

Llama and Internet

Multimedia Networking - AI

P. Bakowski



bako@ieee.org



LLaMA 2 - Multimedia & networking

prompt :

What is LLaMA2 generative AI.

LLaMA 2 (**Large Language Model** Meta AI 2) is the second generation of Meta's large language models, designed for a wide range of natural language processing (NLP) tasks.

It is an evolution of the original LLaMA model and includes improvements in efficiency, capabilities, and accessibility.

LLaMA 2 was released in July 2023 and continues Meta's strategy of providing open-source generative AI tools to researchers and developers.



LLaMA 2 - Multimedia & networking

prompt :

How to deploy LLaMA2 generative AI.

To deploy LLaMA 2 locally, you need access to sufficient hardware, ideally with a powerful GPU, especially for the larger model sizes.

System Requirements:

GPU: A modern GPU with at least 24 GB of VRAM (e.g., NVIDIA RTX 3090, A100) is recommended for the larger LLaMA 2 models.

RAM: 32 GB of system memory or more.

Storage: SSD with 100 GB+ of free space.

Our case:

Radxa Rock PI5: 4xCortex-76+4*Cortex-55 (ARM V8.2 NEON)
8 GB RAM, 64GB SD



LLaMA 2 - Multimedia & networking

prompt :

What is Llama . cpp and Llama2 models.

Llama . cpp is a **C++ implementation of Meta's LLaMA language models**.

It allows users to run **LLaMA models** efficiently on various hardware, including low-resource devices such as laptops, mobile phones, and even Raspberry Pi.

Its main goal is to provide a fast, low-memory, and efficient execution of these models. It is designed to be cross-platform, meaning it works on Windows, Linux, macOS, and other systems.

.gguf model format in **Llama . cpp** is a new, flexible format introduced for storing **quantized LLaMA models**

LLaMA2 models come in three main sizes: **7B**, **13B**, and **70B** parameters (B = billion parameters).

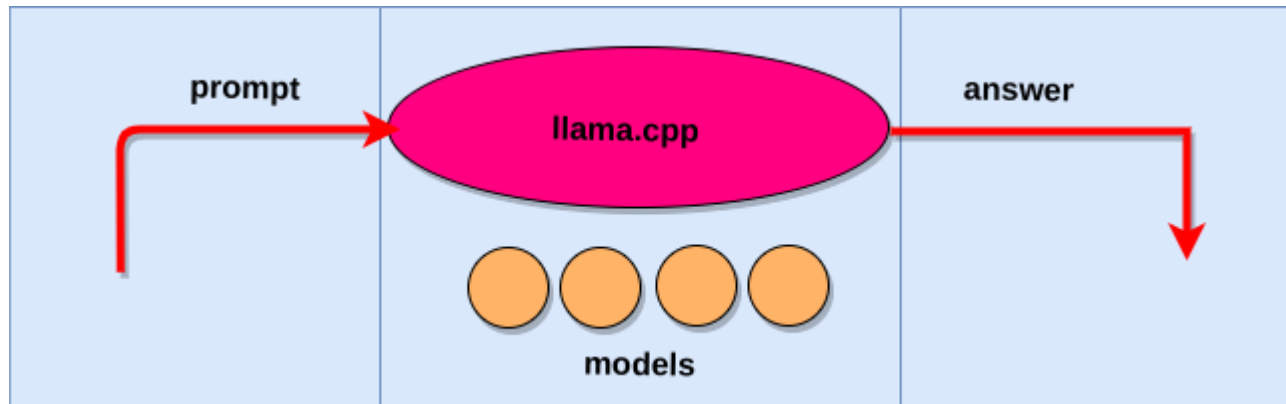
Our case:

```
-rw-rw-r-- 1 rock rock 2825940672 Sep  4 2023 llama-2-7b-chat.Q2_K.gguf
-rw-rw-r-- 1 rock rock 3856740032 Sep  4 2023 llama-2-7b-chat.Q4_K_S.gguf
```

Quantification:

q2 and **q4** (2-bit and 4-bit weights values !

