ChatGPT

Researching a Winning Project Idea for the Gradio Agents & MCP Hackathon 2025

What Makes a Project Likely to Win

To maximize chances of winning, it's important to align with the hackathon's judging criteria and showcase innovation, technical skill, usability, and impact [1]. In this hackathon, judges are looking for projects that:

- **Innovate** – introduce a creative or original solution or use of AI agents/MCP tools.
- **Demonstrate Technical Excellence** – effectively implement AI models or protocols (MCP, Gradio, etc.) in a robust way.
- **Are Highly Usable** – have a clear, user-friendly interface or API; the project should be easy to use or integrate.
- **Have Real-World Impact** – solve a meaningful problem or show potential for significant benefit.
- **Engage the Community** – garner interest or "likes" from others (community feedback is a minor factor) [1].

Reviewing the **hackathon tracks** and current submissions gives insight into what areas are popular and how to stand out. The three tracks are: MCP Server/Tool, Custom Agent Components, and Agentic Demo apps [2] [3]. Many participants are building diverse applications – for example, a job application assistant (**HireMeAgent** for auto-generating resumes/cover letters), educational math tools, computer vision pose estimators, PII compliance checkers, an F1 racing strategy tool, and even a *Clinical Triage* medical assistant [4]. This variety means **no single domain is "mandatory"**, but to win you should either target a **high-impact domain** (like healthcare, education, or enterprise productivity) or present a uniquely creative demo that wows the judges. It also helps to leverage the strengths of Gradio+MCP (for example, giving an LLM new abilities via tools [5]).

Evaluating Your Project Idea vs Alternatives

Your current idea is in **Track 1 (MCP Tool/Server)**: *a medical-symptom MCP server that maps patient-entered symptoms to ICD-10 codes via a local knowledge base, returning a JSON of probable diagnoses with confidence scores.* Let's evaluate this idea in light of the criteria and compare it to other possible directions:

- **Medical Symptom → ICD-10 MCP Server (Proposed)** – This targets healthcare, a high-impact domain. It aims to solve a real problem: mapping symptoms to standardized diagnostic codes (ICD-10) and likely conditions, which can assist in triage or medical coding. Such a tool could save doctors/coders time and reduce errors [6]. It's innovative in that it would provide structured output (JSON with diagnoses and confidence) that an AI agent can use downstream (e.g., for automated medical documentation or decision support). A similar hackathon entry, *Clinical Triage MCP*, shows the viability and interest in this space – that project takes symptoms and returns a structured assessment of urgency, possible condition, and recommended action [7]. Your idea goes a step further into **coding** the condition (ICD-10) which adds value for billing/record-keeping [8] [9]. If well-executed, this could score high on impact and technical depth. Just be mindful to clearly state it’s not giving medical advice, only suggestions.

- **Network Automation Agent/Tool** – Given your network engineering background, another idea could be a network automation or troubleshooting agent (for Track 1 or 3). For example, an MCP server that takes high-level network requirements and outputs router configurations or a diagnostic agent that analyzes network logs to suggest fixes. This would leverage your **CCNP/JNCIP expertise** and could be innovative, as not many participants have networking domain knowledge. It addresses a real need in IT (automating network setups or debugging). However, consider feasibility: you’d need either simulated network data or devices to interact with. A possible MVP could generate configuration snippets or architecture diagrams from text input. While this could impress on technical merit, the **judges’ familiarity** with networking might be limited, so you’d have to ensure the project’s value is clearly explained and demonstrated visually (to score on usability/impact). It’s a viable alternative if you feel more comfortable here, but it might be niche compared to the broad appeal of a medical tool.

- **Creative AI Demo (e.g. Music or Juggling Coach)** – Track 3 allows any agent demo, so you could conceive something like a music composition assistant or a vision-based juggling analyzer (given your interests). These could be fun and novel (imagine an agent that composes melodies or gives juggling tips analyzing video frames). The creativity might score points for innovation. However, building such a demo solo in one week can be challenging if you need specialized models (music generation or computer vision) and a polished interface. It’s also less directly impactful than healthcare or enterprise automation, unless done exceptionally well. As a solo builder on a tight timeline, this might be riskier and require stepping outside your core expertise.

Verdict: Your **medical symptom → ICD-10 MCP server** idea appears to be a strong candidate for winning. It aligns with a **high-impact, real-world problem**, demonstrates a structured use of AI (which highlights technical skill), and can be made user-friendly as an API/tool. It’s also something you’re motivated by (you even have a PhD contact interested in it), which is important for execution. Furthermore, since another team is working on clinical triage, it validates the domain – but you can differentiate your project by focusing on **ICD-10 coding and confidence scoring**, giving it a unique edge.

