USER GUIDE

Locality Engine v2.0

PRACTICAL EXAMPLES & API REFERENCE

Quick Start Guide

Get up and running with Locality Engine in under 5 minutes.

Basic Usage - Your First Transformation

Import Locality Engine and transform a message to be culturally appropriate.

```
# Import the engine and cultural frameworks from locality_engine
import LocalityEngine, CulturalFramework # Initialize engine for
Bantu/African context engine =
LocalityEngine(culture=CulturalFramework.BANTU_COLLECTIVE) #
Transform a direct message to be more dignified original = "What do
you want?" transformed = engine.transform(original) print(f"Original:
{original}") print(f"Transformed: {transformed}")
```

```
Output:
Original: What do you want?
Transformed: What would you like me to assist you with?
```

Core API Reference

`LocalityEngine(culture, active_factors=None)`

Initialize the Locality Engine with a specific cultural framework.

Parameters:

`culture` (CulturalFramework): Cultural framework to apply (BANTU_COLLECTIVE, AMERICAN_INDIVIDUALIST, INDONESIAN_PANCASILA, JAPANESE_WA)

`active_factors` (List[int], optional): List of factor numbers (1-18) to activate. If None, all factors are active.

Returns:

LocalityEngine instance configured for the specified culture

`transform(message, mode=TransformMode.CONTEXT_AWARE)`

Main transformation method. Adapts message based on cultural context.

Parameters:

`message` (str): Input text to transform

`mode` (TransformMode): Transformation strategy - DETERMINISTIC (rule-based), PROBABILISTIC (natural variation), or CONTEXT_AWARE (intelligent auto-selection, default)

Returns:

str: Culturally-adapted message preserving dignity and meaning

`assess_dignity_risk(message)`

Analyzes message for potential dignity violations across all 18 Cultural Reasoning factors.

Parameters:

`message` (str): Text to assess for cultural risks

Returns:

dict: Contains risk_score (0-100), risk_level (low/medium/high/critical), violated_factors (list), recommendation, and culture

`get_active_factors()`

Returns list of currently active Cultural Reasoning factors with their properties.

Returns:

List[dict]: Active factors with number, name, weight, description, and active status

Practical Examples

Example 1: AI Chatbot Integration

Adapt customer service chatbot responses based on user culture.

```
from locality_engine import LocalityEngine, CulturalFramework # Detect user culture (simplified - in production, use proper detection) user_culture = CulturalFramework.INDONESIAN_PANCASILA # Initialize engine for user's culture engine = LocalityEngine(culture=user_culture) # Chatbot's original response (too direct) bot_response = "I disagree with your assessment. That approach won't work." # Transform to be culturally appropriate adapted_response = engine.transform(bot_response) print(f"Original: {bot_response}") print(f"Adapted: {adapted_response}")
```

```
Output:
Original: I disagree with your assessment. That approach won't work.
Adapted: Perhaps we could find harmony in both views. That approach might face challenges.
```

Example 2: Healthcare Communication

Adjust medical questions to account for cultural pain communication patterns.

```
from locality_engine import LocalityEngine, CulturalFramework #
Initialize for Bantu/African patient engine =
LocalityEngine(culture=CulturalFramework.BANTU_COLLECTIVE) # Standard
Western medical question doctor_question = "Tell me your pain level.
Are you in pain?" # Adapt to account for Factor #2 (Honor - pain
understatement) adapted_question = engine.transform(doctor_question)
# Also assess if question has dignity risks risk_assessment =
engine.assess_dignity_risk(doctor_question) print(f"Original:
{doctor_question}") print(f"Adapted: {adapted_question}")
print(f"Risk: {risk_assessment['risk_score']}/100")
print(f"Recommendation: {risk_assessment['recommendation']}")
```

💡 Healthcare Context

In many cultures, patients understate pain to maintain dignity (Factor #2: Honor). The adapted question should be paired with proactive pain management rather than waiting for patient requests. This approach prevents the 2-3X maternal mortality gap.

Example 3: Business Proposal Screening

Scan business proposals for cultural violations before sending to international partners.

```
from locality_engine import LocalityEngine, CulturalFramework #
Prepare engines for different target audiences japanese_engine =
LocalityEngine(culture=CulturalFramework.JAPANESE_WA)
indonesian_engine =
LocalityEngine(culture=CulturalFramework.INDONESIAN_PANCASILA) #
Original proposal (too direct for Asian markets) proposal = """ We
need an immediate decision on this partnership. Your current approach
is inefficient and needs to change. Our solution is clearly superior.
""" # Assess dignity risks for each culture japanese_risk =
japanese_engine.assess_dignity_risk(proposal) indonesian_risk =
indonesian_engine.assess_dignity_risk(proposal) print("JAPANESE
MARKET") print(f"Risk Score: {japanese_risk['risk_score']}/100
({japanese_risk['risk_level']})") print(f"Violated Factors:
{len(japanese_risk['violated_factors'])}") print() print("INDONESIAN
MARKET") print(f"Risk Score: {indonesian_risk['risk_score']}/100
({indonesian_risk['risk_level']})") print(f"Recommendation:
{indonesian_risk['recommendation']}")
```

```
Output:
JAPANESE MARKET
Risk Score: 65.0/100 (high)
Violated Factors: 3

INDONESIAN MARKET
Risk Score: 58.0/100 (high)
Recommendation: TRANSFORM REQUIRED
```

Example 4: Multi-Culture Support (Advanced)

