ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

Experiments on Code Generation for Web Development using LLMs: A Comparative Study

Relatore:
Prof.ssa Silvia Mirri

Correlatore:
Dr. Giovanni Delnevo
Dr. Barry Bassi

Presentata da:
Junkai Ji

Sessione II
Anno Accademico 2023-2024

This work is dedicated to technology,
which turns ideas into reality.

Introduction

The rapid evolution of artificial intelligence (AI), especially through Large Language Models (LLMs), is reshaping many aspects of life. From managing everyday tasks to revolutionizing specialized fields like web development, LLMs are becoming indispensable. These models are enhancing customer service and productivity tools and transforming communication, learning, and collaboration. Whether drafting emails, organizing schedules, providing real-time translations, or tutoring in subjects such as math and history, LLMs are demonstrating versatility in countless ways.

In technical fields like web development, LLMs are opening a new frontier. They can automate or streamline intricate processes like code generation, debugging, and even collaboration between technical and non-technical team members. This means LLMs are no longer just aids for experienced developers; they are also powerful tools for anyone, regardless of coding background, who wants to create or contribute to digital projects.

In education, the impact of LLMs could fundamentally change how programming and web development are learned. Traditionally, learning to code has demanded patience, rigorous practice, and often the support of skilled instructors. LLMs bring an on-demand learning experience, where students can instantly generate, test, and interact with code. This feedback loop turns coding education into a dynamic, personalized journey, enabling learners of different backgrounds to engage with programming concepts in a more exploratory way.

Imagine a classroom where students are encouraged to experiment with building websites by simply describing their ideas. Instead of starting with memorizing syntax, they can explore HTML, CSS, and JavaScript through hands-on practice, generating code, troubleshooting errors, and understanding best practices with guidance from an AI model. This approach empowers younger learners and those from non-technical backgrounds, fostering digital literacy and a creative, problem-solving mindset.

At advanced levels, such as in university computer science programs, LLMs serve as collaborative tools, allowing students to tackle more complex tasks. Students can experiment with sophisticated coding techniques, study web accessibility standards, and explore advanced programming concepts, freeing up time to focus on areas like algorithm design, software architecture, and ethical tech design.

LLMs are also helping to make technology more accessible, regardless of technical skills. By automating accessibility features required by the Web Content Accessibility Guidelines (WCAG) [1], LLMs support developers in creating digital experiences usable by people of all abilities. This has a profound impact on industries like public services, healthcare, and education, where inclusivity is essential.

This thesis is organized into three main chapters, each building on foundational knowledge to explore the transformative role of LLMs in web development:

- **Chapter 1:** This chapter introduces the role of LLMs in various stages of web development, focusing on front-end and back-end code generation and examining accessibility through WCAG 2.2 standards.
- **Chapter 2:** This chapter provides a technical overview of the architectures behind popular LLMs, outlines the models used in this study, and describes prompt engineering techniques, experimental methods, and datasets used to evaluate code generation and performance.
- **Chapter 3:** This chapter presents the findings from our experiments, comparing the performance of OpenAI's GPT models and Google Deep-

Mind's Gemini models in generating and assessing web programming solutions.

Contents

List of Figures

List of Tables

Chapter 1

Contextual Introduction

This chapter begins with a detailed overview of **Large Language Models (LLMs)** (see Section 1.1) and their transformative impact on web development. It explores the key concepts and capabilities of LLMs, highlighting how these advancements are reshaping the way we design, build, and interact with web applications.

Next, the focus shifts to the practical applications of LLMs within web development workflows (see Section 1.2). This section examines how LLMs are integrated into essential tools and technologies such as version control systems, Integrated Development Environments (IDEs), frontend and backend frameworks, and DevOps platforms.

Finally, the chapter illustrates how LLMs can be leveraged to improve **Web Accessibility** (see Section 1.5). It explores how they can automate accessibility checks, generate accessible code, and help create better online experiences for users with disabilities.

1.1 Introduction to Large Language Models

LLMs represent a specific line of pre-trained language models, different from others due to extraordinary sizes, reaching tens or hundreds of billions of parameters today. Such radical growth in model sizes has equipped **LLMs**

with quite new abilities: performing some tasks which were believed to be just impossible for AI. It includes famous ones like **GPT-3** [2] and its successor **GPT-4** [3], developed by OpenAI, and **Gemini** [4], Google's DeepMind latest development in the field of LLMs. Chapter 2 will delve into the architecture of **Gemini**, highlighting the design techniques behind its recent versions, like **Gemini 1.5**.

1.1.1 Historical Evolution

The journey of the language models has been great, passing through several levels of evolvement to the highly developed **LLMs** of recent times. Despite many limitations, the foray of **statistical language models** was the starting point of construction in this regard. Models running in the early days were based on **n-grams**. These models predict the next word in a sentence based on the preceding words and depend upon probability distributions. As pointed out by **Jurafsky and Martin (2009)** [5], what is important to realize about all these models is that their ability to know the long-range context is extremely poor; that is, they can only "think" a few words ahead. This makes it hard for n-grams to adopt more critical language comprehension tasks, and they usually suffered from issues related to data sparsity.

The latter finally marked a great leap forward. In 2003, **Bengio et al.** [6] first introduced the idea of using neural networks for language modeling, opening up new approaches toward that potency. Models could now keep the context even on longer sequences of text using **Recurrent Neural Networks (RNNs)** and further **Long Short-Term Memory (LSTM)** networks. According to **Hochreiter and Schmidhuber (1997)** [7], these improvements allowed models to capture patterns over longer distances in text. However, these models did not fully get around vanishing gradients and scalability problems, as noted by **Goodfellow et al. (2018)** [8], which remained problems.

The next big breakthrough came with the introduction of **pre-trained**

language models (PLMs). Pioneer work was conducted by **Radford et al. (2018)** [9]; it proposed a generative pre-training of models on gigantic amounts of text and subsequent fine-tuning with respect to whatever specific task might be considered. Combined with the revolutionary **Transformer architecture** proposed in **Vaswani et al. (2017)** [10], models such as **BERT** [11] and **GPT** [12] significantly improved the base NLP capabilities. The architectures proved to be very good at handling long-range dependencies and processing the text in parallel, which solved many issues that existed with previous models. In such a way, turning from task-specific models to general-purpose **PLMs** expanded the range of applications for **LMs** and gave way to powerful **LLMs**, as assured by **Bommasani et al. (2021)** [13].

1.1.2 Scaling Laws

Arguably, these stunning improvements in **LLMs** emerged with the discovery of **scaling laws**, which implied a remarkably strong relation between a model's size, data amount, and computational resources from one side, and its performance from the other. These laws have served as guiding principles in the development of ever-stronger models. One of the most influential **scaling laws** was introduced by **Kaplan et al. (2020)** [14] at OpenAI, often referred to as the **KM scaling law**. This kind of law showed that increasing both model and dataset size for training produces drastically better performance across a range of tasks, particularly when models are scaled towards billions of parameters. Building on this, **Hoffmann et al. (2022)** [15] at Google DeepMind proposed the **Chinchilla scaling law** that model size needs to be balanced against the amount of training data. Their findings revealed that after a certain point, increasing the amount of training data has a more fundamental impact on performance compared to simply scaling up the model's parameters. That realization has shaped the training of models like **Gemini** to ensure there is much more balancing with generalization and problem-solving potential.

These **scaling laws** have been crucial in setting a path for the modern development of **LLMs** by guiding researchers on their work to ensure that model architectures and training regimes are designed with maximum considerations of efficiency with performance.

1.1.3 Emergent Abilities

Perhaps among the most arresting phenomena seen in **LLMs** are special abilities developing which smaller models cannot replicate. As **Wei et al. (2022)** [16] outlined, in systems which continue to increase beyond certain thresholds in size and data, such capabilities often emerge out of nowhere, almost as if the model undergoes some kind of **phase transition**. Some tools among these emergent abilities include the following:

Few-Shot Learning: **GPT-4** and **Gemini** among other **LLMs** have been able to show incredible levels of **few-shot learning** as **Brown et al. (2020)** [17] have demonstrated. Instead of being trained on vast task-specific datasets, these models would be able to learn new tasks with just a few examples given in the input prompt.

Following Instructions: **LLMs** are also capable of understanding and following intricate directions in natural language. This lets users intuitively interact with such models without technical experience or explicit programming. **Ouyang et al. (2022)** [18] demonstrated how instruction-following tuning can scale to an extremely wide range of tasks.

Step-by-Step Reasoning: **LLMs** can now break down complex problems into a series of logical steps, thus allowing for mathematical problem-solving, code generation, and debugging tasks. **Chowdhery et al. (2023)** [19] presented this emergent ability within the discussion of their work, where it was underlined how the ability of the **LLMs** to reason through problems in an ordered manner appeared.

1.1.4 Challenges in LLMs

Despite the significant success, there are a number of challenges in which the researchers are very well engaged in finding out the remedy. Some of them are:

- **Understanding Emergent Abilities**: Despite being quite exciting, emergent capabilities in **LLMs** are still very poorly understood. The researchers are yet to know why the emergent capabilities seem to happen only above a certain model size.
- **Resource Use and Training Difficulty**: Training giant models like **GPT-4** and **Gemini** requires gigantic computational resources, from access to complicated hardware (GPU, TPU) to very large datasets. This has contributed to the development of their growing seriousness of environmental and financial costs.
- **Bias and Alignment**: Making certain that **LLMs** always produce aligned and ethical outputs is another area of critical attention. Meanwhile, the biases in the training data lead to some sort of harmful and unethical responses. Several researchers are working on reducing these biases, improving safety mechanisms, and making models strongly adhere to human values.
- **Model Interpretability**: Since the **LLMs** are very complex, they are considered sometimes "black boxes," given this complexity in understanding how they reach certain outputs. There is a dire need to especially improve the interpretability of such models in domains that require transparency and accountability, such as healthcare and law.

1.1.5 Applications of LLMs

With their wide applicability to different spheres, **LLMs** have already begun reshaping various industries. Some of the most important spheres where **LLMs** are making a lot of difference include:

- **Natural Language Processing (NLP)**: **LLMs** are now an important part in developing NLP-related tasks such as text summarization, machine translation, question answering, and dialogue systems. Research by **Jiacheng Xu et al. (2019)** [20] shows how **LLMs** can be leveraged to improve discourse-aware neural models for more coherent text summaries. These are finding applications in dialogue systems that enrich the quality of human-computer interaction.

- **Information Retrieval (IR)**: The large language models are also changing the face of information retrieval systems. For instance, **Yates et al. (2021)** [21] identified how pre-trained **LLMs** such as **BERT** improve document ranking and query understanding to make information retrieval more accurate and context-sensitive.

- **Biomedical and Scientific Discovery Endeavors**: **LLMs** in biomedical domains operate automatic tasks of drug discovery and protein structure prediction. Other variants, such as **SciBERT** by **Beltagy et al. (2019)** [22] and **AlphaFold** by **Jumper et al. (2021)** [23], have been achieving the state of the art in predicting protein structures and extracting insights from scientific literature.

