dgroove
Tecnologia emozionale.

ICT Training Center

Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda

Spring AI

Generative Artificial Intelligence con Java

Simone Scannapieco

Corso base per Venis S.p.A, Venezia, Italia

Settembre 2025

Implicit e zero-shot prompting

1. *Implicit prompting*
   - Strategia: atto di fede nei confronti del LLM
     - Utente fornisce solo l'*input* su cui lavorare
     - *Output* del LLM determinato dalle sue capacità emergenti
     - ⚠ LLM determina la *task* dal contenuto dell'*input* e dalla sua formattazione
     - ⚠ Preferire formato strutturato di *markdown* nel definire l'*input* (rendere il processo di cattura dei *pattern* più semplice)
   - Vantaggi
     - Semplice da utilizzare
   - Svantaggi
     - Visto come esercizio di stile
     - Molto limitato a *task* semplici (focalizzate sul puro significato dei *token*)

2. *Zero-shot prompting*
   - Strategia: *implicit prompting* + direttive
     - Utente non fornisce esempi
   - Vantaggi
     - Semplice da utilizzare
   - Svantaggi
     - Molto limitato a *task* semplici
     - Perché non utilizzare esempi…?

- Sistema di traduzione automatica da italiano a multilingua
  1. Creazione modello `TranslationRequest.java` per richiesta traduzione lemma
  2. Creazione *prompt template* per traduzione da italiano a multilingua
  3. Modifiche ad interfaccia ed implementazione del servizio Gemini
  4. Modifica del controllore MVC per servizio Gemini
  5. *Test* delle funzionalità con Postman/Insomnia
- Sistema di *sentiment analysis* per artefatto
  1. Creazione enumeratori `ArtifactType.java` e `Sentiment.java`
  2. Creazione modello di artefatto `Artifact.java`
  3. Creazione modello richiesta per artefatto `ArtifactRequest.java`
  4. Creazione *prompt template* per *sentiment analysis* di tipo *0-shot* per artefatti
  5. Modifiche ad interfaccia ed implementazione del servizio Gemini
  6. Modifica del controllore MVC per servizio Gemini
  7. *Test* delle funzionalità con Postman/Insomnia

TRADUZIONE MULTILINGUA

Modello per richiesta traduzione

```
package it.venis.ai.spring.demo.model;

public record TranslationRequest(String lemma) {}
```

⚠ Il *path* `resources/templates` viene creato automaticamente da Spring AI in fase di creazione del progetto

File `get-lemma-translation-prompt.st`

```
ITA:ciao->ENG:hello,ESP:hola,GER:hallo,FR:bonjour|
ITA:casa->ENG:house,ESP:casa,GER:haus,FR:maison|
ITA:{lemma}->
```

Interfaccia servizio Gemini

```
package it.venis.ai.spring.demo.services;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.ArtifactRequest;
import it.venis.ai.spring.demo.model.DefinitionRequest;
import it.venis.ai.spring.demo.model.DefinitionResponse;
import it.venis.ai.spring.demo.model.Question;

public interface GeminiFromClientService {

    String getAnswerFromClient(String question);

    Answer getAnswerFromClient(Question question);

    Answer getDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getCustomFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getJSONUserFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    DefinitionResponse getJSONOutputConverterFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getTranslationForLemma(TranslationRequest translationRequest);

}
```

Implementazione servizio Gemini - I

```
package it.venis.ai.spring.demo.services;

import java.util.Map;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.template.st.StTemplateRenderer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;
import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.DefinitionRequest;
import it.venis.ai.spring.demo.model.Question;

@Service
public class GeminiFromClientServiceImpl implements GeminiFromClientService {

    private final ChatClient chatClient;

    public GeminiFromClientServiceImpl(ChatClient.Builder chatClientBuilder) {
        this.chatClient = chatClientBuilder.build();
    }

    @Override
    public String getAnswerFromClient(String question) {
        return this.chatClient.prompt()
                .user(question)
                .call()
                .content();
    }
    ...
```

