H■ TH■NG G■I Ý MÓN ■N VI■T NAM

TÀI LI■U K■ THU■T VÀ THU■T TOÁN

Tên h■ th■ng:	Vietnamese Food Recommendation System
Phiên b■n:	v4.0
Ngày t■o:	29/06/2025
Tác gi■:	AI Assistant
Công ngh■:	Flask + Machine Learning + AI Agent

M■C L■C

1. T■NG QUAN H■ TH■NG

H■ th■ng Vietnamese Food Recommendation System là m■t ■ng d■ng web thông minh ■■■c phát tri■n ■■ g■i ý món ■n Vi■t Nam phù h■p v■i s■ thích và nhu c■u c■a t■ng ng■■i dùng. H■ th■ng k■t h■p nhi■u k■ thu■t AI và Machine Learning tiên ti■n ■■ t■o ra tr■i nghi■m cá nhân hóa t■i ■u.

1.1 ■■c ■i■m chính

• Hybrid Recommendation System v■i 5+ thu■t toán ML

• AI Agent v■i kh■ n■ng x■ lý ngôn ng■ t■ nhiên

• Vector Database v■i RAG (Retrieval Augmented Generation)

• Cold Start Solution cho ng■■i dùng m■i

• Real-time performance monitoring

• Responsive web interface v■i modern UI/UX

2. KI■N TRÚC T■NG TH■

H■ th■ng ■■■c thi■t k■ theo ki■n trúc microservices v■i các thành ph■n chính:

Thành ph■n	Công ngh■	Ch■c n■ng
Web Framework	Flask	API endpoints và web serving
ML Engine	Scikit-learn, CatBoost	Recommendation algorithms
Deep Learning	TensorFlow/Keras	Neural Collaborative Filtering
Vector DB	ChromaDB	Semantic search và RAG
AI Agent	LLM + Custom Logic	Natural language processing
Database	CSV + Memory Cache	Data storage và caching
Frontend	HTML5 + Modern CSS + JS	User interface

3. CÁC THU■T TOÁN MACHINE LEARNING

3.1 CatBoost Regression

CatBoost là thu■t toán gradient boosting ■■■c s■ d■ng làm engine chính cho rating prediction. ■■■c ch■n vì kh■ n■ng x■ lý categorical features t■t và performance cao.

```
# CatBoost Model Implementation from catboost import CatBoostRegressor
model = CatBoostRegressor( iterations=1000, learning_rate=0.1, depth=6,
l2_leaf_reg=3, border_count=64, thread_count=4 ) model.fit(X_train,
y_train, eval_set=(X_val, y_val), verbose=100)
```

3.2 Collaborative Filtering

Collaborative Filtering ■■■c tri■n khai v■i hai ph■■ng pháp: - User-based CF: Tìm ng■■i dùng t■■ng t■ - Item-based CF: Tìm món ■n t■■ng t■

```
# Collaborative Filtering Implementation from sklearn.metrics.pairwise
import cosine_similarity from sklearn.neighbors import NearestNeighbors #
User-based CF user_similarity = cosine_similarity(user_item_matrix) #
Item-based CF item_similarity = cosine_similarity(user_item_matrix.T) #
KNN for recommendations knn_model = NearestNeighbors( metric='cosine',
algorithm='brute', n_neighbors=20 ) knn_model.fit(user_item_matrix)
```

3.3 Content-Based Filtering

Content-Based Filtering s■ d■ng các ■■c tr■ng c■a món ■n ■■ t■o recommendations: - TF-IDF vectorization cho text features - Numerical features: calories, price, preparation time - Categorical features: difficulty, meal type, cuisine type

```
# Content-Based Filtering from sklearn.feature_extraction.text import
TfidfVectorizer from sklearn.preprocessing import StandardScaler # Text
features tfidf = TfidfVectorizer(max_features=1000, stop_words='english')
text_features = tfidf.fit_transform(recipe_descriptions) # Numerical
features scaler = StandardScaler() numerical_features =
scaler.fit_transform(recipe_numerical_data) # Combine features
content_features = hstack([text_features, numerical_features])
```