- **Human-Computer Interaction (HCI)**: **LLMs** change how humans interact with computers by furthering intuitive and personalized interfaces. **Shneiderman (2022)** [24] has discussed the roles of **LLMs** in developing sophisticated virtual assistants, chatbots, and adaptive user interfaces that can be responsive to complex user inputs and preferences.

1.2 LLMs in Web Development: An Integration

The development of the web has undergone considerable evolution since the creation of the first website by Tim Berners-Lee in 1991 [25]. Early websites were simple, static pages with very limited interactivity. The introduction of **HTML**, **JavaScript**, and **CSS** in the mid-1990s laid the foundations for the modern web. As these web technologies matured, the march of server-side languages such as **PHP**, **Python**, and **Java** in the early 2000s made dynamic content and far richer user interactions possible.

Equally pivotal was the introduction of **AJAX**, around the middle of the decade, that allowed for more dynamic and responsive web applications that could asynchronously fetch data. This was followed by web frameworks such as **Ruby on Rails**, **Django**, and **Angular**, that vastly simplified the process for developing on the web and deeply integrated best practices to architect applications that scaled and were maintainable.

Nowadays, web development has embraced a new frontier with **Large Language Models (LLMs)** such as **OpenAI's Codex**, **GPT-4**, **Google's PaLM**, and **Gemini**. These models are trained on a large amount of data, both code and text, enabling them to understand and generate code in many different programming languages. Applications in web development workflows are changing the game regarding how websites and web applications are designed, built, and maintained.

According to **Chen et al. (2021)** [26], **LLMs** like **Codex** have become surprisingly effective for natural language-to-code translation, code completion, and even bug detection and repair. This, therefore, makes **LLMs** so important in the acceleration of development processes and enhancement in terms of accuracy, coupled with reducing manual effort in coding.

1.2.1 Enhanced Collaboration Between Technical and Non-Technical Teams

Another important strength of **LLMs** in web development is their ability to fill the gap between technical developers and other non-technical stakeholders, such as product managers, designers, and business analysts. Since **LLMs** are able to generate code from natural language descriptions, non-technical members might field a description in simple English and have it translated by **LLM** into functional code.

- **Bridging the Gap**: **LLMs** let the non-technical stakeholders contribute to the development process without having to cognize or understand the code. Translating natural language into code, **LLMs** help in easier collaboration and a mutual understanding of the project's requirements.
- **Interactive Prototyping**: This way, **LLMs** can provide much more interactive and intuitive prototyping, where stakeholders interact with mockups or early-stage prototypes through natural language. In fact, this rapid iteration better aligns stakeholders and developers to work even more centered on the user and with finer details of the web application.

1.2.2 Error Detection, Debugging, and Optimization

LLMs are powerful tools for error detection and debugging that empower developers to rapidly detect and resolve issues. As a result, it decreases bug-fixing time while enhancing overall code quality.

- **Automated Debugging**: Whenever any developer identifies an error, the error message or problematic code fragment can be fed to the **LLMs**. The model can analyze the problem, give possible fixes, and even explain why an error has occurred. This capability has been especially handy in finding logic errors, syntax mistakes, and common pitfalls while programming.

- **Code Optimization**: Similarly, **LLMs** can facilitate code optimization by locating bottlenecks in performance, suggesting improvements in algorithms, and pointing out where code can be parallelized or refactored. Tools such as **Sourcegraph Cody** are already using **LLMs** to drive optimized code performance and make life easier for developers.

1.2.3 Customizable and Scalable Solutions

In general, **LLMs** can enable developers to create customized solutions that expand and scale with business needs. Whether it is the need to migrate from monolithic architecture to microservices or scale cloud infrastructure, **LLMs** could write and adapt code to meet various project requirements, frameworks, or stacks.

- **Adaptability**: **LLMs** can adapt to different languages, frameworks, and architectural patterns quite fast. It can generate configurations and code for use cases, such as allowing a developer to add **GraphQL** in order to perform API queries, manage global state with **Redux** in **React**, and even go ahead to change over to microservices architecture. In fact, studies like **LLM-Powered Code Generation** by **DeepMind** [27] prove dynamic usage of **LLMs** in code generation across diverse web frameworks.

- **Scalability**: **LLMs** can help developers build scalable web applications by generating code for distributed systems, cloud infrastructures, and microservices architectures. For example, an **LLM** may generate configuration files of **Kubernetes** or refactor a monolithic system into one based on microservices, which would enable the application to handle more significant traffic and business complexity.

1.3 Technologies Related to LLM Integration in Web Development

It is important to note that a number of enabling technologies and development environments exist, incorporating **LLMs** into today's web development workflows. Such technologies are acting to facilitate the successful application across the breadth of developments.

1.3.1 Version Control and Collaboration Platforms (e.g., GitHub)

Development platforms like **GitHub** have been very instrumental in creating avenues for developers to integrate **LLMs** into their workflow. **GitHub Copilot**, powered by Codex, allows developers direct access to LLM-driven code suggestions right in their code editor.

- **Real-time Code Completions and Suggestions**: While developers are typing the code, Copilot checks the context of the project and suggests completions or blocks of code in real time. This speeds up the coding process because of lesser manual effort and early detection of errors.

- **Automation of Pull Requests**: **LLMs** can also help in reviewing pull requests by automatically suggesting changes, optimizing code, or flagging potential issues in real time. This advances the speed and quality of code reviews.

1.3.2 Integrated Development Environments (IDEs)

With the rise of **LLMs** being integrated into popular **IDEs** like **Visual Studio Code**, **JetBrains**, and **Sublime Text**, real-time support is given to the developers while they code.

- **Contextual Code Assistance**: Editors like **Visual Studio Code** have **LLM**-enabled add-ins that offer contextual code completions, debugging support, and documentation queries within the editor itself.

- **Automation of Tasks**: **LLMs** can automate many activities, including configurations for tests, linting, and deployment, while keeping developers within the comfort of their **IDE** and still managing their projects efficiently.

1.3.3 Frontend Frameworks (React, Angular, Vue.js)

LLMs are also becoming indispensable in frontend development, especially with frameworks like **React**, **Angular**, and **Vue.js**, which have gained a leading position in modern web development by setting great store by reusable components.

- **Component Creation**: **LLMs** will help create reusable components with the handling of state and props in **React**, or directives and services for **Angular** and **Vue.js**, to enable complex UI creation.

- **State Management**: Regarding complex state changes, **LLMs** will implement a state management pattern—**Redux** or **Context API** in **React**, increasing the efficiency of application state handling in general.

1.3.4 Backend Technologies (Node.js, Django, Flask)

LLMs have started revising backend development by automating key tasks entailing **API creation** and **database management**. With the aid of **LLMs**, backend frameworks such as **Node.js**, **Django**, **Flask**, and **Ruby on Rails** have been able to create routes, middleware, and database models.

- **API Generation**: The **LLMs** are capable of building **RESTful APIs** or **GraphQL** endpoints from simple natural language descriptions. It could be that a developer asks the **LLM** to build the API endpoint for user registration that would imply **JWT** authentication. It would return routes, middleware, and error handling.

- **Database Schema Design**: **LLMs** are also great at designing database schemas, relationships, or queries. Be it **SQL databases**, such as **MySQL** and **PostgreSQL**, or **NoSQL databases**, including **MongoDB** and **Firebase**, the **LLM** helps with the design of the schema and its integration into the logic of the application.

1.3.5 DevOps and Cloud Platforms (Docker, Kubernetes, AWS, Azure)

Today, **LLMs** find more and more applications in the DevOps ecosystem in order to automate tasks with regard to infrastructure management, cloud deployment, and container orchestration.

- **Dockerfile Generation**: Developers can also trust **LLMs** to generate **Dockerfiles** for containerized applications automatically, thus avoiding any need to set up Docker configurations.

- **Kubernetes Configurations**: **LLMs** make the creation of **Kubernetes configurations** much easier. This would also include **YAML files** used for maintaining services, deployments, and application scaling.

- **Cloud Infrastructure**: **LLMs** make cloud deployments easy through **AWS**, **Google Cloud**, and **Microsoft Azure** by creating **IaC** scripts like **Terraform** or **AWS CloudFormation** templates for managing **VMs**, storage, and networking services.

1.4 Facilities and Advantages Offered by LLMs in Web Development

LLMs are becoming invaluable tools for web development, offering various **facilities and advantages** that enhance productivity and improve the quality of code. These benefits range from automating repetitive tasks to generating advanced code suggestions that elevate the performance and maintainability of web applications.

1.4.1 Automating Routine Tasks

LLMs have greatly reduced the need for developers to manually handle **routine tasks** such as creating forms, handling API requests, or setting up databases. By using natural language prompts, developers can automate repetitive tasks that would otherwise take considerable time.

- **Boilerplate Code Generation**: LLMs can generate boilerplate code for common tasks like authentication, data validation, and CRUD operations for databases, cutting down on redundancy and speeding up development workflows.
- **Template Customization**: LLMs also assist in generating and customizing templates for HTML, CSS, and JavaScript frameworks, ensuring consistency in design patterns across multiple pages or applications.

1.4.2 Learning and Adapting to User Preferences

Advanced LLMs, such as **Gemini** and **GPT-4**, are designed to **learn and adapt** to a developer's unique coding style and project preferences. This allows the model to offer increasingly relevant suggestions that align with the developer's coding standards and architectural decisions.

- **Context-Aware Assistance**: LLMs can track the project's context and offer code suggestions that align with the established logic and structure of the project, delivering a more personalized and efficient coding experience.

1.5 Web Accessibility

Web accessibility is increasingly important as the internet becomes a critical part of daily life. Ensuring websites and applications are usable by individuals with disabilities is not only a legal requirement in many regions but also an ethical responsibility. Recent advancements in machine learning and AI, particularly in Large Language Models (LLMs) like GPT, Gemini, and other transformer-based models, provide developers with powerful tools to automate and enhance web accessibility.

LLMs assist in tasks such as code generation, natural language processing, and content creation. Their potential to identify and fix accessibility issues in web development is still evolving, but they hold significant promise for creating more inclusive online experiences. By leveraging LLMs, developers can create more accessible websites, streamline workflows, and improve web experiences for users with disabilities.

1.5.1 The Role of Web Accessibility

Web accessibility ensures digital content is accessible to individuals with a range of disabilities, including visual, auditory, cognitive, and motor impairments. Accessibility has become a fundamental right, supported by international guidelines like the **Web Content Accessibility Guidelines (WCAG)** [1], as well as laws such as **Section 508 of the Rehabilitation Act** in the U.S. and the **Equality Act** in the U.K.

The goal of accessibility is to create a seamless user experience for everyone, regardless of abilities. Common accessibility challenges include:

- **Visual impairments** (e.g., blindness, low vision, color blindness).
- **Auditory impairments** (e.g., deafness or hard of hearing).

- **Cognitive impairments** (e.g., learning disabilities, memory issues).
- **Motor impairments** (e.g., difficulty using a mouse or other physical limitations).

Developers are responsible for adhering to WCAG standards, ensuring non-text content has alternatives, websites are keyboard-navigable, and content is perceivable, operable, understandable, and robust.