Implementazione servizio Gemini - II

```
    ...
    @Value("classpath:templates/get-lemma-translation-prompt.st")
    private Resource lemmaTanslationPrompt;

    @Override
    public Answer getTranslationForLemma(TranslationRequest translationRequest) {

        List<String> stopSequences = Stream.of(" ", "|", "\n").collect(Collectors.toList());

        String chatResponse = this.chatClient.prompt()
                .options(ChatOptions.builder()
                .model("gemini-2.0-flash")
                .temperature(0.1)
                //.topP(1.0)
                //.topK(30)
                .maxTokens(20)
                //.frequencyPenalty(0.1)
                //.presencePenalty(0.1)
                .stopSequences(stopSequences)
                .build())
                .user(u -> u.text(this.lemmaTanslationPrompt)
                        .params(Map.of("lemma", translationRequest.lemma())))
                .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('{')
                        .endDelimiterToken('}')
                        .build())
                .call()
                .content();

        return new Answer(chatResponse);
    }
}
```

Implementazione controllore REST

```
package it.venis.ai.spring.demo.controllers;

import org.springframework.web.bind.annotation.RestController;
...
import it.venis.ai.spring.demo.services.GeminiFromClientService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController
public class QuestionFromClientController {

    private final GeminiFromClientService geminiService;

    ...

    @PostMapping("/client/definition/json/user")
    public Answer getJSONUserFormatDefinition(@RequestBody DefinitionRequest definitionRequest) {

        return this.geminiService.getJSONUserFormatDefinitionFromClient(definitionRequest);

    }

    ...

    @PostMapping("/client/translate")
    public Answer getTranslationForLemma(@RequestBody TranslationRequest translationRequest) {

        return this.geminiService.getTranslationForLemma(translationRequest);

    }

}
```

- Sistema di traduzione con lingua parametrizzata
  - `language` come parametro aggiuntivo del modello
  - *Output* atteso con `{''lemma'':''aid'',''language'':''ENG''}` è `ITA:aiuto,ESP:aiudo,…`
- Sistema di sinonimi e contrari per lingua italiana

Sentiment analysis

Enumeratore per tipo di artefatto

```
package it.venis.ai.spring.demo.data;

public enum ArtifactType {

    LIBRO("Libro"),
    FILM("Film"),
    SPETTACOLO("Spettacolo"),
    ARTICOLO("Articolo");

    private String artifactType;

    ArtifactType(String artifactType) {
        this.artifactType = artifactType;
    }

    public String getArtifactType() {
        return artifactType;
    }

}
```

Enumeratore per *sentiment*

```
package it.venis.ai.spring.demo.data;

public enum Sentiment {

    MOLTO_POSITIVO("molto positivo"),
    POSITIVO("positivo"),
    NEUTRALE("neutrale"),
    NEGATIVO("negativo"),
    MOLTO_NEGATIVO("molto negativo");

    private String sentiment;

    Sentiment(String sentiment) {
        this.sentiment = sentiment;
    }

    public String getSentiment() {
        return sentiment;
    }

}
```

Modello di artefatto

```
package it.venis.ai.spring.demo.model;

import java.util.Objects;

import it.venis.ai.spring.demo.data.ArtifactType;

public record Artifact(String title, String subtitle, ArtifactType type, String body) {

    @Override
    public String title() {
        return Objects.requireNonNullElse(this.title, "---");
    }

    @Override
    public String subtitle() {
        return Objects.requireNonNullElse(this.subtitle, "---");
    }

}
```

Modello di richiesta per artefatto

```
package it.venis.ai.spring.demo.model;

import java.util.UUID;

public record ArtifactRequest(UUID id, Artifact artifact) {

    public ArtifactRequest(String title, Artifact artifact) {
        this(UUID.randomUUID(), artifact);
    }

}
```

⚠ Il *path* `resources/templates` viene creato automaticamente da Spring AI in fase di creazione del progetto

