProviderSelection(  
 provider\_name=optimal\_provider.name,  
 model=optimal\_provider.recommended\_model,  
 estimated\_cost=optimal\_provider.estimated\_cost,  
 expected\_latency=optimal\_provider.expected\_latency  
 )  
  
class OpenAIAdapter(BaseProviderAdapter):  
 def \_\_init\_\_(self):  
 self.client = OpenAI()  
 self.model\_mappings = {  
 'gpt-4': 'gpt-4-1106-preview',  
 'gpt-3.5': 'gpt-3.5-turbo',  
 'gpt-4-turbo': 'gpt-4-turbo-preview'  
 }  
   
 async def execute\_prompt(self, request: ProviderRequest) -> ProviderResponse:  
 # Transform prompt template to OpenAI format  
 messages = self.transform\_to\_messages\_format(  
 request.prompt\_template,  
 request.variables  
 )  
   
 # Execute API call  
 response = await self.client.chat.completions.create(  
 model=request.model,  
 messages=messages,  
 temperature=request.parameters.get('temperature', 0.7),  
 max\_tokens=request.parameters.get('max\_tokens', 1000),  
 top\_p=request.parameters.get('top\_p', 1.0),  
 frequency\_penalty=request.parameters.get('frequency\_penalty', 0),  
 presence\_penalty=request.parameters.get('presence\_penalty', 0)  
 )  
   
 return ProviderResponse(  
 content=response.choices[0].message.content,  
 token\_usage={  
 'prompt\_tokens': response.usage.prompt\_tokens,  
 'completion\_tokens': response.usage.completion\_tokens,  
 'total\_tokens': response.usage.total\_tokens  
 },  
 model\_used=response.model,  
 provider='openai',  
 response\_metadata=self.extract\_metadata(response)  
 )  
   
 def transform\_to\_messages\_format(self, prompt\_template: str, variables: dict) -> list:  
 # Parse template and replace variables  
 rendered\_prompt = self.render\_template(prompt\_template, variables)  
   
 # Convert to OpenAI messages format  
 if '{{system}}' in rendered\_prompt and '{{user}}' in rendered\_prompt:  
 # Multi-turn format  
 parts = rendered\_prompt.split('{{user}}')  
 system\_part = parts[0].replace('{{system}}', '').strip()  
 user\_part = parts[1].strip()  
   
 messages = [  
 {"role": "system", "content": system\_part},  
 {"role": "user", "content": user\_part}  
 ]  
 else:  
 # Simple user message  
 messages = [{"role": "user", "content": rendered\_prompt}]  
   
 return messages

### Performance Analytics and Monitoring

#### Real-Time Metrics Collection

* **Response Time Tracking:** Latency monitoring across all providers
* **Cost Analytics:** Real-time cost tracking and budget alerts
* **Quality Metrics:** Automated quality scoring using various techniques
* **Success Rate Monitoring:** Template performance and error rate tracking
* **Usage Pattern Analysis:** User behavior and prompt effectiveness insights

#### Advanced Analytics Dashboard

* **Provider Comparison:** Side-by-side performance analysis
* **Template Performance Trends:** Historical performance tracking
* **Cost Optimization Insights:** Recommendations for cost reduction
* **A/B Test Results:** Statistical analysis and confidence intervals
* **Collaboration Metrics:** Team productivity and workflow efficiency

### Security and Compliance Framework

#### Content Security Scanning

* **PII Detection:** Automated identification of personal information
* **Sensitive Data Classification:** Content categorization and handling
* **Policy Enforcement:** Automated compliance rule enforcement
* **Audit Trail:** Comprehensive logging of all security events
* **Access Control:** Fine-grained permissions and role management

## LLD (Low Level Design)

### Detailed Component Implementation

#### 1. Template Storage and Metadata Management

##### Advanced Template Repository

```python class TemplateRepository: def **init**(self): self.primary\_db = PostgreSQLConnection() self.document\_store = MongoDBConnection() self.search\_index = ElasticsearchConnection() self.cache = RedisConnection()

async def save\_template(self, template: Template) -> Template:  
 async with self.primary\_db.transaction() as tx:  
 # Save core template data  
 template\_record = await tx.execute(  
 """  
 INSERT INTO templates   
 (id, name, category, created\_by, created\_at, status, folder\_id)  
 VALUES ($1, $2, $3, $4, $5, $6, $7)  
 RETURNING \*  
 """,  
 template.id, template.name, template.category,  
 template.created\_by, template.created\_at, template.status,  
 template.folder\_id  
 )  
   
 # Save template content in document store  
 content\_doc = {  
 'template\_id': template.id,  
 'content': template.content,  
 'variables': template.variables,  
 'metadata': template.metadata,  
 'version': template.version  
 }  
   
 await self.document\_store.templates.insert\_one(content\_doc)  
   
 # Save tags  
 if template.tags