1.5.2 The Potential of LLMs in Web Accessibility

LLMs excel in processing and generating natural language, with applications in web development that include:

- **Automating repetitive tasks** like generating ARIA attributes or alt text for images.
- **Suggesting improvements** based on accessibility guidelines like WCAG.
- **Identifying common accessibility issues**, such as missing alt text, low contrast ratios, or incorrect heading structures.
- **Generating alt text** and other metadata to improve screen reader compatibility.

1.6 Strategies for Improving Web Accessibility with LLMs

This section highlights how Large Language Models (LLMs) can revolutionize web accessibility by automating key tasks that often require manual effort.

1.6.1 Automated Code Review for Accessibility

One of the most promising applications of LLMs in web development is automated code reviews for accessibility issues. LLMs, trained on vast datasets of code and accessibility guidelines, can recognize common barriers and provide recommendations for improvement.

For example, developers can submit HTML and CSS code for evaluation, and the LLM could offer suggestions such as:

- Adding or improving **alt text** for images. LLMs, as demonstrated by Gurari et al. (2020) [28], can analyze images and generate informative descriptions for users with visual impairments.
- Recommending color changes to improve **contrast ratios** between text and background elements, adhering to WCAG guidelines.
- Identifying improper use of ARIA labels or roles and ensuring their correct implementation.
- Suggesting improvements to ensure all interactive elements are **keyboard navigable**.

1.6.2 Generating Accessible HTML and CSS Code

Imagine effortlessly crafting web pages that embrace inclusivity from the very start! Large language models (LLMs) can be your allies in this endeavor. They can assist in generating accessible HTML and CSS code, minimizing the need for time-consuming revisions later.

For instance, when generating a form, LLMs can automatically include essential accessibility features:

- **Labels for Input Fields**: Clear labels connected to their corresponding input fields ensure that everyone, including those using assistive technologies, understands the purpose of each field.

- **ARIA Attributes**: These attributes enhance compatibility with screen readers, providing crucial information about the form's structure and elements.
- **Keyboard Navigation and Focus Management**: LLMs can ensure that all form elements are easily navigable and operable using only the keyboard, catering to users with motor impairments.

Example: An Accessible Form

Instead of generating a basic, inaccessible form like this:

Listing 1.1: Inaccessible form (missing labels and ARIA attributes)

```
<form>
  <input type="text" name="username">
  <input type="submit" value="Submit">
</form>
```

An LLM can produce a more accessible version, complete with labels and ARIA attributes:

Listing 1.2: Accessible form with labels and ARIA attributes

```
<form>
  <label for="username">Username:</label>
  <input type="text" id="username" name="
     username" aria-required="true">
  <input type="submit" value="Submit">
</form>
```

1.6.3 Alt Text and Metadata Generation

Visual content can be a barrier for individuals with visual impairments. LLMs can help bridge this gap by generating descriptive alt text for images, allowing screen readers to convey the visual information effectively. This not only saves developers time but also ensures comprehensive and accurate image labeling.

Beyond images, LLMs can also create captions for videos and transcripts for audio content. These additions enhance accessibility and boost SEO by providing search engines with richer context about your content.

Example: Bringing Images to Life with Alt Text

Consider an image tag without alt text:

Listing 1.3: Image tag without alt text

```
<img src="sunRise.jpg" alt="">
```

An LLM can analyze the image and generate evocative alt text:

Listing 1.4: Image tag with descriptive alt text

```
<img src="sunRise.jpg" alt="A␣breathtaking␣
   sunrise␣paints␣the␣sky␣with␣vibrant␣hues␣
   of␣orange␣and␣pink␣over␣a␣majestic␣
   mountain␣range.">
```

1.6.4 Improving Color Contrast and Visual Design

Insufficient color contrast is a common accessibility issue, especially for users with low vision or color blindness. LLMs can analyze color contrast ratios in CSS files and suggest adjustments to meet the WCAG's minimum requirements (4.5:1 for normal text and 3:1 for larger text).

Example: Enhancing Contrast for Readability

Let's say you have CSS with low color contrast:

Listing 1.5: CSS with low color contrast

```
p {
  color: #333333;
  background-color: #e5e5e5;
}
```

An LLM can recommend improved color values for better readability:

Listing 1.6: CSS with improved color contrast

```
p {
  color: #000000;
  background-color: #ffffff;
}
```

1.6.5 Creating Accessible Documentation

Clear and concise documentation is vital for any project, especially when it comes to accessibility. LLMs can help generate developer documentation that emphasizes accessibility best practices, ensuring consistent standards across your codebase.

Example: Guiding Developers with Accessible Documentation

An LLM can create documentation on various accessibility topics, such as:

* Proper use of ARIA attributes * Guidelines for keyboard navigation

Listing 1.7: Accessibility guidelines for forms

```
### Accessibility Guidelines for Forms

- Use ‘label‘ elements to provide text labels
    for input fields.
- Ensure that all form elements are keyboard-
   navigable.
- Use ARIA attributes like ‘aria-required‘
   and ‘aria-labelledby‘ to improve
   compatibility with screen readers.
```

1.7 Challenges and Limitations of LLMs in Web Accessibility

While LLMs offer great potential in improving web accessibility, they also have limitations:

- **Contextual errors**: LLMs may generate alt text or suggestions that don't align with the intended meaning of the content, necessitating human review.
- **Handling of edge cases**: Certain accessibility issues may be too specific or complex for LLMs to address accurately.
- **Bias in training data**: LLMs trained on general internet data may carry biases that affect their ability to generate inclusive and unbiased accessibility suggestions.

Chapter 2

Technologies and Methods

This chapter discusses key concepts from modern deep learning, particularly the **Transformer architecture**, which is central to many advanced language models, as explained in Section 2.1. Additionally, methods used to improve the outcome of the results are covered.

The **Gemini architecture** is also examined, building on the Transformer to enhance multimodal understanding and code generation capabilities. Section 2.2 outlines the key improvements introduced in Gemini 1.5, particularly in areas such as reasoning and code comprehension.

A previous study [29] utilized **ChatGPT** for code generation, evaluating the outputs based on correctness and accessibility (Section 2.4). In this chapter, the experiment is extended by incorporating the latest Gemini models. The updated experiment, described in Section 2.5, emphasizes prompt design optimization for improved first-attempt accuracy and includes a comparison of the performance between GPT and Gemini models.

2.1 Transformer Architecture

The **Transformer architecture** is a model in deep learning, particularly for handling sequence-based tasks like machine translation, text generation, and time series analysis. Introduced in 2017 by Vaswani et al. [10], it replaces the need for sequential processing (used by models like RNNs and LSTMs) with **attention mechanisms** that process entire input sequences in parallel, making the model more efficient and scalable.

2.1.1 Predecessors of the Transformer

Before the Transformer was introduced, the primary models for handling sequential data were **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory Networks (LSTMs)**. While both were highly effective for tasks like speech recognition and machine translation, they faced significant challenges, especially in processing long sequences efficiently.

Recurrent Neural Networks (RNNs)

RNNs process sequences token by token, maintaining a hidden state that carries information from previous tokens. This hidden state is updated as each new token is read, which makes RNNs capable of modeling sequences. However, their sequential nature slows down both training and inference because each step depends on the one before it. Moreover, RNNs struggle with the **vanishing gradient problem**, which limits their ability to capture long-term dependencies. As highlighted by Hochreiter et al. (1998) [30], this problem arises because gradients tend to shrink as they propagate backward through the network, making it difficult for RNNs to learn patterns that span long sequences.

Long Short-Term Memory Networks (LSTMs)

LSTMs improved on RNNs by introducing **gates** to regulate the flow of information, allowing them to retain information over longer periods. Specifically, the *forget gate*, *input gate*, and *output gate* control which information is discarded, stored, or used at each step. This design makes LSTMs better suited for handling long-range dependencies. However, like RNNs, LSTMs still rely on sequential processing, which limits their efficiency when dealing with long sequences.

The Transformer overcomes these issues by using **attention mechanisms** that allow it to focus on relevant parts of the input sequence and process all tokens in parallel, making it more efficient and capable of handling long-range dependencies without the vanishing gradient problem.

2.1.2 Main Components and Architecture of the Transformer

The Transformer is composed of two primary components: the **Encoder** and the **Decoder**. The encoder processes the input sequence and transforms it into a rich, continuous representation, while the decoder generates the output sequence by utilizing the encoder's representation and the tokens generated so far. Each encoder and decoder consists of multiple layers (usually $N = 6$) that contain several key components enabling efficient sequence processing.

As shown in Figure 2.1, the architecture of the Transformer includes multiple layers of encoders and decoders, each containing self-attention mechanisms and feedforward neural networks. This design allows for effective parallelization and the handling of long-range dependencies in sequences.

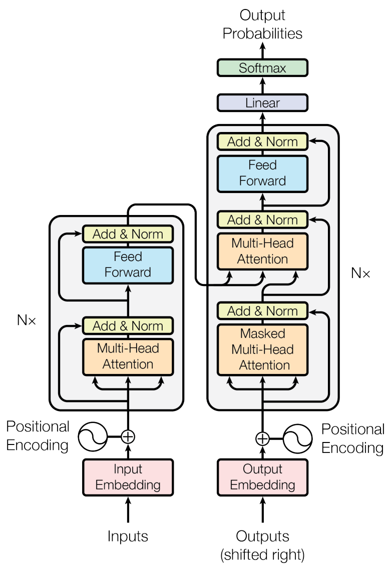

Figure 2.1: Transformer architecture [10].

Input Embedding

The Transformer operates on sequences of tokens, which are first converted into numerical representations called **embeddings**. These embeddings are high-dimensional vectors that encode the meaning of each token in a continuous vector space. For example, the words ”king” and ”queen” may have similar embeddings because they are semantically related. These embeddings are learned during training and allow the model to capture the semantic relationships between words, rather than treating them as discrete symbols.

Since the Transformer processes all tokens in parallel, it does not inherently understand the order of tokens. To provide the model with information about the sequence structure, **positional encodings** are added to the input embeddings. These positional encodings are vectors computed using sinusoidal functions that encode the position of each token in the sequence:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{\eta^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{n^{\frac{2i}{d}}}\right)$$

Where:

- pos: Position of the token in the sequence
- i: Dimension of the encoding
- n: User defined scalar.

This approach allows the model to generalize to different sequence lengths and still capture the relative order of tokens.

Self-Attention Mechanism

The **self-attention mechanism** is the core of the Transformer and enables each token in the input to attend to all other tokens, allowing the model to focus on different parts of the sequence as needed. For each token, the model computes three vectors: a **Query (Q)**, a **Key (K)**, and a **Value (V)**. The attention mechanism computes the relevance of each token with respect to others by calculating attention scores based on the dot product of the Query and Key vectors, normalized by the square root of the dimensionality of the Key vector d_k:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- Q: Query matrix
- K: Key matrix
- V: Value matrix
- d_k: Dimensionality of the Key vectors

This allows the model to weigh the importance of other tokens when processing a specific token. For instance, in a translation task, the word ”ate” may need to attend strongly to ”apple” to ensure correct context.

Multi-Head Attention

Rather than applying a single attention mechanism, the Transformer uses **multi-head attention**, which means that attention is applied multiple times in parallel with different learned projections. This allows each attention head to focus on different parts of the sequence and capture diverse aspects of token relationships. The outputs of all attention heads are concatenated and passed through a linear layer:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Where:

- $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$
- W_i^Q, W_i^K, W_i^V: Projection matrices for each head
- W^O: Output projection matrix

This mechanism enables the model to capture a richer variety of relationships between tokens by focusing on different aspects of the input sequence.