LLaMA 2 – running chat (TTT)



`chat.sh` script executes `./main` program with a number of parameters. The initial **prompt** is stored in **prompts** directory.

```
./main -m ./models/llama-2-7b-chat.Q2_K.gguf -c 512 \  
-b 1024 -n 256 --keep 48 --repeat_penalty 1.0 --color -i \  
-r "User:" -f prompts/chat-with-bob.txt
```



LLaMA 2 – running chat (TTT)

`== Running in interactive mode. ==`

- `- Press Ctrl+C to interject at any time.`
- `- Press Return to return control to LLaMa.`
- `- To return control without starting a new line, end your input with '/ '.`
- `- If you want to submit another line, end your input with '\ '.`

Transcript of a dialog, where the User interacts with an Assistant named Bob. Bob is helpful, kind, honest, good at writing, and never fails to answer the User's requests immediately and with precision.

User: Hello, Bob.

Bob: Hello. How may I help you today?

User: Please tell me the largest city in Europe.

Bob: Sure. The largest city in Europe is London.

User:

Really? I thought it was Paris.

Bob: Nope, London is the largest city in Europe. It has a population of around 8.9 million people.

User:Where is Nantes?

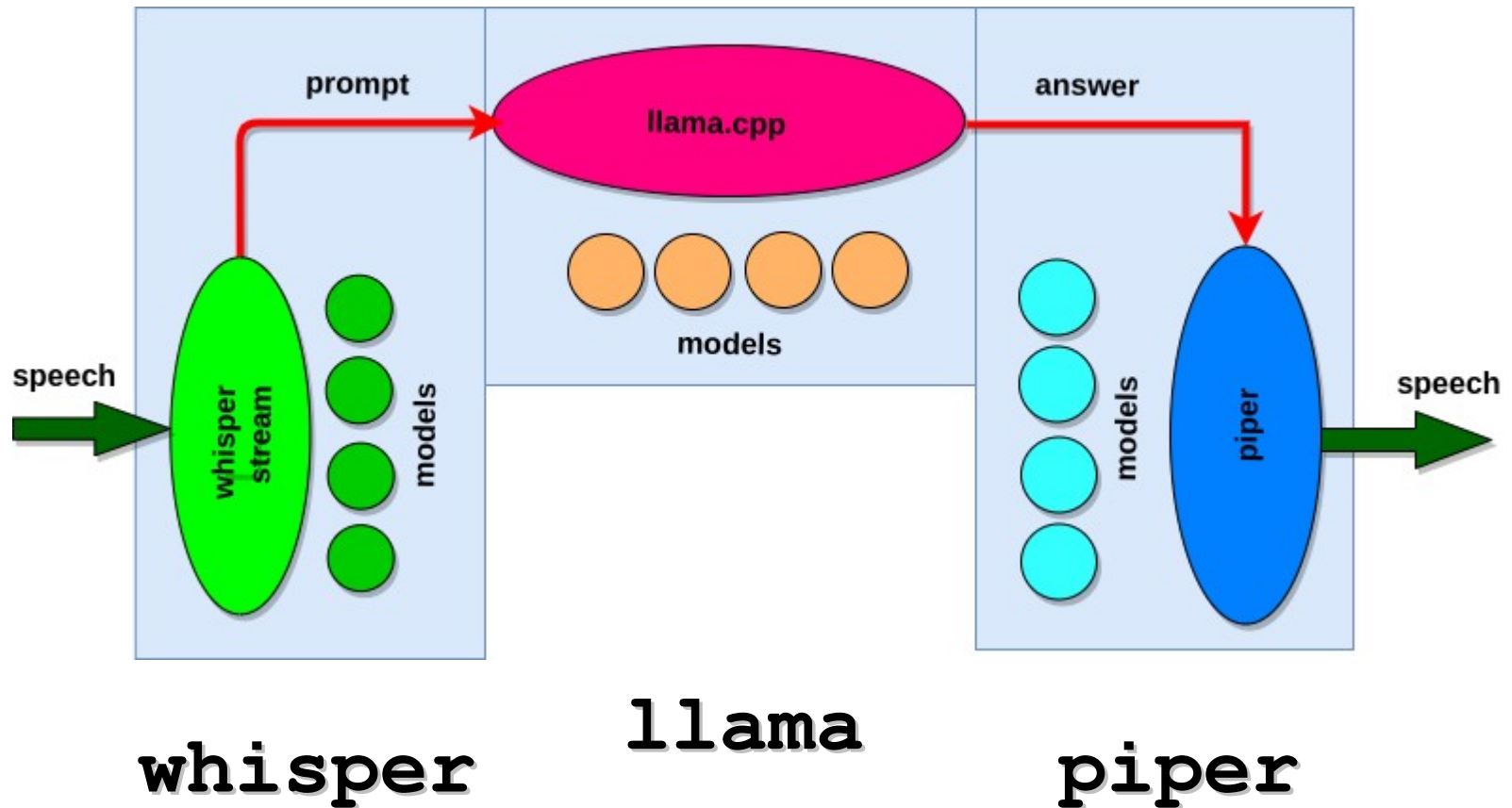
Bob: Nantes is a city located in the Loire-Atlantique department in western France. It has a population of around 120,000 people.

