- Read
- Write
- Glob
- Grep

DSPy RAG Pipeline

Goal

Build retrieval-augmented generation pipelines with ColBERTv2 that can be systematically optimized.

When to Use

- Questions require external knowledge
- You have a document corpus to search
- Need grounded, factual responses
- Want to optimize retrieval + generation jointly

Inputs

Input	**Type**	**Description**
`question`	`str`	User query
`k`	`int`	Number of passages to retrieve
`rm`	`dspy.Retrieve`	Retrieval model (ColBERTv2)

Outputs

Output	**Type**	**Description**
`context`	`list[str]`	Retrieved passages
`answer`	`str`	Generated response

Workflow

Phase 1: Configure Retrieval

```
import dspy

# Configure LM and retriever
colbert = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')
dspy.configure(
    lm=dspy.LM("openai/gpt-4o-mini"),
    rm=colbert
)
```

Phase 2: Define Signature

```
class GenerateAnswer(dspy.Signature):
    """Answer questions with short factoid answers."""
    context = dspy.InputField(desc="May contain relevant facts")
    question = dspy.InputField()
    answer = dspy.OutputField(desc="Often between 1 and 5 words")
```

Phase 3: Build RAG Module

```
class RAG(dspy.Module):
    def __init__(self, num_passages=3):
        super().__init__()
        self.retrieve = dspy.Retrieve(k=num_passages)
        self.generate = dspy.ChainOfThought(GenerateAnswer)

    def forward(self, question):
        context = self.retrieve(question).passages
        pred = self.generate(context=context, question=question)
        return dspy.Prediction(context=context, answer=pred.answer)
```

Phase 4: Use

```
rag = RAG(num_passages=3)
result = rag(question="What is the capital of France?")
print(result.answer)  # Paris
```

Production Example

```python
import dspy
from dspy.teleprompt import BootstrapFewShot
from dspy.evaluate import Evaluate
import logging

logger = logging.getLogger(__name__)

class GenerateAnswer(dspy.Signature):
    """Answer questions using the provided context."""
    context: list[str] = dspy.InputField(desc="Retrieved passages")
    question: str = dspy.InputField()
    answer: str = dspy.OutputField(desc="Concise factual answer")

class ProductionRAG(dspy.Module):
    def __init__(self, num_passages=5):
        super().__init__()
        self.num_passages = num_passages
        self.retrieve = dspy.Retrieve(k=num_passages)
        self.generate = dspy.ChainOfThought(GenerateAnswer)

    def forward(self, question: str):
        try:
            # Retrieve
            retrieval_result = self.retrieve(question)
            context = retrieval_result.passages

            if not context:
                logger.warning(f"No passages retrieved for: {question}")
                return dspy.Prediction(
                    context=[],
                    answer="I couldn't find relevant information."
                )

            # Generate
            pred = self.generate(context=context, question=question)

            return dspy.Prediction(
                context=context,
                answer=pred.answer,
                reasoning=getattr(pred, 'reasoning', None)
            )

        except Exception as e:
            logger.error(f"RAG failed: {e}")
            return dspy.Prediction(
                context=[],
                answer="An error occurred while processing your question."
            )

def validate_answer(example, pred, trace=None):
    """Check if answer is grounded and correct."""
    if not pred.answer or not pred.context:
        return 0.0

    # Check correctness
    correct = example.answer.lower() in pred.answer.lower()

    # Check grounding (answer should relate to context)
    context_text = " ".join(pred.context).lower()
    grounded = any(word in context_text for word in pred.answer.lower().split())

    return float(correct and grounded)
```

```python
def build_optimized_rag(trainset, devset):
    """Build and optimize a RAG pipeline."""

    # Configure
    colbert = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')
    dspy.configure(
        lm=dspy.LM("openai/gpt-4o-mini"),
        rm=colbert
    )

    # Build
    rag = ProductionRAG(num_passages=5)

    # Evaluate baseline
    evaluator = Evaluate(devset=devset, metric=validate_answer, num_threads=8)
    baseline = evaluator(rag)
    logger.info(f"Baseline: {baseline:.2%}")

    # Optimize
    optimizer = BootstrapFewShot(
        metric=validate_answer,
        max_bootstrapped_demos=4,
        max_labeled_demos=4
    )
    compiled = optimizer.compile(rag, trainset=trainset)

    optimized = evaluator(compiled)
    logger.info(f"Optimized: {optimized:.2%}")

    compiled.save("rag_optimized.json")
    return compiled
```

Multi-Hop RAG

```python
class MultiHopRAG(dspy.Module):
    """RAG with iterative retrieval for complex questions."""

    def __init__(self, num_hops=2, passages_per_hop=3):
        super().__init__()
        self.num_hops = num_hops
        self.retrieve = dspy.Retrieve(k=passages_per_hop)
        self.generate_query = dspy.ChainOfThought("context, question -> search_query")
        self.generate_answer = dspy.ChainOfThought(GenerateAnswer)

    def forward(self, question):
        context = []

        for hop in range(self.num_hops):
            # First hop: use original question
            # Later hops: generate refined query
            if hop == 0:
                query = question
            else:
                query = self.generate_query(
                    context=context,
                    question=question
                ).search_query

            # Retrieve and accumulate
            new_passages = self.retrieve(query).passages
            context.extend(new_passages)

        # Generate final answer
        pred = self.generate_answer(context=context, question=question)
        return dspy.Prediction(context=context, answer=pred.answer)
```

Best Practices

1. **Tune k carefully** - More passages = more context but also noise
2. **Signature descriptions matter** - Guide the model with field descriptions
3. **Validate grounding** - Ensure answers come from retrieved context
4. **Consider multi-hop** - Complex questions may need iterative retrieval

Limitations

- Retrieval quality bounds generation quality
- ColBERTv2 requires hosted index
- Context length limits affect passage count
- Latency increases with more passages