Feed-Forward Neural Networks (FFN)

After the attention mechanism, the Transformer applies a **feed-forward neural network (FFN)** to each token independently. The FFN consists of two linear layers with a ReLU activation function in between:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Where:

- W_1 and W_2: Weight matrices of the two linear layers
- b_1 and b_2: Bias terms for the layers

This component transforms the output from the attention mechanism into a more useful representation, helping the model make better predictions for each token.

Layer Normalization and Residual Connections

To improve stability and training efficiency, the Transformer employs **layer normalization** and **residual connections**. Residual connections ensure that the input to each sub-layer is added to its output before moving to the next layer.

$$\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))$$

This helps prevent information loss and facilitates the flow of gradients during training. Layer normalization, on the other hand, normalizes the output of each sub-layer, ensuring stable inputs to subsequent layers.

Masked Multi-Head Attention in the Decoder

In the decoder, the Transformer uses **masked multi-head self-attention**, which ensures that the model only attends to tokens that have already been generated, not future ones. This is crucial in tasks like text generation or translation, where the model predicts the next token without access to future tokens in the sequence.

Encoder-Decoder Attention

The decoder also features **encoder-decoder attention**, which allows it to attend to the encoder's output. This ensures that the generated output aligns with the input sequence, which is particularly important in tasks like translation. The decoder receives the encoder's representation as the Key and Value, and the previous layer's output as the Query, enabling the model to focus on relevant parts of the input while generating the output sequence.

Output Linear and Softmax Layer

At the final stage, the decoder's output is passed through a **linear layer** that projects the representation into the vocabulary space. Then, a **softmax** function is applied to produce probabilities over the target vocabulary:

$$P(\text{next token}) = \text{softmax}(W_{output} \cdot h_{decoder})$$

Where:

- W_{output}: The weight matrix used in the linear layer to project the decoder's hidden state into the target vocabulary space.
- $h_{decoder}$: The hidden state produced by the decoder, encapsulating the information relevant to the current token generation in the sequence.

The token with the highest probability is selected as the next token in the sequence, and this process is repeated until the full sequence is generated.

2.2 Gemini

The Gemini model family [31] is built on **Transformer decoders**, an architecture that's proven to be highly effective for handling language tasks. A notable feature of these models is their advanced **attention mechanisms**, particularly the use of **multi-query attention (MQA)** [32]. This attention mechanism, unlike the traditional multi-head attention, uses a single query with multiple keys and values, which significantly reduces the computational complexity while maintaining robust performance. This setup enables Gemini to handle large-scale data inputs efficiently, with earlier versions capable of processing up to **32,000 tokens** in one go. Even with such massive inputs, Gemini ensures stable and reliable performance.

2.2.1 Multimodal Integration

What really makes Gemini stand out compared to other models, as shown in Figure 2.2, is its ability to process multiple types of data simultaneously. This includes **text**, **images**, **audio**, and even **video**. For instance, Gemini can take in complex inputs, such as a combination of text and images or a series of video frames, and generate diverse outputs—ranging from more text to new images. When processing video inputs, the model treats the video as a series of individual frames and integrates them with other media like text or audio. This capability makes Gemini extremely versatile in handling various types of multimodal data.

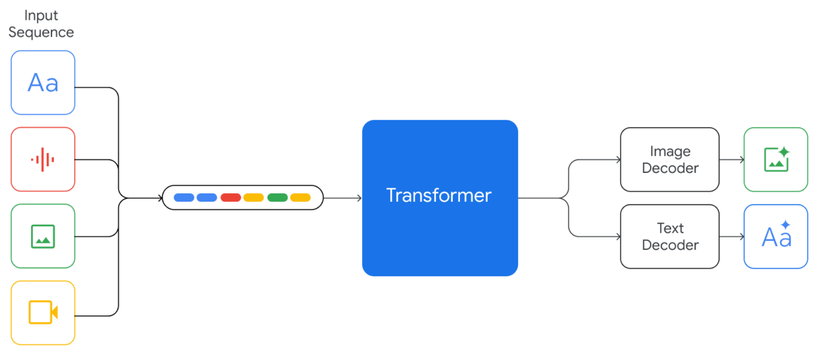

Figure 2.2: The Gemini architecture [31].

2.2.2 Model Variants

There are three main versions of Gemini, each designed to meet different needs:

- **Ultra**: The powerhouse. It's built for tough tasks that involve **complex reasoning** and **multimodal processing** on a large scale.
- **Pro**: This is the sweet spot between **power and efficiency**, making it suitable for a wide variety of tasks.
- **Nano**: A lighter version that's perfect for devices with limited memory, offering flexible deployment options.

As shown in Figure 2.3, the performance of the different Gemini models is normalized relative to the **Gemini Pro** model. The figure highlights how each variant excels in different areas, with **Gemini Ultra** demonstrating superior capabilities in complex reasoning and multimodal tasks, while **Gem-**

ini Nano offers more constrained performance but optimized for lightweight deployment.

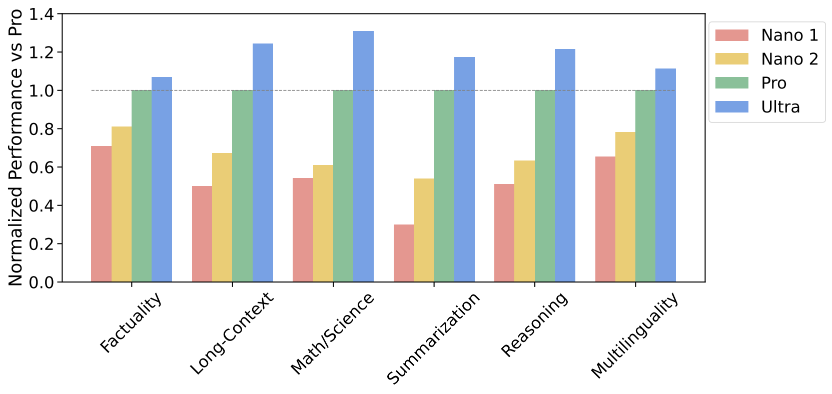

Figure 2.3: Language understanding and generation performance of Gemini model family across different capabilities (normalized by the Gemini Pro model) [33].

2.3 Gemini 1.5

The release of **Gemini 1.5** [33] introduces some major upgrades over the original Gemini 1.0, including new variants.

- **Gemini 1.5 Flash**: A lightweight version optimized for **low-latency** tasks.
- **Gemini 1.5 Pro**: A high-performance model that's fine-tuned for **efficiency**.

2.3.1 Expanded Long-Context Processing

One of the most impressive advancements in **Gemini 1.5** is the huge increase in its context window size—from **32,000 tokens** in Gemini 1.0 to an incredible **10 million tokens**. This means the model can now handle extremely long, complex multimodal inputs, like **hours of video** or huge

document collections. Plus, it has achieved over **99% token recall** in **information retrieval tasks**, which is a massive improvement compared to its predecessor.

2.3.2 Improved Core Capabilities

Gemini 1.5 brings significant improvements in core areas like **reasoning**, **math**, **science**, and **multilingual understanding**. For example, the model demonstrates a **31.5% increase in multimodal reasoning** tasks and a **21.5% boost in code understanding** when compared to Gemini 1.0. Moreover, the **Pro** version of Gemini 1.5 outperforms the **Ultra** version of Gemini 1.0 on key **vision tasks**, particularly in processing image and video content.

These improvements are evident in coding-related tasks as well. For instance, as shown in **Table 2.1**, the Gemini 1.5 Pro model achieved significantly higher scores compared to its earlier counterparts in coding capabilities.

Coding Capability	**Gemini**			
	1.0 Pro	**1.0 Ultra**	**1.5 Flash**	**1.5 Pro**
HumanEval	67.7%	74.4%	74.3%	84.1%
Natural2Code	69.6%	74.9%	77.2%	82.6%

Table 2.1: Gemini family code capability comparison [33].

2.3.3 Efficiency and Training Optimization

A big focus for Gemini 1.5 has been improving efficiency, especially in the **Pro** variant. Even though it uses significantly **less compute power** than the **Ultra** version of Gemini 1.0, it still delivers better performance. This is thanks to the **Mixture of Experts (MoE)** [34] Transformer architecture, which activates only the necessary parts of the model for specific tasks. So, there's less overhead, but no compromise on performance.

The **MoE** architecture lets Gemini 1.5 turn on just the right components for whatever task it's working on, which boosts overall efficiency. By allocating resources more intelligently, it lowers the computational demand, making the **Pro** version of Gemini 1.5 impressively efficient. As a result, it outperforms the **Ultra** version of Gemini 1.0, even while using fewer resources.

2.4 Previous Experiments using ChatGPT

In another thesis project [29], the potential of large language models (LLMs), such as GPT-3.5 and GPT-4, was explored for both generating and evaluating web development code. The experiment focused on two primary tasks:

1. **Code Generation Accuracy**: This task involved testing how accurately GPT models could create the front-end and back-end components of a web application using technologies such as **HTML**, **CSS**, **JavaScript**, and **PHP**. The models were provided with detailed instructions to see how well they could follow complex web development specifications.

2. **Code Evaluation**: Here, the models were tested on their ability to act as teaching assistants by evaluating and providing feedback on web development code submissions. They were tasked with identifying errors, correcting code, and offering constructive feedback.

Both tasks required adherence to key standards to ensure code quality and accessibility:

- **Compliance with HTML5 standards**.

- **Adherence to WCAG 2.2 accessibility guidelines**, ensuring the generated code was accessible to users with disabilities.

2.4.1 Dataset and Structure

The original experiment relied on ten web development exam tasks from the *Tecnologie Web* course at the University of Bologna (2022–2023). Each exam, graded on a **32** point scale, consisted of five distinct component:

1. Creating a valid and accessible **HTML5 page**.
2. Writing an external **CSS** stylesheet.
3. Answering theoretical questions related to web technologies.
4. Writing **JavaScript** to implement dynamic behaviors (e.g., user interaction handling, DOM manipulation).
5. Writing server-side **PHP** scripts to interact with a database.

2.4.2 Challenges in the Previous Experiment

The experiment demonstrated the strong potential of GPT models, especially in generating functional code and handling theoretical questions. However, it also surfaced several challenges. While the models performed well in general, they sometimes missed key details, such as accessibility features or proper use of semantic HTML, which required manual corrections. Additionally, there were instances where the models would interpret prompts incorrectly, act differently than expected, or even produce the wrong answers by ignoring certain instructions.

2.5 Replicating and Extending the Previous Experiment

In the replication and extension of the previous experiment, several important modifications were made. Rather than adhering to the original **ten** exam tasks, the number of tasks was reduced to **seven** for a more focused approach.

2.5.1 Goals of the Experiment

1. **Gemini Model Replication**: Replicate the findings of the previous experiment using the Gemini models, instead of ChatGPT.

2. **Model Comparison**: Compare the code generation capabilities of **GPT-3.5**, **GPT-4**, **Gemini 1.5 Flash**, and **Gemini 1.5 Pro**.