Designing the Medical Symptom→ICD-10 MCP Server

To nail this project, let’s outline the Minimum Viable Product and how to build it:

Project Name (Tentative): *MedicodeMCP* – an MCP Tool for Symptom-to-ICD Diagnosis Mapping.

MVP Scope: The MVP will accept a patient's symptom description (free-text input) and output a structured JSON with a list of probable diagnoses, each with an ICD-10 code and a confidence score. We’ll keep the scope manageable – for example, handling a subset of common symptoms and returning a few likely diagnoses (maybe top 3-5). This is enough to demonstrate the concept without needing an exhaustive medical encyclopedia upfront.

How It Works:
1. **Input Interface:** A simple Gradio interface (for testing) with a text box for symptoms (e.g. *“chest pain and*

shortness of breath"). Since this is an MCP server, the primary interface is programmatic (the MCP client calls it), but a demo UI helps during development and for judges to try it out.
2. **Processing Logic:** The core engine will map the input to diagnoses. For the MVP, the *quickest path* is to leverage a **Large Language Model** with medical knowledge: - We can call an API like OpenAI's GPT-4 or Anthropic's Claude (using those free credits) to parse the symptoms and suggest diagnoses. For example, we'd prompt the LLM with something like: *"The patient reports: {symptoms}. Provide a JSON list of up to 5 possible diagnoses, each with an ICD-10 code and a confidence score between 0 and 1. Use official ICD-10 names and codes."*. The LLM's vast medical knowledge can associate symptoms with likely conditions. In fact, recent experiments with medical foundation models (like Google's Med-PaLM/MedGEMMA) show they can identify relevant diagnosis codes from text with reasonably good accuracy just by prompt-based reasoning [10]. Using GPT-4/Claude in the loop will dramatically speed up development and likely yield high-quality suggestions (ensuring the "Technical Implementation" is strong, thanks to state-of-the-art AI [11]). - **Confidence Scoring:** LLMs aren't inherently calibrated for confidence, but we can instruct the model to assign a subjective probability or confidence level for each diagnosis. This will mostly be an educated guess, but it adds useful structure. Alternatively (or additionally), we could rank diagnoses by the model's internal reasoning or use multiple prompts to double-check, but given time constraints, a single-pass answer with approximate confidences is acceptable for MVP. - **ICD-10 Code Mapping:** LLMs often know common ICD-10 codes (e.g., they might output chest pain as R07.9 or heart attack as I21.x). To be sure of correctness, we could post-process the LLM's output by verifying codes against a local ICD-10 database. For MVP, a simpler approach is to trust but verify a bit: maintain a small dictionary of common ICD-10 codes (or use regex pattern to ensure format) to sanity-check the output. If the model returns a code that doesn't match any known pattern or common code, we could flag or adjust it. (In future, one could integrate a full ICD-10 lookup list for validation.) - **Alternate approach:** If not using an API, there are open models fine-tuned for ICD coding. For instance, a published *Clinical BERT* model can take clinical text and predict the top ICD-10 codes [12]. One could load that model (from Hugging Face) and feed the symptom text to get a set of likely codes, then map those to diagnosis names. This requires more coding and possibly a GPU, but it's doable with provided resources. However, for the MVP in a hackathon, using a powerful API (GPT/Claude) is the fastest route to a working solution, and using those credits is encouraged [11]. 3. **Output Format:** The output will be a **JSON structure** that an agent can easily parse. For example:

```
{
  "diagnoses": [
    {
      "icd_code": "I20.0",
      "diagnosis": "Unstable angina",
      "confidence": 0.85
    },
    {
      "icd_code": "J18.9",
      "diagnosis": "Pneumonia, unspecified organism",
      "confidence": 0.60
    }
  ]
}
```

This is just an example for input "chest pain and shortness of breath" – the system might list cardiac-related issues (angina/MI) and a respiratory cause, with confidence estimates. The key is that it returns **structured data** (much like the Clinical Triage MCP example which returns JSON with fields like urgency_level, etc. [13] [14]). Structured output is ideal for MCP tools, since an agent can use those fields directly in its reasoning or present them to users.