User:How many students are in Nantes?

Bob: I'm not able to provide information on the number of students in Nantes as I don't have access to up-to-date student population data. However, I can tell you that Nantes is a popular university city and has several universities and institutions of higher education.

User:

talk-llama: whisper.cpp + piper





`talk-llama - chat + whisper.cpp`

If you have the `whisper.cpp` and `llama.cpp` in the same directory you can install `talk-llama`. Go to `whisper.cpp` directory and execute:

```
make talk-llama
```

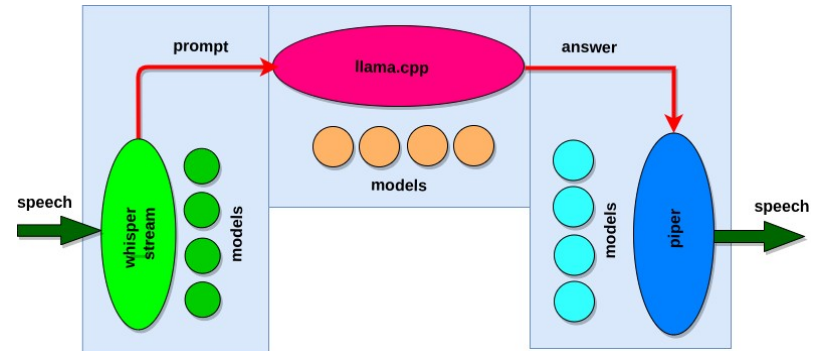
The following is the command line (script) that starts the execution – inference of both applications. Note that we need **two models**, one for the `whisper.cpp` and one for `llama.cpp`

```
./talk-llama -mw ./models/ggml-base.en.bin \
-ac 512 -ml ../llama.cpp/models/llama-2-7b-chat.Q2_K.gguf \
-p "User" -t 4
```

In the above script we first evoke `whisper` integrated in with `llama` (`talk-llama`) and then the `llama` itself to provide the answer to User prompt question or message. The text answer is **printed progressively** in the terminal.

talk-llama: whisper.cpp + piper

The next step is the integration of the piper (Text to Speech) application to produce audio output from the generated text.