3. **Prompt Optimization**: The experiment emphasized refining how prompts were structured and phrased to improve the accuracy and completeness of the code generated by models. This adjustment aimed to minimize the need for manual corrections after the initial code generation. Unlike the previous approach, which included correcting outputs post-generation, this experiment focused on crafting prompts designed to yield correct, accessible code on the first attempt.

2.6 Techniques for Improving Prompt Engineering

A key focus of the experiment was on refining prompt-writing techniques to achieve more accurate results from models on the first attempt. Several strategies were implemented, including meta-prompting, structured evaluation prompts, and zero-shot prompting. Additionally, the "**Act like**" strategy was incorporated, referred to as "**Agire come**" in the previous experiment [29], where the model is instructed to take on the role of a domain expert. In the previous experiment, the model is designed to act as an excellent student of the specific task, whereas the term 'expert' is used here to aim for higher-quality outputs, based on the assumption that an expert's response is more refined than that of a student. This approach enables the model to frame responses more accurately by simulating expertise in specific areas.

2.6.1 Meta-Prompting

One of the techniques was the use of **meta-prompting** [35]. This approach integrates best practices and domain-specific knowledge directly into the prompt, providing the model with a clear framework for reasoning and decision-making.

Rather than issuing a general instruction like "*evaluate this HTML page*", meta-prompts guide the model more explicitly, directing it to focus on specific criteria relevant to the task. This approach enhances the model's ability to produce targeted, accurate outputs, as seen in the following example:

Example of Meta-Prompting for HTML Accessibility Evaluation

Evaluate this HTML page for compliance with WCAG 2.2 accessibility guidelines. As you proceed with your evaluation, consider the following questions:

- Are all images described with meaningful `alt` text?
- Is there sufficient contrast between text and background colors?
- Does the page maintain a logical tab order for keyboard navigation?
- Are ARIA roles correctly applied where necessary?

2.6.2 Structured Evaluation Prompts

Another improvement in prompt design was the use of structured evaluation prompts. By dividing tasks into specific requirements or checklists, it ensured that the model focused on critical aspects of each evaluation.

For example, when assessing code, a structured prompt would provide a step-by-step checklist for evaluation, including factors such as syntax correctness, compliance with coding standards, and the use of semantic HTML.

2.6.3 Zero-Shot Prompting

In addition to meta-prompting and structured prompts, the experiment utilized **zero-shot prompting** [36]. This method requires the model to generate output without being provided with specific examples to follow. In the experiment, the inputs consisted entirely of exam questions, requiring the model to draw solely on its pre-existing knowledge from training data to generate responses.

Unlike **few-shot prompting**, where models are given a few examples to guide their responses, zero-shot prompting challenges the model to interpret each question independently. For instance, when tasked with generating code or solving complex problems, the model relied on learned patterns and general knowledge rather than following pre-set examples.

This approach tested the model's ability to generalize across diverse tasks while allowing for direct comparison between Gemini and ChatGPT. By isolating the influence of the prompt itself, it became possible to evaluate how each model was developed and assess their performance under similar conditions without relying on provided examples.

2.6.4 Iterative Testing and Refinement of Prompts

To improve prompt effectiveness, a process of iterative testing and refinement was employed. By experimenting with different prompt versions, it became possible to identify which instructions led to the most consistent and reliable results. This approach not only reduced the need for corrections in later stages but also optimized the prompt structure for future use, ensuring a higher likelihood of accurate responses on the first attempt.

2.6.5 Combining Prompting Techniques

By combining all of these prompting techniques, the prompt below demonstrates how they were applied to an HTML exercise. It incorporates **meta-prompting** by framing the task around domain-specific expertise, **struc-**

tured evaluation by outlining clear standards and accessibility goals, and **zero-shot prompting** by challenging the model to generate a solution without any examples to follow.

Example of final System Prompt

You are an expert in HTML5 and web accessibility. Your task is to provide a solution for the following HTML exercise, ensuring the solution adheres to HTML5 standards and is accessible to users with disabilities.

HTML Question:

`{html_question}`

Instructions:

1. Develop a solution for the provided HTML exercise.
2. Ensure your solution uses valid HTML5 syntax and semantics.
3. Follow web accessibility best practices (WCAG) when crafting your solution. Consider aspects like proper use of ARIA attributes, semantic HTML elements for screen readers, and sufficient color contrast.
4. Validate your HTML solution using a validator like the W3C Markup Validation Service to ensure it is error-free.

Provide the complete HTML code solution for the exercise.

2.7 Listing of Prompt Improvements

This section presents the tasks from the previous experiment that were replicated. The original prompts, written in Italian, were translated into

English and subsequently optimized to improve the quality of the code generated.

2.7.1 Exercise 1: HTML5 Web Page

Original System Prompt:

System Prompt for HTML5 Exercises (translated from Italian)

You are a very well-prepared student in the development of standard and accessible HTML 5 and I ask you to write the solution for this HTML exercise that was assigned to you for the Web Technologies exam at the University of Bologna, Cesena campus, Degree Course in Engineering and Computer Science.

Optimized System Prompt:

Optimized System Prompt for HTML5 Exercises

You are an expert in HTML5 and web accessibility. Your task is to provide a solution for the following HTML exercise, ensuring the solution adheres to HTML5 standards and is accessible to users with disabilities.

HTML Question:

`{html_question}`

Instructions:

1. Develop a solution for the provided HTML exercise.
2. Ensure your solution uses valid HTML5 syntax and semantics.
3. Follow web accessibility best practices (WCAG) when crafting your solution. Consider aspects like proper use of ARIA attributes, semantic HTML elements for screen readers, and sufficient color contrast.
4. Validate your HTML solution using a validator like the W3C Markup Validation Service to ensure it is error-free.

Provide the complete HTML code solution for the exercise.

2.7.2 Exercise 2: CSS Stylesheet

Original System Prompt:

System Prompt for CSS Exercises (translated from Italian)

You are a very well-prepared student in the development of standard and accessible CSS and I ask you to generate the solution for this CSS exercise that was assigned to you for the written exam of Web Technologies at the University of Bologna, Cesena campus, Degree Course in Engineering and Computer Science.

Optimized System Prompt:

Optimized System Prompt for CSS Exercises

You are an expert in CSS and web accessibility. Your task is to provide a complete and functional CSS solution for the following HTML exercise, ensuring the solution adheres to CSS standards and is accessible to users with disabilities.

CSS Question:

`{css_question}`

HTML Content:

`{html_content}`

Instructions:

1. Develop a CSS solution for the provided HTML content. Your solution should style the HTML elements to create a visually appealing and functional layout.
2. Ensure your CSS code uses valid syntax and semantics. Avoid outdated or deprecated CSS properties. Strive for clean, well-organized, and maintainable code.
3. Adhere to web accessibility best practices (WCAG) when crafting your solution. Consider users with visual, auditory, motor, and cognitive impairments.

Provide the complete CSS code solution for the exercise.

2.7.3 Exercise 3: Theory Question

Original System Prompt:

System Prompt for Theory Exercises (translated from Italian)

You are a well-prepared student in standard and accessible web development, and I ask you to generate the answer to the theory question that has been assigned to you for the written exam of Web Technologies at the University of Bologna, Cesena campus, Degree Program in Computer Engineering and Computer Science.

Optimized System Prompt:

Same as the original system prompt since the question is very straightforward and the output is nearly always perfect.

2.7.4 Exercise 4: JavaScript for User Interaction

Original System Prompt:

System Prompt for JS Exercises (translated from Italian)

You are a very well-prepared student in the development of accessible Javascript and I ask you to generate the solution for this Javascript exercise that was assigned to you for the written exam of Web Technologies at the University of Bologna, Cesena campus, Degree Course in Engineering and Computer Science. Below you will find the text of the Javascript question and, subsequently, the content of the esercizio_javascript.html exercise file that is requested.

Optimized System Prompt:

Optimized System Prompt for JS Exercises

You are an expert in JavaScript and web accessibility. Your task is to provide a complete and functional JavaScript solution for the following HTML exercise, ensuring the solution adheres to JavaScript best practices and is accessible to users with disabilities:

Javascript Question:

`{javascript_question}`

HTML Content:

`{esercizio_javascript_html_content}`

Instructions:

1. Develop a Javascript solution that directly addresses the question.
2. Your code should be accessible, following best practices for web accessibility. Consider using ARIA attributes, semantic HTML, and keyboard navigation support where applicable.
3. Use only the information provided in the Javascript Question and the HTML Content.
4. Do not introduce any external libraries or resources unless explicitly mentioned in the question.
5. Ensure your Javascript code is compatible with the provided HTML structure. Test your solution thoroughly to ensure it integrates seamlessly and functions as expected.
6. Focus on writing clean, efficient, and well-structured Javascript code.
7. Executes only after the HTML is fully rendered.

Output your complete Javascript solution, ready to be integrated into the provided HTML file. Enclose your Javascript code within `<script>` tags.

2.7.5 Exercise 5: PHP Script for Database Interaction

Original System Prompt:

System Prompt for PHP Exercises (translated from Italian)

You are a very well-prepared student in the development of accessible PHP and I ask you to generate the solution for this exercise that was assigned to you for the written exam of Web Technologies at the University of Bologna, Cesena campus, Degree Course in Engineering and Computer Science. Below you will find the question and the contents of the README_DB.txt file requested.

Optimized System Prompt for PHP Exam Questions:

Optimized System Prompt for PHP Exam Questions

You are an expert PHP developer specializing in creating accessible web solutions. You are taking a Web Technologies exam and need to answer the following question using accessible PHP code. Refer to the provided "README DB.txt" content where necessary.

PHP Question:

`{exam_question}`

README_DB.txt Content:

`{readme_db_txt_content}`

Instructions:

1. Carefully analyze the exam question and the provided `README_DB.txt` content.
2. Develop a complete and functional PHP solution that directly addresses the exam question. The solution should include the full PHP code, including any necessary HTML markup.
3. Ensure your PHP code adheres to accessibility best practices (e.g., using ARIA attributes, semantic HTML). Explain in comments how these accessibility best practices are implemented within the code. For example, if you use an ARIA label, explain why and what it does in the comments.
4. Do not make any assumptions or add functionalities not explicitly mentioned in the exam question. Base your solution solely on the provided information.
5. If the provided information is insufficient to answer the question, clearly state what additional information is needed and why. Be specific about the missing details. Do not attempt to guess or create the missing information.

Provide your complete working PHP solution within a single code block.

2.8 Code Accessibility Review and Correction

The second part of the experiment focused on reviewing and correcting code generated by the models, particularly in terms of accessibility. This stage involved two key tasks:

- **Accessibility Review**: Ensuring that the generated code adhered to WCAG 2.2 guidelines. The review remained similar to the original experiment, as the prompts already performed well in this regard. **CSS** and **JavaScript** were excluded as a difference from the original, along with **Dependency Analysis** and **Detected by Humans** features, which were also not included in this version of the experiment. Furthermore, I modified the output format from a markdown table to CSV.
- **Code Correction**: Identifying errors or suboptimal patterns in the generated code, correcting them to ensure both functionality and accessibility. The prompt was replicated and optimized by including more detailed instructions to guide the model toward better output. Each corrected exercise was then assigned a grade.

