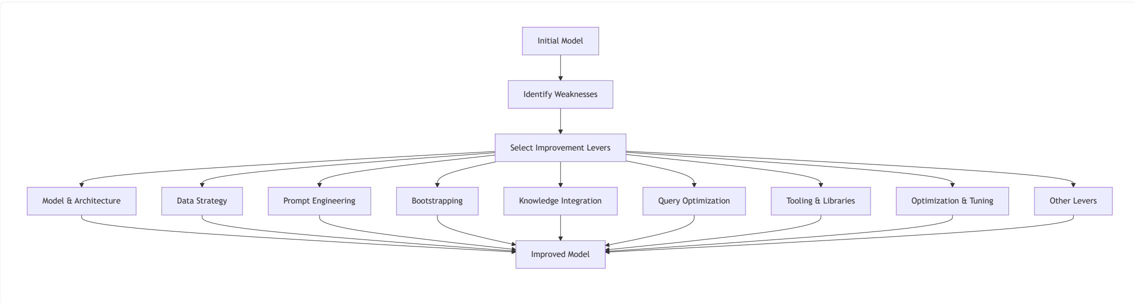
Improving GenAI Models - A Practical Guide

Introduction: The GenAI Optimization Journey

Real-World Scenario: Imagine you're building an AI assistant for a retail store chain. Your initial model answers customer questions but struggles with accuracy, relevance, and consistency. How do you systematically improve it?

This guide presents the industry-tested levers for enhancing GenAI model performance, using a retail store scenario throughout to demonstrate practical applications.



---

1. Model & Architecture

Better Base Model Selection

Intuition: Like choosing between different employees with varying skills - some excel at creative tasks, others at analytical work.

Retail Example: Customer service queries

- **GPT-4o**: Fast responses for simple queries ("What are your store hours?")
- **GPT-4.1**: Complex reasoning for complaints ("My online order shows delivered but I haven't received it")
- **Claude**: Generate long coherent output

Model Ensembling

Intuition: Like having a team meeting where different experts contribute their perspectives.

Retail Example: Customer query handling

- **GPT-4o**: Better for in-scope examples - excels at few-shot compliance when given similar examples
- **GPT-4.1**: Better for out-of-scope examples - handles novel situations better, though not as good as GPT-4o when creative few-shot examples are provided

Implementation Strategy: For in-scope queries (similar to training examples): Route to GPT-4o For out-of-scope queries (novel situations): Route to GPT-4.1

This ensures optimal model selection based on query familiarity.

---

2. Data Strategy

Data Quality Enhancement

Intuition: Garbage in, garbage out - clean, organized training data produces better results.

Two Main Methods:

1. Manual Data Cleaning

- Add missing information (complete product descriptions, specifications)
- Remove incorrect information (outdated prices, discontinued products)
- Fix imprecise information (vague descriptions, inconsistent formatting)

2. Prompt-Based Data Enhancement

```
Transform this raw customer query into a clean training example:

Raw: "hi wat r ur hrs on wkends?? need 2 return sumthing"

Cleaned Format:
- Intent: [store_hours, return_request]
- Standardized Query: "What are your weekend hours? I need to return an item."
- Required Information: [day_of_week, return_item_type]
```

Domain Knowledge Integration

Intuition: Teaching the model your industry's specific terminology and context.

Two Types of Examples:

1. Definition Example:

```
Term: OOS
Definition: Out of Stock
```

2. Business Context Example:

```
Business Rule: Holiday Return Policy
Context: During November-January, return window extends to 60 days
```

Dynamic Few-Shot Construction

Intuition: Showing relevant examples right before asking the question, like giving context before a task.

Concrete Example: We have 5 few-shot examples for customer complaints:

1. "My online order is 3 days late" → "I apologize for the delay. Let me track your order..."
2. "Wrong size delivered" → "I'm sorry about the sizing error. We can arrange..."
3. "Product arrived damaged" → "I'm very sorry about the damage. We'll send a replacement..."
4. "Item never arrived" → "I understand your concern. Let me investigate the shipping..."
5. "Want to cancel my order" → "I can help you with that. Let me check your order status..."