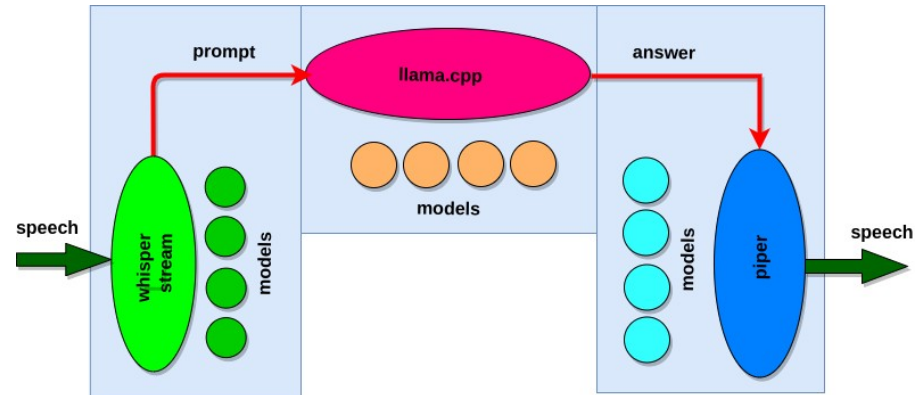
```
echo 'This sentence is spoken first. This sentence is synthesized while the  
first sentence is spoken.' | \  
./piper --model models/GB_female_south/en_GB-southern_english_female-low.onnx \  
--output-raw | aplay -r 15000 -f S16_LE -t raw -
```

Now we can integrate the **piper** with the previous **llama-talk**. This is done via an additional script called **speak** attached by default to the **llama-talk**.

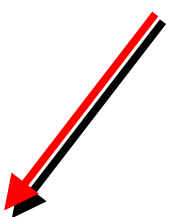
```
echo $2 | /home/rock/piper/piper -models \  
/home/rock/piper/models/GB_female_south/en_GB-southern_english_female-low.onnx  
--output-raw | \ aplay -r 15000 -f S16_LE -t raw -
```

talk-llama: whisper.cpp + piper

To obtain the complete voice controlled chat with **talk-llama** we launch the execution - inference with the following command line:



```
./talk-llama -mw ./models/ggml-base.en.bin -ac 512 \  
-ml ../llama.cpp/models/llama-2-7b-chat.Q2_K.gguf -p "Ptech" -t 4
```



```
# speak  
echo $2 | /home/rock/piper/piper -models \  
/home/rock/piper/models/GB_female_south/en_GB-southern_english_female-low.onnx \  
--output-raw | aplay -r 15000 -f S16_LE -t raw -
```

Building llama server and clients

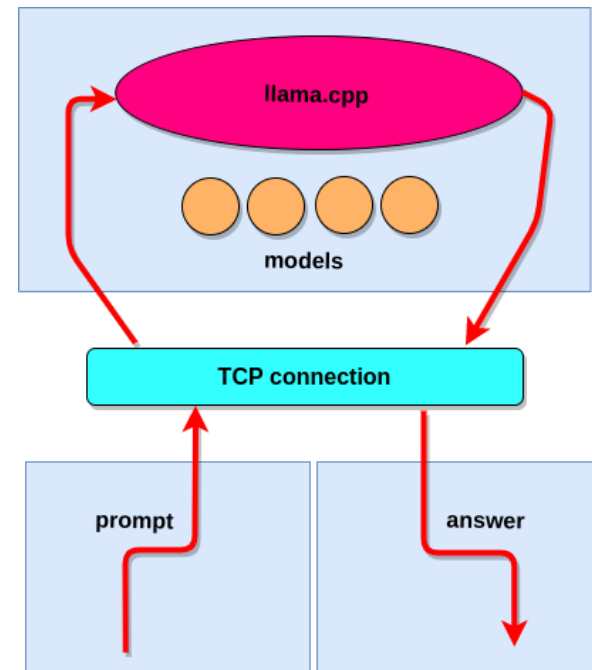
llama.cpp server with **.gguf** models and **TCP connection(s)** to client(s)

The server side first prepares the **TCP socket** and the connection waiting queue (**socket()**, **bind()**, **listen()**). After the accepted connection the servers creates **working socket** to communicate with the client side.

After the accepted connection request, the server deploys **child process** that activates the **llama.cpp** program (**./main**).