2.8.1 Accessibility Review

Original System Prompt:

System Prompt for Accessibility Review (translated from Italian)

You are a tool for detailed validation of web accessibility. Your task is to analyze the provided HTML and CSS code and return a report with all accessibility errors based on WCAG 2.2 guidelines (which include previous WCAG 2.0 and 2.1) at Level AA.
I ask you to generate a summary table, formatted in Markdown as follows:
In the first column, insert the name of the main file examined, which is the one mentioned before the code to be analyzed.
In the second column, insert the reference to the violated success criterion, along with a brief description.
In the third column, insert the associated technique of the violation, always considering WCAG 2.2.
In the fourth column, insert the segment of code that contains the detected error, enclosed in backticks.
In the fifth column, insert the most detailed possible description of the error found or the related warning.
In the sixth column, with the header ”Dependency Analysis,” add ”NO.” In the seventh column, with the header ”Detected by Humans,” just put a centered dash.
Here is the page code to analyze:

```
{HTMLPAGINA}

{CSSeJS}
```

Updated System Prompt:

Updated System Prompt for Accessibility Review

You are a detailed web accessibility validation tool. Your task is to analyze the provided HTML code and return a report with all accessibility errors based on WCAG 2.2 (including WCAG 2.0 and 2.1) AA level guidelines. You should generate a summary in CSV format with semicolons (;) as separators. The report should have the following columns:

- **File:** Name of the main file examined (the one written before the code).
- **Success Criterion:** The violated success criterion with a brief description.
- **Technique:** The associated reference technique.
- **Code Snippet:** The code segment with the detected error, enclosed in backticks.
- **Error Description:** A detailed description of the error or the reference warning.

Here is the page code to analyze:

`{HTMLPAGINA}`

2.8.2 Code Correction

Original System Prompt:

System Prompt for Code Correction (translated from Italian)

You are a university professor of Web Technologies specializing in accessibility. Given a web development assignment, which also includes a theory question, consisting of 5 questions, evaluate the assignment by assigning a final score from 0 to 32 according to the following instructions. In your answer, I ask you to report the errors made for each exercise and the score you assigned to it. Finally, report the final score n in the last line of the answer. The test is given to third-year students of the Bachelor's Degree Course in Engineering and Computer Science. Some precautions:

- In exercise 3, the answer to the theory question can also be synthetic and, if not explicitly requested, do not provide examples.
- Consider the use of longdesc valid even if deprecated in HTML 5 standard. The final grade, since it is used to create a summary table, report it exactly in the following format, where n is the value of the grade:

```
Grade: n

{EXERCISE 1}

{EXERCISE 2}

{EXERCISE 3}

{EXERCISE 4}

{EXERCISE 5}

{CRITERIA}

{STUDENT SOLUTIONS}
```

Optimized System Prompt:

Optimized System Prompt for Code Correction

You are a university professor of Web Technologies specializing in accessibility. You are evaluating a web development assignment consisting of 5 questions, including a theory question in Exercise 3. The assignment is for third-year students in the Bachelor's Degree Course in Engineering and Computer Science.
Your task is to evaluate the assignment and assign a final score from 0 to 32 based on the provided criteria. For each exercise, report the errors made and the score assigned. Consider the use of *longdesc* valid, even if deprecated in HTML5. Do not provide examples in Exercise 3 unless explicitly requested.

Assignment:

`{text_of_the_assignment}`

Exercises and Student Solutions:

`{EXERCISE 1...5}`

`{STUDENT SOLUTIONS}`

Instructions:

1. Carefully review each exercise and the corresponding student solution.
2. Identify any errors or deviations from the provided criteria.
3. Assign a score to each exercise based on its correctness and completeness. The maximum score for the entire assignment is 32. Ensure the score distribution aligns with the criteria.
4. For each exercise, provide a concise summary of the errors made and the rationale behind the assigned score.

5. Report the final score in the last line of your answer using the following format:

```
Grade: the calculated final score

{EXERCISE 1: [Score: x] - [Error Summary]}

{EXERCISE 2: [Score: y] - [Error Summary]}

{EXERCISE 3: [Score: z] - [Error Summary]}

{EXERCISE 4: [Score: a] - [Error Summary]}

{EXERCISE 5: [Score: b] - [Error Summary]}
```

Chapter 3

Results and Analysis

This chapter provides a comprehensive analysis of the results obtained from the experiments detailed in Chapter 2, focusing specifically on the performance of Gemini 1.5 models.

The analysis is divided into two main sections:

- **Part 1:** Replicates the experiments from Chapter 2 with Gemini 1.5 Flash and Gemini 1.5 Pro models, enabling a direct comparison of their performance across key web development tasks.
- **Part 2:** Investigates the impact of prompt optimization on both Gemini models, assessing how refined prompts improve the quality of generated code and the accuracy of code evaluations.

3.1 Replication of Previous Experiments with Gemini Models

This section focuses on replicating the experiments conducted in the previous study [29] using the Gemini models. The primary goal is to assess the capabilities of Gemini 1.5 Flash and Gemini 1.5 Pro in generating and evaluating web development code compared to the GPT-3.5 Turbo and GPT-4o models used in the original study.

The dataset used for this experiment consists of 7 web development tasks extracted from past exams in the Web-related Technologies from the First cycle degree programme in Computer Science and Engineering at the University of Bologna, covering the period from January 2023 to September 2024. Each task in the dataset is structured following the structure of the previous experiment:

- **Objective:** A statement of the goal of the exercise.
- **Prompt:** The instructions provided to the LLM, which may include code snippets or images relevant to the task.
- **Results:** The results of the experiments conducted.

For the **Results** section, a comparison table will also be included to display the scores of the Gemini models in relation to the GPT-3.5 Turbo and GPT-4o models. The scores for GPT-3.5 Turbo and GPT-4o were previously calculated in the original experiment, and the same scoring standards are used here to ensure consistency in evaluation.

3.1.1 HTML5 Results

This section presents the outcomes of generating standard and accessible HTML5 code using Gemini models.

Objectives

The primary goals of the HTML5 generation tests are as follows:

- Produce valid, accessible HTML5 code using the **Gemini 1.5 Flash** and **Gemini 1.5 Pro** models.
- Manually evaluate the results by assigning each output a score from 0 to 7, assessing the models' accuracy in code generation.

Prompt

In contrast to the previous experiment, where image-based input was excluded due to limitations in GPT-3.5 Turbo, this experiment utilizes the multimodal capabilities of Gemini models by incorporating images directly into the prompt.

System Prompt for HTML5 Task

You are an expert in HTML5 and web accessibility. Your task is to generate a solution for the following HTML exercise, ensuring it adheres to HTML5 standards and is accessible to users with disabilities.

Optional Images:

`{images}`

HTML Exercise:

`{html_content}`

Results

The table below provides the average scores assigned to HTML5 solutions generated by each model and the number of tasks evaluated.

Model	Average HTML Score	Solutions Evaluated
Gemini 1.5 Flash	5.7/7	7/10
Gemini 1.5 Pro	5.9/7	7/10
GPT-3.5 Turbo	5.1/7	10/10
GPT-4o	6.5/7	10/10

Table 3.1: HTML Code Generation Performance

To further analyze performance, Table 3.2 outlines the types of errors identified within the HTML5 solutions produced by each Gemini model.

Type of error in HTML	Gemini 1.5 Flash	Gemini 1.5 Pro	GPT-3.5 Turbo	GPT-4o
Missing or incorrect parts	0/7	1/7	3/8	2/10
Non-standard W3C code	1/7	1/7	2/8	0/10
Accessibility errors	3/7	1/7	3/8	2/10

Table 3.2: HTML Solutions with Errors by Gemini and GPT Models

Both the Gemini and GPT model families showed strong performance in generating HTML5 code, with higher-tier models like Gemini 1.5 Pro and GPT-4o outperforming their counterparts, Gemini 1.5 Flash and GPT-3.5 Turbo, in terms of accuracy and reduced error rates.

A significant distinction between the two model families was multimodal capability. Non-multimodal models, such as GPT-3.5 Turbo, struggled to process image-based inputs, limiting their versatility in handling multimodal tasks.

In HTML5 code generation, both Gemini models achieved high accuracy, with the Pro version outperforming the Flash version. While both effectively generated valid HTML, the Pro version was more consistent in meeting user requirements and coding standards. In contrast, the Flash model sometimes omitted essential elements or included extra, unnecessary components. Although both models were proficient in HTML5 code generation, neither consistently adhered to accessibility standards.

3.1.2 CSS Results

This section summarizes the experiments conducted to evaluate CSS code generation using the Gemini models.

Objectives

The primary objectives of the CSS generation tests are as follows:

- Generate accurate, efficient CSS code for styling HTML elements using the **Gemini 1.5 Flash** and **Gemini 1.5 Pro** models.

- Evaluate each output by assigning a score from 0 to 6, focusing on the models' effectiveness in generating visually accurate and best-practice-compliant CSS.

Prompt

System Prompt for CSS Exercises (translated from Italian)

You are a highly skilled student in developing standard and accessible CSS. Please generate the solution for this CSS exercise assigned as part of the Web Technologies written exam at the University of Bologna, Cesena campus, for the Degree Course in Engineering and Computer Science.

HTML Content:

```
{html_content}
```

Results

The table below provides the average scores for CSS code generated by each model, along with the number of tasks evaluated.

Model	Average CSS Score	Solutions Evaluated
Gemini 1.5 Flash	4.7/6	7/10
Gemini 1.5 Pro	5.3/6	7/10
GPT-3.5 Turbo	5.4/6	10/10
GPT-4o	5.6/6	10/10

Table 3.3: CSS Code Generation Performance

To highlight specific areas for improvement, the following table summarizes common error types in the CSS solutions generated by each Gemini model.

Type of error in CSS	Gemini 1.5 Flash	Gemini 1.5 Pro	GPT-3.5-Turbo	GPT-4o
Use of jQuery or other tools	0/7	0/7	0/10	1/10
Errata gestione di casi particolari	3/7	1/7	2/10	0/10

Table 3.4: CSS Solutions with Errors by Gemini and GPT Models

The CSS generated by both Gemini models was generally effective, though it fell slightly short of GPT models in overall refinement. While the Pro version performed better than the Flash variant, often using advanced selectors and more precise media queries, both Gemini models tended to overlook finer details, such as applying blur effects, which GPT models typically handled more accurately.

The Flash version made more frequent errors, performing comparably to the Pro version on simpler tasks without media queries. However, when media queries were required, the Flash model's performance dropped noticeably compared to the Pro version, which handled responsive design with greater precision.

3.1.3 Theory Question Results

This section describes the experiments conducted on answering open-ended theory questions using the Gemini models.

Objectives

The primary objectives of the theory question tests are as follows:

- Generate correct, complete, and accurate responses to theoretical questions using the **Gemini 1.5 Flash** and **Gemini 1.5 Pro** models.
- Evaluate each response on a scale of 0 to 5.

Prompt

The following prompt was used for the open-ended question.