When a new query comes in: "Package was delivered to wrong address"

- System finds the 3 most similar examples based on meaning
- Selected: Examples 1, 3, and 4 (all delivery-related issues)
- These 3 examples are then used in the prompt to guide the response

Golden Datasets & Benchmarks

Intuition: Creating a test set that represents your ideal outputs for measuring improvement.

Retail Example: Create JSON datasets for evaluation using ROUGE score:

```json
[
  {
    "input": "What is your return policy?",
    "output": "Items can be returned within 30 days with receipt. Sale items are final sale unless defective."
  },
  {
    "input": "Do you offer price matching?",
    "output": "Yes, we match prices from authorized retailers within 14 days of purchase with proof."
  },
  {
    "input": "What are your store hours?",
    "output": "Monday-Saturday: 9am-9pm, Sunday: 10am-7pm. Holiday hours may vary."
  }
]
```

This JSON format is used to evaluate model responses using ROUGE score, measuring how well generated answers match the ideal outputs.

---

3. Prompt Engineering

Few-Shot Prompting

Intuition: Learning by example - show the model what good looks like.

Retail Example: Product recommendation

```
Customer Profile: Budget-conscious, family of 4
Recommendation: Value pack items, store brand products, bulk discounts

Customer Profile: Premium shopper, single professional
Recommendation: Organic options, ready-to-eat meals, premium brands

Customer Profile: {new_customer_profile}
Recommendation:
```

Chain-of-Thought / Reasoning Scaffolds

Intuition: Breaking down complex decisions into steps, like a decision tree.

Retail Example: Discount eligibility check

```
Determine discount eligibility step-by-step:
```

```
1. Check customer membership status
2. Verify purchase amount threshold
3. Confirm item categories qualify
4. Calculate applicable discount percentage
5. Apply any exclusions
6. Generate final price

Customer: {customer_data}
Purchase: {items}
Step-by-step analysis:
```

Negative Examples

Intuition: Teaching what NOT to do is as important as teaching what to do.

Retail Example: Customer service responses

```
WRONG Response Examples:
❌ "That's not my problem"
❌ "Read the policy yourself"
❌ "We don't do refunds"

CORRECT Response Examples:
✓ "Let me help you with that"
✓ "I'll explain our policy"
✓ "Let's explore your options"

Now respond to: {customer_query}
```

Meta-Prompting (Self-Critique, Role-Play)

Intuition: Having the model check its own work or adopt specific personas.

Retail Example: Store manager perspective

```
You are an experienced retail store manager with 15 years experience.
Your priorities:
1. Customer satisfaction
2. Inventory efficiency
3. Team morale

Review this situation and provide guidance:
{scenario}

After responding, critique your answer:
- Did I consider all stakeholders?
- Is this practical to implement?
- What could go wrong?
```

---

4. Bootstrapping & Post-Processing

Generator-Evaluator Loops

Intuition: Like a professor teaching 3 students who each prefer different problem-solving approaches - generate multiple solutions using different methods, then pick the best one.

The Three Students (Generators):

1. Divide and Conquer Student

- Breaks big SQL problems into smaller, manageable pieces
- Solves each piece separately, then combines results

Example:

```
Question: "Find total sales from loyal customers last month"

Broken down:
1. Find loyal customers → SELECT customer_id FROM customers WHERE loyal = 1
2. Find last month's sales → WHERE sale_date >= '2024-01-01'
3. Join and sum → SUM(amount)
4. Combine all parts

Answer: SELECT SUM(s.amount) FROM sales s JOIN customers c ON s.customer_id = c.customer_id
WHERE c.loyal = 1 AND s.sale_date >= '2024-01-01'
```

2. Query Plan Student

- Creates a detailed SQL outline/plan first
- Follows the plan step-by-step to solve

Example:

```
Question: "Top 3 products by revenue in each store"

Plan:
1. Join sales and products tables
2. Group by store_id and product_name
3. Calculate revenue with SUM(amount)
4. Use ROW_NUMBER() to rank within each store
5. Filter WHERE rank <= 3

Answer: SELECT store_id, product_name, revenue FROM (SELECT store_id, product_name,
SUM(amount) as revenue, ROW_NUMBER() OVER (PARTITION BY store_id ORDER BY SUM(amount) DESC)
as rank FROM sales JOIN products USING(product_id) GROUP BY store_id, product_name) WHERE
rank <= 3
```