1. **Gradio MCP Integration:** We will implement this logic in the `app.py` of a Gradio Space and ensure it's exposed as an MCP server. According to hackathon docs, any Gradio app can be an MCP tool if it's built correctly [5]. In practice, this means using the Gradio 5.x **Chatbot interface** or Blocks with specific annotations so that an external LLM agent can invoke the tool. We'll follow the guide on *"Building an MCP Server with Gradio"* for the exact setup. (Typically, you define the tool's function and metadata so an MCP client can discover it.) For example, the *Counting Letters* tutorial in Gradio's docs shows how to define a tool that an agent can call to solve a subtask [15]. Our tool will be similar: the function wraps the symptom-to-diagnosis logic. We should test it with an MCP client (Gradio provides a minimal MCP client example, or we can use something like the open-source **TinyAgents** client).

2. **User Demo (Client App):** To showcase the project to judges (and fulfill submission requirements), we should create a small MCP **client** or demo app that uses our server. For example, a simple Gradio interface where a user enters symptoms, and under the hood it calls the MCP server and displays the JSON result nicely. In the *Clinical Triage* project, they provided a separate demo space that connects to the MCP server [16] [17]. We can do similarly or record a short video. Hackathon guidelines specifically ask for a recorded demonstration of the MCP tool in action with an agent client [18]. So for submission, plan to either link a live client app or include a video where you use (say) Hugging Face's Claude-2 chatbot or another agent to call your tool. This will prove that your MCP server works as intended in an agentic context.

MVP Development Steps: (as a brief to-do list) 1. **Set up Gradio Space** – Initialize a new Hugging Face Space (Gradio SDK 5.x) for the MCP server. Add the hackathon tag `mcp-server-track` in the README as required [19] [20]. 2. **Implement Symptom-to-Diagnosis Function** – Write a Python function that takes a symptom text and returns the JSON output. Inside, start with a straightforward call to an API: - Use OpenAI's library or Anthropic's to query the model. Ensure to format the prompt for JSON output. Parse the model's response (which should be JSON) into a Python dict. Handle any JSON formatting quirks (sometimes the model might produce extra text; we can use regex or `json.loads` after trimming). - (If API usage runs into issues or limits, have a fallback: e.g., a limited rule-based approach or an offline model for at least a few demo cases.) 3. **Testing** – Try various inputs (common symptoms, combinations) to see if the output is sensible. Tweak the prompt as needed to improve relevance of diagnoses and correctness of codes. For instance, if the model gives very broad or nonspecific answers, guide it to be more specific. Also, verify the JSON structure is always valid. 4. **Confidence Calibration** – Decide how the confidence scores are determined. If using GPT, you might accept whatever self-reported confidences it gives. If that's unsatisfactory, you could rank by the order the model lists diagnoses (first = highest confidence). Document how confidence is defined (since it's an approximation in this context). 5. **Integrate with Gradio Blocks** – Wrap the function in a Gradio interface. Possibly use a `gr.Interface` or `gr.ChatInterface` with a custom tool definition. Ensure the app can receive an input string and output the JSON (or a pretty display of it). For the MCP aspect, we might need to define the `fn` and set the Gradio app's metadata so it advertises an API endpoint. (The Gradio docs mention that the MCP server essentially exposes the function for agent use). 6. **Build a Quick Client (optional but recommended)** – For demonstration, either build a minimal second Space or a local script that uses the `gradio.Client` or `requests` to call the Space's prediction API. This will act as the MCP client. If building a Space, it could be a simple chatbot interface that shows how an LLM (even a dummy agent) would use the tool. Otherwise, prepare a screen recording where you paste an example into an AI agent (like Claude Desktop with MCP configured) and it calls your tool to get the answer. 7. **Polish Documentation** – In the Space's README, clearly explain what the tool does, how to use it, and include the required elements: the track tag, and a link to the video or client demo [18]. Also mention the technologies used (e.g. "Uses OpenAI GPT-4 via API to map symptoms to diagnoses" or "built on ClinicalBERT model…"). Good documentation and presentation can set you apart [21].

By following these steps, the MVP will demonstrate end-to-end functionality: you enter symptoms, and get back structured diagnostic insights. This alone is a significant achievement for a week-long solo project, and it checks the boxes for innovation (applying MCP to a medical coding task), technical implementation (integrating LLM or fine-tuned model, JSON output, MCP protocol), usability (clear input/output and example client usage), and impact (healthcare application).