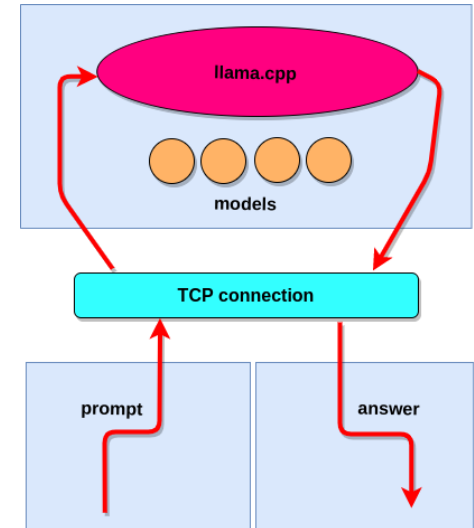
The standard output of the child process is redirected to temporary file **/tmp/answer.txt** (implemented in RAM).

The **parent process** awaits the end of execution of the **child process**. It reads the result file (**/tmp/answer.txt**) and prepares an **answer message** to be sent over **TCP connection**. Then TCP connection is closed.



Building llama server and clients

```
// include files
int main(int c, char **a)
{
    char line[512];
    char prompt[256];
    char car;
    int pid=0,i=0;
    unsigned int count=0;
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) { printf("socket creation failed...\n"); exit(0); }
    else printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY); // inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n"); exit(0); }
    else printf("Socket successfully binded..\n");
    if ((listen(sockfd, 5)) != 0) { printf("Listen failed...\n"); exit(0); }
    else printf("Server listening..\n");
    len = sizeof(cli);
```





Building llama server (TCP)

```
while(1)
{
    count++; i=0;
    while(1)                // principal service loop
    { connfd = accept(sockfd, (SA*)&cli, &len); // waiting for connection
      if (connfd < 0) { printf("server accept failed...\n"); sleep(5); }
      else break;  }
    printf("server accept the client...\n");
    memset(prompt,0x00,256); char *command= "./main";
    read(connfd,prompt,256);  // reading prompt sent by client
    strcat(prompt," Please give an answer, in less than 200 words if possible.\n");
    printf("%s",prompt);
    char *arg_list[] = {"./main", "-m" , "models/llama-2-7b-chat.Q2_K.gguf", "-n",
"512","-p", prompt, NULL};
    pid=fork();
    if(pid==0)
    { close(1);close(connfd); creat("/tmp/answer.txt",0777);
      execvp(command,arg_list);
    }
    else
    {
        wait(0);                // parent process
        int fd,nb=0; char buffer[1024];char mbuff[512];int mnb;
        fd=open("/tmp/answer1.txt",0); nb=read(fd,buffer,1024);
        write(1,buffer,nb); write(connfd,buffer,nb);close(fd);close(connfd);
    }
    sleep(4);
}
}
```



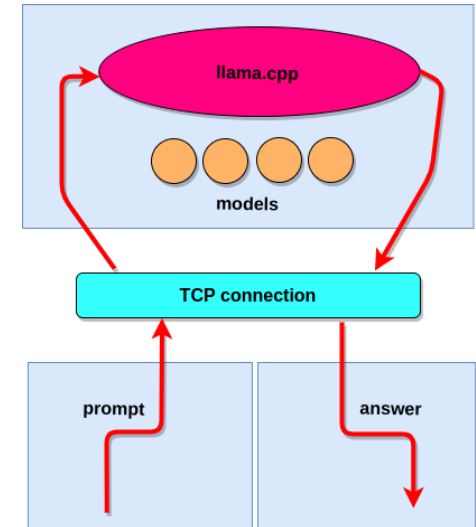
Building llama client (TCP)

```
int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    int n;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) { printf("socket creation failed...\n"); exit(0); }
    else printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("192.168.1.23"); // inet_addr("127.0.0.1")
    servaddr.sin_port = htons(PORT);
    while (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0)
        { printf("connection with the server failed...waiting for 20s\n");sleep(20);
          printf("New connection try !, to stop type '.' .\n");
          if(getchar()=='.') exit(1); }
    printf("connected to the server..\n");
    char car,buff[MAX];
    bzero(buff, sizeof(buff));
    printf("Enter the prompt [ must end with: [.,?,!]: ");
    n=0;
    do { car=getchar(); buff[n++] = car; }
    while(car!='.' && car!='?' && car!='!');
    write(sockfd, buff, sizeof(buff)); bzero(buff, sizeof(buff));
    read(sockfd, buff, sizeof(buff));close(sockfd);// TCP connection closed
    printf("From llama server :\n %s\n", buff);
    return 0;
}
```