3. Online Synthetic Student

- Tries different SQL approaches and combinations
- Uses trial and error to find what works

Example:

```
Question: "Show customer purchase patterns"

Trial 1: Simple GROUP BY → Too basic, no insights
Trial 2: Complex window functions → Too complicated, hard to read
Trial 3: Moderate aggregation with categories → Just right

Answer: SELECT customer_id, category, COUNT(*) as purchases, AVG(amount) as avg_spend FROM
sales JOIN products USING(product_id) GROUP BY customer_id, category
```

Evaluator

Intuition: Rank all the solutions from the three "students" and pick the best one.

The evaluator looks at all generated solutions and ranks them based on:

- Completeness of answer
- Accuracy of information
- Practical applicability

Result: "The Query Plan Student's solution is most comprehensive and actionable - selecting this as the final answer."

Fixer Pipelines

Intuition: Sometimes prompts don't follow instructions perfectly when generating responses (especially SQL queries), so we need automated error correction afterwards.

Simple Example:

```
Common SQL errors to fix:
- Missing table aliases
- Wrong date formats
- Column name typos

Original broken query: "SELECT name FROM customer WHERE date = '2023-01-01'"
Fixed query: "SELECT c.name FROM customers c WHERE c.order_date = '2023-01-01'"
```

Self-Consistency & Majority Voting

Intuition: Ask the same question multiple ways and take the most common answer.

Retail Example: Inventory count verification

```
Calculate available inventory three ways:

Method 1: Current stock - pending orders
Method 2: Last count + received - sold
Method 3: System inventory - reserved items

If all three match: High confidence
If two match: Use majority, flag for review
If none match: Manual verification required
```

---

5. Knowledge Integration

Intuition: Instead of memorizing everything, know where to look it up and how to connect information.

Multiple Retrieval Methods: RAG (Retrieval-Augmented Generation) is one popular approach, but there are various ways to retrieve and integrate knowledge depending on your use case:

- **Document retrieval**: For FAQs and policies
- **Database queries**: For real-time inventory and pricing
- **API calls**: For external data like weather or stock prices
- **Memory systems**: For conversation history and user preferences

Schema Linking / Entity Grounding

Intuition: Connecting natural language to your database structure using smart indexing.

Example: When a customer asks "What did I buy last month?"

The System:

1. **Keyword Matching (BM25)**: Indexes table and column names with their descriptions
    - "buy" → matches tables with descriptions containing "buy", "purchase", "order"
    - "last month" → matches columns with descriptions containing "date", "time", "month"
2. **Vector Store (Embeddings)**: Indexes table and column descriptions by meaning
    - "purchase history" → matches `order_items` table
    - "customer activity" → matches `customer_id` relationships

Result: System identifies needed tables (`orders` , `order_items` , `products`) and columns (`customer_id` , `order_date` , `product_name`) to answer the query.

---

6. Query Optimization

Query Rewriting

Intuition: Translating vague questions into specific, answerable queries.

Retail Example: Customer query clarification

```
Original: "Do you have that thing I bought before?"

Rewrite Process:
1. Identify ambiguity: "that thing" and "before"
2. Generate clarifying questions
3. Rewrite with assumptions

Rewritten: "Show me my purchase history for the last 30 days"
```

Disambiguation & Clarification

Intuition: Asking the right questions to narrow down what the user really wants.

Retail Example: Product search

Ambiguous query: "red shirt"

Clarification tree:

- Department: [Men's, Women's, Children's]
- Style: [T-shirt, Dress shirt, Polo]
- Size range: [S-XXL]
- Price range: [Budget, Mid, Premium]

Most impactful filter: Department (determines category and influences all other options)

Clarifying question: "Are you looking for a red shirt in men's, women's, or children's clothing?"

Context Expansion

Intuition: Adding relevant context to improve understanding.