4. H■ TH■NG HYBRID RECOMMENDATION

H■ th■ng Hybrid k■t h■p nhi■u ph■■ng pháp recommendation ■■ t■i ■u ■■ chính xác:

Thu■t toán	Tr■ng s■	Mô t■
Collaborative Filtering	30%	User-item và item-item similarity
Content-Based Filtering	25%	Recipe features matching
Matrix Factorization	25%	SVD và NMF decomposition
Deep Learning	20%	Neural Collaborative Filtering

```
# Hybrid Ensemble Implementation class HybridRecommendationSystem: def
__init__(self): self.ensemble_weights = { 'collaborative': 0.3,
'content_based': 0.25, 'matrix_factorization': 0.25, 'deep_learning': 0.2
} def get_hybrid_recommendations(self, customer_id, n_recommendations=10):
# Get recommendations from each method cf_recs =
self.get_collaborative_recommendations(customer_id) cb_recs =
self.get_content_based_recommendations(customer_id) mf_recs =
self.get_matrix_factorization_recommendations(customer_id) dl_recs =
self.get_deep_learning_recommendations(customer_id) # Combine with
weighted ensemble final_scores = self.weighted_ensemble( cf_recs, cb_recs,
mf_recs, dl_recs ) return sorted(final_scores, key=lambda x: x.score,
reverse=True)[:n_recommendations]
```

5. AI AGENT VÀ NATURAL LANGUAGE PROCESSING

AI Agent là thành ph■n x■ lý truy v■n ngôn ng■ t■ nhiên và t■o ph■n h■i thông minh:

• Intent Recognition: Nh■n di■n ý ■■nh ng■■i dùng

• Entity Extraction: Trích xu■t thông tin t■ câu h■i

• Context Awareness: Hi■u ng■ c■nh h■i tho■i

• Multi-turn Conversation: H■ tr■ h■i tho■i nhi■u l■■t

• Dietary Restrictions Processing: X■ lý h■n ch■ dinh d■■ng

• Regional Preference Understanding: Hi■u s■ thích vùng mi■n

```
# AI Agent Implementation class FoodAIAgent: def __init__(self):
self.intent_classifier = self.load_intent_model() self.entity_extractor =
self.load_entity_model() self.context_manager = ContextManager() def
process_query(self, message, customer_id): # Extract intent and entities
intent = self.intent_classifier.predict(message) entities =
self.entity_extractor.extract(message) # Get context context =
self.context_manager.get_context(customer_id) # Generate response based on
intent if intent == 'recipe_recommendation': return
self.get_recipe_recommendations(entities, context) elif intent ==
'nutrition_info': return self.get_nutrition_information(entities) elif
intent == 'cooking_instructions': return
self.get_cooking_instructions(entities) return
self.generate_default_response(message)
```

6. VECTOR DATABASE VÀ RAG

Vector Database (ChromaDB) ■■■c s■ d■ng ■■ l■u tr■ và tìm ki■m semantic c■a thông tin món ■n. RAG (Retrieval Augmented Generation) k■t h■p tìm ki■m vector v■i generation ■■ t■o ra câu tr■ l■i chính xác.

```
# Vector Database + RAG Implementation import chromadb from
sentence_transformers import SentenceTransformer class VectorFoodDB: def
__init__(self): self.client = chromadb.Client() self.collection =
self.client.create_collection("food_recipes") self.encoder =
SentenceTransformer('all-MiniLM-L6-v2') def add_recipe(self, recipe_data):
# Create embedding text = f"{recipe_data['name']}
{recipe_data['description']}" embedding = self.encoder.encode(text) #
Store in vector DB self.collection.add( embeddings=[embedding.tolist()],
documents=[text], metadatas=[recipe_data], ids=[recipe_data['id']] ) def
semantic_search(self, query, n_results=5): query_embedding =
self.encoder.encode(query) results = self.collection.query(
query_embeddings=[query_embedding.tolist()], n_results=n_results ) return
results
```

7. COLD START PROBLEM SOLUTION

Cold Start Problem ■■■c gi■i quy■t thông qua New Customer Registration System v■i các k■ thu■t:

• Profile-based Recommendations: D■a trên thông tin cá nhân

• Demographic Filtering: L■c theo nhóm tu■i và gi■i tính

• Popularity-based Recommendations: Món ■n ph■ bi■n nh■t

• Content-based Initial Matching: Kh■p theo s■ thích ban ■■u

• Active Learning: Thu th■p feedback nhanh chóng

```
# Cold Start Solution Implementation def
get_initial_recommendations(customer_data, randomize=False): # Extract
user preferences age = customer_data.get('age', 25) dietary_restrictions =
customer_data.get('dietary_restrictions', []) health_goals =
customer_data.get('health_goals', []) budget_range =
customer_data.get('budget_range', 'medium') # Filter based on restrictions
filtered_recipes = interactions_df.copy() # Apply dietary filters if
'vegetarian' in dietary_restrictions: filtered_recipes =
filter_vegetarian_recipes(filtered_recipes) # Apply health goal filters if
'weight_loss' in health_goals: filtered_recipes =
filter_low_calorie_recipes(filtered_recipes) # Get top-rated recipes from
filtered set recommendations = get_top_rated_recipes(filtered_recipes,
n=5) return recommendations
```

8. PERFORMANCE OPTIMIZATION

H■ th■ng ■■■c t■i ■u hóa hi■u su■t thông qua nhi■u k■ thu■t:

K■ thu■t	Mô t■	Hi■u qu■
Memory Caching	Cache k■t qu■ recommendations	Gi■m 80% response time
Model Preloading	Load models khi kh■i ■■ng	Lo■i b■ cold start delay
Batch Processing	X■ lý nhi■u requests cùng lúc	T■ng 3x throughput
Lazy Loading	Load data khi c■n thi■t	Gi■m 60% memory usage
Connection Pooling	Tái s■ d■ng database connections	Gi■m overhead
Async Processing	X■ lý b■t ■■ng b■	C■i thi■n user experience

```
# Performance Monitoring Implementation from functools import wraps import time class PerformanceMonitor: def __init__(self): self.metrics = {} def monitor_performance(self, endpoint): def decorator(func): @wraps(func) def wrapper(*args, **kwargs): start_time = time.time() try: result = func(*args, **kwargs) status = 'success' except Exception as e: result = None status = 'error' end_time = time.time() duration = end_time - start_time # Log metrics self.log_metric(endpoint, duration, status) if result is None: raise return result return wrapper return decorator def log_metric(self, endpoint, duration, status): if endpoint not in self.metrics: self.metrics[endpoint] = [] self.metrics[endpoint].append({ 'duration': duration, 'status': status, 'timestamp': time.time() })
```

9. API DOCUMENTATION

H■ th■ng cung c■p RESTful API v■i Swagger documentation:

Endpoint	Method	Mô t■
/api/recommend	POST	L■y recommendations cho user
/api/agent/query	POST	X■ lý natural language query
/api/register-customer	POST	■■ng ký khách hàng m■i
/api/hybrid/recommendations/<id>	GET	Hybrid recommendations
/api/rate-recipe	POST	■ánh giá món ■n
/api/user-profile/<id>	GET	Thông tin profile user
/api/recipe/<id>	GET	Chi ti■t món ■n
/api/search	GET	Tìm ki■m món ■n

10. DEPLOYMENT VÀ MONITORING

H■ th■ng ■■■c thi■t k■ ■■ deployment d■ dàng v■i monitoring toàn di■n:

• Docker containerization cho portable deployment

• Health check endpoints cho monitoring

• Logging system v■i multiple levels

• Error tracking và alerting

• Performance metrics collection

• Auto-scaling capabilities

K■T LU■N

Vietnamese Food Recommendation System là m■t ■ng d■ng AI toàn di■n, k■t h■p nhi■u k■ thu■t Machine Learning và AI hi■n ■■i ■■ t■o ra tr■i nghi■m g■i ý món ■n cá nhân hóa t■t nh■t cho ng■■i dùng Vi■t Nam. H■ th■ng ■ã ■■■c t■i ■u hóa v■ c■ accuracy và performance, có kh■ n■ng scale và maintain d■ dàng. V■i ki■n trúc modular và comprehensive documentation, h■ th■ng có th■ ■■■c m■ r■ng và c■i ti■n liên t■c.