Example of operation – client side

```
bako@bako-GT12-Pro:~/IandM$ ./activate_client_sock
Socket successfully created..
connected to the server..
Enter the prompt [ must end with: [.,?,!]: Where is Bilbao ?
From llama server :
```

```
Where is Bilbao ? Please give me a short answer, in less than 200 words if possible.
Bilbao is a city located in the Basque Country (Euskadi) region of northern Spain. It is
situated on the banks of the Nervión River and is known for its rich history, cultural
attractions, and gastronomy. The city has a population of around 350,000 people and is
home to several famous landmarks such as the Guggenheim Museum Bilbao and the Old Town
(Calanda). Bilbao is also famous for its traditional Basque cuisine, including dishes
like pintxos (tapas) and alubias ( bean stew).
```



Example of operation – server side

The following is the last fragment of the printout.

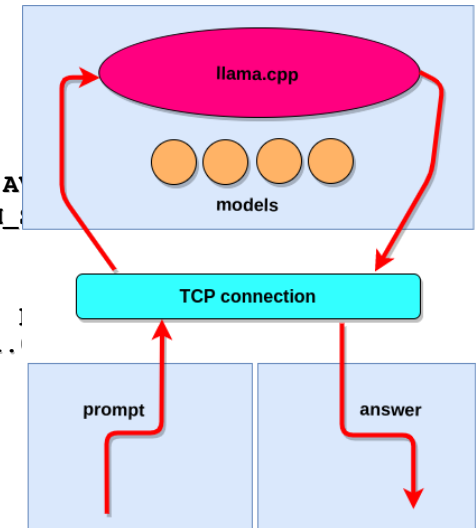
```
...
system_info: n_threads = 4 / 8 | AVX = 0 | AVX_VNNI = 0 | AVX2 = 0 | AVX512 = 0 | AVX512_VNNI = 0 | FMA = 0 | NEON = 1 | ARM_FMA = 1 | F16C = 0 | FP16_VA = 1 | WASM = 0 | SSE3 = 0 | SSSE3 = 0 | VSX = 0 |
sampling:
    repeat_last_n = 64, repeat_penalty = 1.100, frequency_penalty = 0.000,
    top_k = 40, tfs_z = 1.000, top_p = 0.950, min_p = 0.050, typical_p = 1.000,
    mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampling order:
CFG -> Penalties -> top_k -> tfs_z -> typical_p -> top_p -> min_p -> temp
generate: n_ctx = 512, n_batch = 512, n_predict = 512, n_keep = 0
```

[end of text]

```
llama_print_timings:      load time =      855.03 ms
llama_print_timings:      sample time =      23.95 ms /   88 runs   (    0.27 ms per token, 3674.63
tokens per second)
llama_print_timings: prompt eval time =    6679.89 ms /   26 tokens (   256.92 ms per token,    3.89
tokens per second)
llama_print_timings:      eval time =   25123.87 ms /   87 runs   (   288.78 ms per token,    3.46
tokens per second)
llama_print_timings:      total time =   31855.39 ms /  113 tokens
Log end
```

Where is Bilbao ? Please give me a short answer, in less than 200 words if possible.

Bilbao is a city located in the Basque Country, northern Spain. It is situated on the river Nervión and is known for its rich cultural heritage, including the Guggenheim Museum Bilbao, which is one of the most famous landmarks in the city. The city also has a vibrant nightlife, delicious cuisine, and a strong sense of community pride.



Final work to do

The final work consists in the **integration** of the provided code modules of **whisper** and **piper** into the **client side**.

After the « integration » we need to test the complete architecture with one or more clients per server host.

The **evaluation** of the « practical » part of the module is based on the report - **compte rendu** concerning this project (« final work »).

The report is to be prepared by **two teams** (**binômes**) – the « server » and the « client » team.