Transform the same message for multiple cultural contexts simultaneously.

```
from locality_engine import LocalityEngine, CulturalFramework,
TransformMode # Original message message = "Your performance needs
improvement. You're not meeting expectations." # Create engines for
each culture cultures = { "Bantu/African":
CulturalFramework.BANTU_COLLECTIVE, "American":
```

```
CulturalFramework.AMERICAN_INDIVIDUALIST, "Indonesian":
CulturalFramework.INDONESIAN_PANCASILA, "Japanese":
CulturalFramework.JAPANESE_WA } print(f"ORIGINAL
MESSAGE:\n{message}\n") print("=" * 70) # Transform for each culture
for culture_name, culture_framework in cultures.items(): engine =
LocalityEngine(culture=culture_framework) transformed =
engine.transform(message, mode=TransformMode.DETERMINISTIC)
print(f"\n{culture_name.upper()} ADAPTATION:") print(transformed)
```

Example 5: Selective Factor Activation

Activate only specific Cultural Reasoning factors for targeted transformations.

```
from locality_engine import LocalityEngine, CulturalFramework # Focus
only on dignity (Factor 1) and honor (Factor 2) engine =
LocalityEngine( culture=CulturalFramework.BANTU_COLLECTIVE,
active_factors=[1, 2] # Only Meliorism and Honor ) # Check which
factors are active active = engine.get_active_factors()
print(f"Active factors: {len(active)}") for factor in active:
print(f" - Factor #{factor['number']}: {factor['name']} (weight:
{factor['weight']})") # Transform message message = "You're wrong
about this. That's a bad idea." transformed =
engine.transform(message) print(f"\nOriginal: {message}")
print(f"Transformed: {transformed}")
```

Transformation Modes Explained

Mode	Description	Best For
DETERMINISTIC	Rule-based transformations with predictable, consistent results. Same input always produces same output.	Official communications, legal documents, medical records, contracts where consistency is critical
PROBABILISTIC	Natural variation using culturally-appropriate alternatives. Introduces randomness to prevent robotic patterns.	Chatbots, customer service, conversational AI, marketing content where variety improves engagement
CONTEXT_AWARE	Intelligent automatic selection. Analyzes message content and chooses optimal transformation strategy.	General-purpose applications, mixed content types, when you want optimal results without manual mode selection

🎯 Recommendation

Use **CONTEXT_AWARE** mode (default) for most applications. It intelligently selects the best strategy based on message content. Use **DETERMINISTIC** when consistency is critical, and **PROBABILISTIC** for natural conversational variety.

Best Practices

✅ DO: Initialize Engine Once

Create a single LocalityEngine instance per culture and reuse it throughout your application. Initialization loads cultural configurations, so creating new instances repeatedly wastes resources.

```
# Good: Create once, use many times engine =
LocalityEngine(culture=CulturalFramework.BANTU_COLLECTIVE) for message
in messages: transformed = engine.transform(message)
```

✅ DO: Assess Risk Before Important Communications

Always use assess_dignity_risk() before sending high-stakes communications (job offers, medical diagnoses, business proposals). A risk score above 40 indicates transformation is recommended.

```
risk = engine.assess_dignity_risk(message) if risk['risk_score'] > 40:
message = engine.transform(message) send_message(message)
```

✅ DO: Combine with User Preferences

Store user cultural preferences and apply appropriate engine. Allow users to opt-in or opt-out of cultural adaptation. Respect user autonomy while offering dignity-preserving defaults.

❌ DON'T: Assume Culture from Demographics

Never assume cultural framework based solely on race, nationality, or ethnicity. Allow users to self-identify or use behavioral signals. Cultural frameworks are about values and decision-making patterns, not stereotypes.

❌ DON'T: Over-Transform Casual Communication

Not every message needs transformation. Casual, low-stakes communication between peers can remain direct. Reserve transformation for high-stakes contexts or when dignity is at risk.

✅ DO: Log Transformations for Audit

In regulated industries (healthcare, finance, legal), log original and transformed messages with transformation rationale. This provides audit trail and demonstrates cultural intelligence compliance.

```
import json risk = engine.assess_dignity_risk(original_message)
transformed = engine.transform(original_message) audit_log = {
"timestamp": datetime.now().isoformat(), "original": original_message,
"transformed": transformed, "culture": engine.culture.value,
"risk_score": risk['risk_score'], "violated_factors":
risk['violated_factors'] } log_to_file(json.dumps(audit_log))
```

Integration Patterns

Pattern 1: Pre-Processing Layer

Add Locality Engine as a pre-processing step before sending messages to users.

```
# Example: Email notification system def send_email(recipient,
subject, body): # Detect recipient culture (from profile/preferences)
user_culture = get_user_culture(recipient) # Initialize engine for
user's culture engine = LocalityEngine(culture=user_culture) #
Transform subject and body adapted_subject =
engine.transform(subject) adapted_body = engine.transform(body) #
Send adapted message email_service.send( to=recipient,
subject=adapted_subject, body=adapted_body )
```

Pattern 2: Real-Time Chatbot Adaptation

Adapt chatbot responses in real-time based on conversation context.

```
class CulturalChatbot: def __init__(self,
default_culture=CulturalFramework.AMERICAN_INDIVIDUALIST):
self.engines = { culture: LocalityEngine(culture=culture) for culture
in CulturalFramework } self.default_culture = default_culture def
respond(self, user_message, user_culture=None): # Select appropriate
engine culture = user_culture or self.default_culture engine =
self.engines[culture] # Generate response (your AI logic here)
```

```
raw_response = self.generate_response(user_message) # Adapt response
culturally adapted_response =
```