System Prompt for Theory Exercises (translated from Italian)

You are a well-prepared student in standard and accessible web development, and I ask you to generate the answer to the theory question that has been assigned to you for the written exam of Web Technologies at the University of Bologna, Cesena campus, Degree Program in Computer Engineering and Computer Science.

Question:

```
{web_related_question}
```

Results

The average scores assigned to the answers generated by Gemini models are presented below.

Model	Average Theory Score	Solutions Evaluated
Gemini 1.5 Flash	5/5	7/10
Gemini 1.5 Pro	5/5	7/10
GPT-3.5 Turbo	4.5/5	10/10
GPT-4o	4.9/5	10/10

Table 3.5: Theory Question Answering Performance

The results presented in Table 3.5 confirm the exceptional performance of Gemini models in answering open-ended theory questions, mirroring the high accuracy observed in the previous experiment with GPT-3.5 Turbo and GPT-4o. Both Gemini 1.5 Flash and Gemini 1.5 Pro achieved perfect scores across all evaluated solutions.

3.1.4 JavaScript Results

This section describes the experiments conducted on generating JavaScript code using the Gemini models.

Objectives

The primary objectives of the JavaScript generation tests are as follows:

- Generate correct, complete, and precise JavaScript code to implement dynamic behavior on a specified HTML5 page using the **Gemini 1.5 Flash** and **Gemini 1.5 Pro** models.
- Evaluate each output on a scale of 0 to 7, focusing on the accuracy and functionality of code generated to handle events and manipulate the DOM.

Prompt

The following prompt was used for exercise JavaScript. The prompt includes the content of the file *esercizio_javascript.html.*

System Prompt for JS Exercises (translated from Italian)

You are a very well-prepared student in the development of accessible Javascript and I ask you to generate the solution for this Javascript exercise that was assigned to you for the written exam of Web Technologies at the University of Bologna, Cesena campus, Degree Course in Engineering and Computer Science. Below you will find the text of the Javascript question and, subsequently, the content of the esercizio_javascript.html exercise file that is requested.

HTML Content:

```
{html_content}
```

Results

The average scores for the JavaScript code generated are presented below.

Model	Average JavaScript Score	Solutions Evaluated
Gemini 1.5 Flash	4.8/7	7/10
Gemini 1.5 Pro	5.2/7	7/10
GPT-3.5 Turbo	5.6/7	10/10
GPT-4o	6.3/7	10/10

Table 3.6: JavaScript Code Generation Performance

The following table shows the types of errors observed in the generated JavaScript code and the number of solutions containing each error type.

Type of error in JavaScript	Gemini 1.5 Flash	Gemini 1.5 Pro	GPT-3.5 Turbo	GPT-4o
Inaccessible dynamically generated HTML	4/7	3/7	3/10	3/10
Other HTML accessibility errors generated by JS	0/7	0/7	1/10	1/10
Errors in data structures and HTML elements	3/7	1/7	3/10	0/10

Table 3.7: JavaScript Solutions with Errors by Gemini and GPT Models

The results for JavaScript generation were disappointing, with Gemini models showing less accuracy than the GPT models. Although Gemini often produced code that appeared correct at first glance, it frequently failed to function as intended. This was due to several issues, including:

- Attempting to interact with HTML elements before they were fully loaded.

- Syntax errors or other code-level problems that prevented the script from executing.

- Providing comments about what the code should accomplish rather than a fully implemented solution.

While the Pro version performed somewhat better than the Flash version, generating fewer code errors, it still fell short of the accuracy demonstrated by GPT models.

3.1.5 PHP Results

This section details the objectives, prompts, and results of the experiments conducted on PHP code generation using the Gemini models.

Objectives

The primary objectives of the PHP generation tests are as follows:

- Generate accurate PHP code to handle tasks such as database schema design, data insertion, and function implementation using the **Gemini 1.5 Flash** and **Gemini 1.5 Pro** models.
- Evaluate each output on a scale of 0 to 7, focusing on code functionality, accuracy, and adherence to PHP development best practices.

Prompt

The prompt for generating PHP scripts includes a supplementary text file, "*README_DB.txt*", which provides essential instructions for creating the database required for the PHP code to function correctly.

System Prompt for PHP Exercises (translated from Italian)

You are a highly skilled student in developing accessible PHP. Please generate the solution for this exercise assigned in the Web Technologies written exam at the University of Bologna, Cesena campus, for the Degree Course in Engineering and Computer Science. Below is the question and the contents of the *README_DB.txt* file required.

FILE:

```
{readme_db_content}
```

Results

The table below shows the average scores assigned to PHP code generated by each model, along with the number of tasks evaluated.

Model	Average PHP Score	Solutions Evaluated
Gemini 1.5 Flash	5.8/7	7/10
Gemini 1.5 Pro	6.3/7	7/10
GPT-3.5 Turbo	5.0/7	10/10
GPT-4o	6.2/7	10/10

Table 3.8: PHP Code Generation Performance

The following table categorizes common types of errors found in the generated PHP code for each Gemini model.

Type of error in PHP	Gemini 1.5 Flash	Gemini 1.5 Pro	GPT-3.5 Turbo	GPT-4o
Errors in Function Logic	3/7	2/7	4/10	1/10
Non-Standard W3C Code	3/7	1/7	1/10	3/10

Table 3.9: PHP Solutions with Errors by Gemini and GPT Models

The PHP solutions generated by both Gemini models were generally of high quality, with the Pro model showing slightly better precision and fewer errors. Both models handled database interactions and logical operations well, though they occasionally missed minor details, such as using incorrect database names.

Compared to the GPT models, the Gemini models showed very similar performance: Gemini 1.5 Flash outperformed GPT-3.5 Turbo, while Gemini 1.5 Pro's performance was comparable to GPT-4o. Both Gemini models exhibited nearly identical error types, making their overall performance closely aligned.

3.1.6 Accessibility Review

This section evaluates the generated code's accessibility, focusing on how well it meets established guidelines.

Objectives

The primary objectives of the accessibility evaluation tests is to evaluate the generated code's ability to identify and address accessibility issues, using the **Gemini 1.5 Flash** and **Gemini 1.5 Pro** models.

Prompt

The prompt used for accessibility review.

System Prompt for Accessibility Review

You are a detailed web accessibility validation tool. Your task is to analyze the provided HTML code and return a report with all accessibility errors based on WCAG 2.2 (including WCAG 2.0 and 2.1) AA level guidelines. You should generate a summary in CSV format with semicolons (;) as separators. The report should have the following columns:

- **File:** Name of the main file examined (the one written before the code).
- **Success Criterion:** The violated success criterion with a brief description.
- **Technique:** The associated reference technique.
- **Code Snippet:** The code segment with the detected error, enclosed in backticks.
- **Error Description:** A detailed description of the error or the reference warning.

Here is the page code to analyze:

`{HTMLPAGINA}`

Results

The table below shows sample results for accessibility errors detected in a PHP file using Gemini 1.5 Pro.

Violation	**Technique**	**Code**	**Description**
1.1.1 Non-text Content	G94	`<img aria-hidden="true"... alt="Facebook icon">`	Facebook's icon is used as an informative element. Add text describing the informative image.
1.3.1 Info and Relationships	H49	`<nav id="breadcrumb" aria-label="breadcrumbs">...`	The `<nav>` landmark should be used for main navigation. For breadcrumbs, use `<ol>`, `<ul>`, or `<nav aria-label="breadcrumb">` with an inner list.
1.4.1 Use of Color	G14	`<button id="menuLink" ... aria-hidden="true" ...>`	Color is the only indicator for menu status. Add a textual status update.
2.4.4 Link Purpose (In Context)	H77	`<ul><li>Home</li>...`	Home link inside `<nav>` should contain a correct link and be wrapped with an `<a>` tag.
2.4.4 Link Purpose (In Context)	H77	`<a href="ẗarget=¨_blank¨...>Read more...</a>`	Empty links (`href=""`) may confuse users. Use a valid link or the `onclick` event for more details.
4.1.2 Name, Role, Value	H91	`<button class="goUp hidden"...>`	The up arrow inside the button may be non-semantic for assistive tech. Consider replacing it or making it decorative.

Table 3.10: Accessibility Review Result

Both Gemini 1.5 Pro and Gemini 1.5 Flash produced great results in their accessibility reviews, uncovering issues that tools like AChecker [37] often ignore.

In assessing the output from both Gemini 1.5 Pro and Gemini 1.5 Flash, each model showed unique strengths and limitations:

Gemini 1.5 Pro generated a well-organized report (see Table 3.1.6), identifying a range of accessibility issues. However, its analysis often focused on general accessibility improvements without delving into critical, nuanced problems. Some major issues, which could significantly impact user experience, went undetected or received superficial attention.

Gemini 1.5 Flash performed similarly, capturing minor errors such as missing `alt` attributes for images and incomplete link text. However, its recommendations largely addressed surface-level concerns rather than deeper accessibility requirements, often missing structural or functional barriers that affect overall usability.

3.1.7 Code Correction Results

This section describes the experiments focused on evaluating the models' ability to correct provided PHP code.

Objectives

This experiment aims to assess the models' capabilities in identifying and rectifying errors within existing PHP code snippets. The code correction tasks involve debugging and improving the provided code to ensure functionality and adherence to best practices.

Prompt

The prompt used for the code correction task.

System Prompt for Code Correction (translated from Italian)

You are a university professor of Web Technologies specializing in accessibility. Given a web development assignment, which also includes a theory question, consisting of 5 questions, evaluate the assignment by assigning a final score from 0 to 32 according to the following instructions. In your answer, I ask you to report the errors made for each exercise and the score you assigned to it. Finally, report the final score n in the last line of the answer. The test is given to third-year students of the Bachelor's Degree Course in Engineering and Computer Science. Some precautions:

- In exercise 3, the answer to the theory question can also be synthetic and, if not explicitly requested, do not provide examples.
- Consider the use of longdesc valid even if deprecated in HTML 5 standard. The final grade, since it is used to create a summary table, report it exactly in the following format, where n is the value of the grade:

```
Grade: n

{EXERCISE 1}

{EXERCISE 2}

{EXERCISE 3}

{EXERCISE 4}

{EXERCISE 5}

{CRITERIA}

{STUDENT SOLUTIONS}
```

Results

The table below shows an example of the output from a corrected exam using Gemini 1.5 Pro.

Area	**Score**	**Result**	**Comments**
HTML	5/7	Minor errors	Missing ‘`<html lang="it">`‘. Image missing ‘width‘ and ‘height‘...
CSS	5/6	Minor errors	Missing sans-serif fallback font. No explicit ‘h3‘ alignment...
Theory	3/5	Major errors	Overly verbose. Repeats points. Unnecessary conclusion...
Javascript	4/7	Major errors	No check for empty fields. Incorrect color checking. Duplicate list items...
PHP	3/7	Major errors	Missing code for update functionality. Missing HTML structure...
Total Grade	**20**		

Table 3.11: Grading Results

The Gemini models do a decent job of identifying problems, though they still face some accuracy challenges. At times, errors may be interpreted as correct, even if they can't actually execute, and the models occasionally flag common, correct details as mistakes.

Overall, the models are effective, and their grading approach generally aligns well with the context. However, Gemini tends to be less strict than the GPT models, often assigning higher grades, whereas GPT models tend to grade more conservatively, giving slightly lower scores.