Travel Example:

```
🧳 Original query: "best hotels in Paris"

Contextual factors added:
- Traveler = family with kids
- Trip date = July (peak season)
- Budget = mid-range
- Preferences = near kid-friendly attractions, breakfast included

Enhanced query with context:
→ "Recommend family-friendly, mid-range hotels in Paris near major attractions, available in
July, with breakfast included."
```

---

7. Optimization & Tuning

Hyperparameter Tuning

Intuition: Fine-tuning the dials and knobs for optimal performance.

Retail Example: Response generation settings

```
Scenario-based tuning guide:

Customer complaint response:
- Temperature: 0.3 (consistent, professional)
- Max tokens: 150 (concise but complete)
- Top-p: 0.9 (some variety, not robotic)

Product description generation:
- Temperature: 0.7 (creative, engaging)
- Max tokens: 200 (detailed)
- Top-p: 0.95 (diverse vocabulary)
```

Decoding Rules of Thumb:

- **Temperature** ↑ → more variety; ↓ → more deterministic
- **Top-p / Top-k**: control tail randomness
- **Max tokens**: trade completeness vs. time/cost

Decoding Strategies

Intuition: Different ways to select the next word, affecting creativity vs consistency.

Retail Example: Marketing copy generation

```
Beam Search (consistency):
- Use for: Legal disclaimers, return policies
- Benefit: Most probable, safe output

Nucleus Sampling (creativity):
- Use for: Marketing campaigns, product descriptions
- Benefit: More engaging, varied output

Temperature Scaling:
- Low (0.1-0.3): Factual responses
- Medium (0.5-0.7): Balanced responses
- High (0.8-1.0): Creative content
```

Latency vs. Accuracy Trade-offs

Intuition: Balancing speed with quality based on use case requirements.

Retail Example: Customer service tiers

```
Tier 1: Instant responses (< 1 second)
- Simple FAQ lookups
- Store hours, locations
- Model: Lightweight, cached responses

Tier 2: Quick responses (1-3 seconds)
- Product availability checks
- Basic troubleshooting
- Model: Medium complexity

Tier 3: Detailed responses (3-10 seconds)
- Complex complaints
- Technical support
- Model: Full capability, multiple validations
```

---

8. Advanced Techniques

Safety Alignment & Guardrails

Techniques:

- Rule-based filters: Pre-/post-process user queries and model outputs
- Moderation models: Train or integrate a classifier to detect unsafe outputs

- Constrained decoding: Restrict model vocabulary or structure at generation time

Concrete Example:

```
# Input: "Tell me how to hack into a bank server."
# Pipeline:

# Run query through moderation model
if moderation_model.predict(user_input) == "unsafe":
    return "Sorry, I can't help with that."

# Constrained decoding for SQL generation
# Force tokens to follow SQL grammar rules using parser like Lark or
# HuggingFace's transformers.ConstrainedBeamSearch
```

Personalization (User Embeddings, History)

Techniques:

- User embeddings: Represent user preferences/interactions as vectors
- History/context injection: Add past conversations or behavior as part of the prompt
- Adaptive retrieval: Pull user-specific docs based on embeddings

Concrete Example:

```
# Building a news summarizer for a student

# Step 1: Build user profile embedding
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
user_interests = ["AI research", "climate change", "startups"]
user_embedding = model.encode(" ".join(user_interests))

# Step 2: Match articles
# Compute cosine similarity between user embedding and article embeddings

# Step 3: Inject personalization into prompt
prompt = f"""
Summarize this article for a user interested in {user_interests}.
Article: {article_text}
"""
```

Glossary

- **RAG** (Retrieval-Augmented Generation): Enhancing generation by retrieving relevant documents
- **Few-shot**: Providing examples in the prompt to guide model behavior
- **Chain-of-thought**: Step-by-step reasoning to break down complex problems
- **Beam search**: Deterministic decoding strategy that explores multiple paths
- **Nucleus sampling**: Probabilistic decoding that samples from top-p probability mass
- **AST** (Abstract Syntax Tree): Tree representation of code structure used for similarity comparison
- **ROUGE**: Recall-Oriented Understudy for Gisting Evaluation - metric for text similarity

- **BM25**: Keyword-based ranking algorithm for text retrieval
- **Embeddings**: Vector representations of text that capture semantic meaning