File `get-artifact-sentiment-prompt.st`

```
Classifica recensioni di artefatti come |MOLTO_POSITIVO|POSITIVO|NEUTRALE|
NEGATIVO|MOLTO_NEGATIVO|.
Restituisci la risposta usando la minima formattazione possibile, senza markdown
 e senza spazi finali.
Recensione {artefatto}: {recensione}
Sentimento:
```

Interfaccia servizio Gemini

```
package it.venis.ai.spring.demo.services;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.ArtifactRequest;
import it.venis.ai.spring.demo.model.DefinitionRequest;
import it.venis.ai.spring.demo.model.DefinitionResponse;
import it.venis.ai.spring.demo.model.Question;

public interface GeminiFromClientService {

    String getAnswerFromClient(String question);

    Answer getAnswerFromClient(Question question);

    Answer getDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getCustomFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getJSONUserFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    DefinitionResponse getJSONOutputConverterFormatDefinitionFromClient(DefinitionRequest definitionRequest);

    Answer getTranslationForLemma(TranslationRequest translationRequest);

    Answer getSentimentForArtifact(ArtifactRequest artifactRequest);

}
```

Implementazione servizio Gemini - I

```
package it.venis.ai.spring.demo.services;

import java.util.Map;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.template.st.StTemplateRenderer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;
import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.DefinitionRequest;
import it.venis.ai.spring.demo.model.Question;

@Service
public class GeminiFromClientServiceImpl implements GeminiFromClientService {

    private final ChatClient chatClient;

    public GeminiFromClientServiceImpl(ChatClient.Builder chatClientBuilder) {
        this.chatClient = chatClientBuilder.build();
    }

    @Override
    public String getAnswerFromClient(String question) {
        return this.chatClient.prompt()
                .user(question)
                .call()
                .content();
    }
    ...
```

Implementazione servizio Gemini - II

```
    ...

    @Value("classpath:templates/get-artifact-sentiment-prompt.st")
    private Resource artifactSentimentPrompt;

    @Override
    public Answer getSentimentForArtifact(ArtifactRequest artifactRequest) {
        List<String> stopSequences = Stream.of(" ", "\n").collect(Collectors.toList());
        Sentiment chatResponse = Sentiment.valueOf(this.chatClient.prompt()
                .options(ChatOptions.builder()
                .model("gemini-2.0-flash")
                .temperature(0.1)
                //.topP(1.0)
                //.topK(30)
                .maxTokens(10)
                //.frequencyPenalty(0.1)
                //.presencePenalty(0.1)
                .stopSequences(stopSequences)
                .build())
                .user(u -> u.text(this.artifactSentimentPrompt)
                        .params(Map.of("recensione", artifactRequest.artifact().body(),
                                       "artefatto", artifactRequest.artifact().type())))
                .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('{')
                        .endDelimiterToken('}')
                        .build())
                .call()
                .content()
                );
        return new Answer(chatResponse.getSentiment());

    }
}
```

Implementazione controllore REST

```
package it.venis.ai.spring.demo.controllers;

import org.springframework.web.bind.annotation.RestController;
...
import it.venis.ai.spring.demo.services.GeminiFromClientService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController
public class QuestionFromClientController {

    private final GeminiFromClientService geminiService;

    ...

    @PostMapping("/client/definition/json/user")
    public Answer getJSONUserFormatDefinition(@RequestBody DefinitionRequest definitionRequest) {

        return this.geminiService.getJSONUserFormatDefinitionFromClient(definitionRequest);

    }

    ...

    @PostMapping("/client/sentiment")
    public Answer getSentiment(@RequestBody ArtifactRequest artifactRequest) {

        return this.geminiService.getSentimentForArtifact(artifactRequest);

    }

}
```

https://github.com/simonescannapieco/spring-ai-base-dgroove-venis-code.git

Branch: `13-spring-ai-gemini-implicit-zero-shot-prompting`