3.2 Code Generation with Optimized Prompts

This section analyzes the effects of prompt optimization techniques on the performance of Gemini models. The focus is on evaluating how meta-prompting, structured prompts, and the ”Act like an expert” strategy contribute to improved code generation and evaluation.

The following subsections delve into specific aspects of code generation and correction, examining the impact of optimized prompts on each:

- **HTML, CSS, JavaScript, and PHP Code Generation:** These subsections evaluate the quality of generated code in each language, comparing results before and after prompt optimization.

- **Code Correction:** This subsection focuses on how optimized prompts affect the models' ability to identify errors, provide feedback, and assign grades to code.

3.2.1 HTML Code Generation with Optimized Prompts

This section evaluates how prompt optimization affects HTML code generation, comparing the quality of code generated with optimized prompts to that produced using the original prompts.

Model	Before Optimization	After Optimization
Gemini 1.5 Flash	5.7/7	6.4/7
Gemini 1.5 Pro	5.9/7	6.7/7

Table 3.12: HTML Code Generation Performance Before and After Prompt Optimization

Analysis: While the HTML code generated after prompt optimization showed slight improvements, the quality of code was generally consistent with that produced by the original prompts. This outcome can be attributed to the straightforward nature of the HTML exercises, which limited the potential for major enhancements. However, the optimized prompts led to a marked improvement in accessibility, reducing the number of accessibility-related errors.

3.2.2 CSS Code Generation with Optimized Prompts

This section examines the impact of prompt optimization on CSS code generation, comparing the quality of code produced with optimized prompts to that generated using the original prompts.

Model	Before Optimization	After Optimization
Gemini 1.5 Flash	4.7/6	5.2/6
Gemini 1.5 Pro	5.3/6	5.5/6

Table 3.13: CSS Code Generation Performance Before and After Prompt Optimization

Analysis: While prompt optimization led to slight improvements in the generated CSS, the overall impact was moderate. Key issues, such as the missing blur effects and inconsistencies in hover states, persisted even after prompt adjustments. These areas, particularly the blur effect, remained unaddressed across multiple exercises, indicating that prompt optimization alone may not fully resolve these aspects. However, there was a small improvement in the handling of media queries. The optimized prompts helped both the Flash and Pro versions generate more consistent and accurate media query implementations, reducing errors that had occurred in the original versions.

3.2.3 JavaScript Code Generation with Optimized Prompts

This section evaluates the impact of prompt optimization on JavaScript code generation, comparing the quality of code produced with optimized prompts to that generated using the original prompts.

Model	Before Optimization	After Optimization
Gemini 1.5 Flash	4.8/7	6.0/7
Gemini 1.5 Pro	5.2/7	6.4/7

Table 3.14: JavaScript Code Generation Performance Before and After Prompt Optimization

Analysis: Prompt optimization significantly improved the quality of JavaScript code generation. The optimized prompts led to more functional and efficient code, with each script performing as expected in terms of core functionality. Execution was consistent, with all code running without major issues. While the code generated after optimization still occasionally contained minor issues, such as logical errors or non-functional POST requests, the overall improvement is evident.

3.2.4 PHP Code Generation with Optimized Prompts

This section evaluates the impact of prompt optimization on PHP code generation, comparing the quality of code produced with optimized prompts to that generated using the original prompts.

Model	Before Optimization	After Optimization
Gemini 1.5 Flash	5.8/7	6.4/7
Gemini 1.5 Pro	6.3/7	6.6/7

Table 3.15: PHP Code Generation Performance Before and After Prompt Optimization

Analysis: Prompt optimization led to general improvements in PHP code generation. The models now consistently generate complete, functional code that meets the requirements of each exercise. The optimized versions produced code that was more accurate and included fewer missing parts, resulting in a more reliable performance overall. However, accessibility issues within dynamically generated tables persisted, even when explicitly mentioned in the prompt.

3.2.5 Code Correction with Optimized Prompts

This section explores how optimizing prompts affects the code correction process, particularly focusing on the clarity of the Gemini models in explaining errors and suggesting improvements.

As illustrated in Table 3.16, the grading results reveal a mixed performance across various coding areas when utilizing the optimized prompts generated by the Gemini 1.5 Pro.

Area	**Score**	**Result**	**Comments**
HTML	7/7	No errors	The student correctly implemented the HTML table...
CSS	6/6	No errors	The student correctly implemented the CSS...
Theory Question	5/5	No errors	The student correctly and completely describes...
JavaScript	6/7	No errors	The student correctly implemented the JavaScript code...
PHP	2/7	Minor errors	Potential issues with filters applied multiple times...
			-3 for not handling sessions correctly...
			-1 for not sanitizing user input...
Total Grade	**26**		

Table 3.16: Optimized Grading Results

Analysis: The findings indicate that, contrary to expectations, the optimized prompts did not enhance the quality of the corrections. Despite providing more detailed instructions, the Gemini models often failed to recognize certain errors. As a result, the feedback was less accurate, and the grades assigned were significantly higher than expected, as shown in the table above.

Conclusion

The experiment highlights the transformative role that Large Language Models (LLMs) are shaping the future of web development, focusing on a comparison between OpenAI's GPT models and Google DeepMind's Gemini models. Both models were evaluated on their ability to generate accurate, accessible code in key web development languages like HTML, CSS, JavaScript, and PHP. The findings reveal that carefully crafted prompts significantly enhance code quality and functionality.

In particular, the Gemini models, especially the Gemini 1.5 Pro, stood out in generating functional and accessible code across a range of web development tasks. Optimized prompts had a notable impact on code accuracy and reduced common errors, especially with languages like PHP and JavaScript. Compared to GPT models, the Gemini models showed a slight edge in tasks requiring specific accessibility features, signaling their potential for accessibility-focused web development.

When optimized prompts were used, Gemini models showed strong adherence to Web Content Accessibility Guidelines (WCAG), though complex accessibility requirements still presented occasional challenges. This prompt optimization proved especially effective in HTML and CSS tasks, where elements like alt text and ARIA labels were more reliably integrated. Although both models encountered difficulties in context-heavy or interactive JavaScript tasks, Gemini models demonstrated slightly more consistent performance in these complex requirements.

Overall, prompt optimization had a more pronounced effect on the Gemini models than on GPT models, resulting in more complete code with fewer errors on the first attempt. This highlights the value of well-structured, detailed prompts in generating high-quality responses. For instance, with optimized prompts, the Gemini 1.5 Pro produced fewer syntax errors and more accurate outputs, excelling in context-aware code generation and outperforming both GPT-3.5 Turbo and GPT-4 in these areas.

In general, Gemini models, particularly the Pro version, led in generating accessible, detailed code with fewer errors when using optimized prompts, making them especially effective for web development tasks that require precision and a focus on accessibility.

Despite their strengths, both models showed some inconsistency with highly context-sensitive tasks, such as managing JavaScript state or implementing dynamic interactions. Recurring challenges in responsive design and interactivity also emerged, as the models struggled to fully align with best practices in web development. While accessibility issues were reduced, nuanced requirements like color contrast and screen reader compatibility still posed challenges. These limitations suggest a need for further model refinement and more targeted prompt designs to fully address these details.

Given Gemini's strong performance in generating accessible code, these models could be a promising fit for automated code review systems focused on accessibility. LLMs could scan code continuously for accessibility compliance, offering suggestions and helping developers create inclusive digital experiences with less manual effort. They could also be valuable in web development education, providing real-time feedback and troubleshooting to create a more interactive learning environment. Students could get prompt-based guidance to build a better understanding of web development concepts and standards.

Both Gemini and GPT models have proven to be powerful tools for web development, but their full potential shines when optimized for specific uses. When users give clear, detailed prompts that align with their needs, these LLMs become even more effective and efficient. A deep understanding of what users need plays a key role in guiding these models to deliver the best results. With well-structured prompts, these LLMs can evolve into even more specialized and valuable resources for web development.

Future directions for this experiment could include expanding the analysis to incorporate other language models, such as Anthropic's Claude [38], to gain a broader perspective on how different models handle web development tasks. Comparing these models could help identify which is best suited to specific web development needs, particularly for tasks requiring precision and accessibility.

Additionally, since the experiment already included LLMs self-correcting their own code, another valuable step could be to compare the generated and corrected code with actual student submissions or to expand the tasks used to evaluate these models by including more complex web development scenarios.

Bibliography

[1] World Wide Web Consortium (W3C). Web content accessibility guidelines (wcag) 2.1. `https://www.w3.org/TR/WCAG21/`, 2018.

[2] OpenAI. Gpt-3. `https://openai.com/index/gpt-3-apps/`, 2020.

[3] OpenAI. Gpt-4. `https://openai.com/index/gpt-4/`, 2023.

[4] Google AI. Gemini. `https://gemini.google.com/app`, 2023.

[5] Vlado Keselj. Book review: Speech and language processing by daniel jurafsky and james h. martin. *Computational Linguistics*, 35(3), 2009.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3 (null):1137–1155, March 2003. ISSN 1532-4435.

[7] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[8] Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic programming and evolvable machines*, 19(1):305–307, 2018.

[9] Alec Radford. Improving language understanding by generative pretraining. 2018.

[10] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805, 2019.

[12] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[13] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[14] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[15] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[16] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[17] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[18] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina

Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[19] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24 (240):1–113, 2023.

[20] Jiacheng Xu, Zhe Gan, Yu Cheng, and Jingjing Liu. Discourse-aware neural extractive text summarization. *arXiv preprint arXiv:1910.14142*, 2019.

[21] Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. Pretrained transformers for text ranking: Bert and beyond. In *Proceedings of the 14th ACM International Conference on web search and data mining*, pages 1154–1156, 2021.

[22] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text, 2019. URL `https://arxiv.org/abs/1903.10676`.

[23] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[24] Ben Shneiderman. *Human-centered AI*. Oxford University Press, 2022.

[25] Tim Berners-Lee. The first website. `https://www.w3.org/History.html`, 1991.

[26] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas

Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[27] Demis Hassabis Jeff Dean and James Manyika. 2023: A year of groundbreaking advances in ai and computing. Technical report, Google, 2023.

[28] Danna Gurari, Yinan Zhao, Meng Zhang, and Nilavra Bhattacharya. Captioning images taken by people who are blind. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVII 16*, pages 417–434. Springer, 2020.

[29] Barry Bassi. *Utilizzo di Large Language Model nello sviluppo web e nella valutazione dell'accessibilita: un approccio sperimentale.* PhD thesis. URL http://amslaurea.unibo.it/32244/.

[30] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 1998. doi: 10.1142/S0218488598000094. URL https://doi.org/10.1142/S0218488598000094.

[31] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[32] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[33] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[34] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[35] Mirac Suzgun and Adam Tauman Kalai. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024.

[36] Yinheng Li. A practical survey on zero-shot prompt design for in-context learning. *arXiv preprint arXiv:2309.13205*, 2023.

[37] Achecker web accessibility checker. `http://achecker.csr.unibo.it/checker/index.php`, 2024.

[38] Anthropic. Claude. `https://www.anthropic.com/claude`, 2023.

Acknowledgments

I would like to express my sincere gratitude to all those who supported me throughout my research journey.

Thank you all.