Utilizing Hackathon Resources Wisely

One concern you raised is how to use the many credits and tools sponsors provided. It's great to use them to **boost your project's performance**, but remember the goal is a compelling demo – you don't *have* to use every service. Quality over quantity. Here's a strategy to leverage the resources without getting overwhelmed:

- **OpenAI / Anthropic Credits** – These are likely your MVP's power source. As noted, using GPT-4 or Claude-2 can greatly improve your tool's accuracy and save development time [11]. For instance, with ~$25 of OpenAI credits, you can make a few hundred GPT-3.5 calls or a smaller number of GPT-4 calls – plenty for prototyping and demo queries. Use these to implement the core symptom→diagnosis mapping. (Tip: during development, use cheaper models or shorter prompts to conserve budget, and only use the most powerful model for the final runs if needed.) Claude, with its larger context window, could even allow you to feed in an **embedded knowledge base** (like a list of ICD-10 descriptions) if you try a retrieval-augmented approach. But given time, a direct prompt approach should suffice.

- **Hugging Face & Mistral Credits** – Hugging Face's credits might allow you to use their Inference API or datasets. For example, if you find an existing model like *AkshatSurolia/ICD-10-Code-Prediction*, you could use the HF Inference API to get results without hosting the model yourself [12]. Mistral AI credits might be more relevant if you choose to run an open-source model on their infrastructure (Mistral provides powerful open LLMs; using one might make you eligible for the **Mistral Choice Award** if you showcase it). However, integrating a new model (and possibly fine-tuning it) could be time-consuming. Consider these as alternatives or future improvements: for instance, after the MVP with GPT, you could experiment with downloading an open 7B model and see if it can do the mapping, but only if time permits. The judges won't deduct points for using API vs open models – what matters is the end result and innovation.

- **Modal Labs $250 Compute Credits** – Modal is a cloud platform for running code and ML workloads. You could use Modal to host your app's backend or to run heavy tasks (like pre-processing a large ICD-10 database or even hosting the Gradio app itself). If your Gradio Space is sufficient (Spaces come with some free CPU/GPU depending on hardware selection), you might not need Modal for the MVP. But you might use Modal if:

- You want to **periodically update or pre-compute** something (Modal can schedule jobs, so for example, pre-indexing all ICD-10 codes embeddings could be done in a Modal job and the results stored for your app).
- You need a more powerful GPU machine temporarily (Modal could run a GPU job to fine-tune a model or perform a batch inference).

- It can also host a FastAPI or other microservice if you prefer not to rely solely on the HF Space for some reason. In short, Modal is like a Swiss-army knife for cloud compute – very handy, but not mandatory for a straightforward app. If time remains, you could show some usage (maybe a Modal function to handle the LLM API call or a retriever) to impress the Modal judges.

- **Nebius and Hyperbolic Labs Credits** – These are cloud GPU providers (Nebius is a cloud with managed AI infra, Hyperbolic offers affordable GPU runtime). They overlap with what Modal offers in some ways (compute resources). You likely won't need to use all three platforms. If you were to train a model or need a specific GPU (say for running the 4B MedGemma model locally [22]), you could spin up a Nebius GPU instance using the credits. For the MVP, though, this might be overkill. Keep them in mind if you hit a limitation on Spaces (for example, if you find the HF Space times out or can't handle a model, you could offload that part to a Nebius VM and have your app call it). But many hackathon winners actually focus on making one platform work end-to-end rather than distributing across many services – integration overhead can introduce bugs. So, **use these credits only if they solve a specific problem you encounter** (like needing a beefier GPU or a separate database service, etc.).

- **LlamaIndex (GPT Index) / Others** – There's a special award from LlamaIndex, which hints that using it in a cool way could yield a prize [23]. For your project, LlamaIndex could be a means to build that "local knowledge base" of medical information. For example, you could load an ICD-10 definitions dataset into LlamaIndex and query it for relevant codes given symptoms. That would indeed be innovative: an agent could first retrieve candidate ICD codes by semantic search, then use an LLM to refine the final answer. If you have time and want to aim for extra innovation points, this could differentiate your project. However, implementing a full RAG (Retrieval-Augmented Generation) pipeline is somewhat advanced for a one-week solo hack. It might be safer to mention this idea as future work or implement a slim version (e.g., index just a dozen common diagnoses for the demo). If you do attempt it, using HF datasets or an open CSV of ICD codes with LlamaIndex, and possibly running the index query on Modal or HF with your credits, would showcase a very cutting-edge approach. It's a bonus, not a requirement for success.

In summary, **don't be intimidated by the credits** – they are there to help, not to force you into complexity. Many participants will probably stick to a few familiar tools. The key is to use enough resources to make your project shine. In your case, using the **LLM APIs (OpenAI/Anthropic)** is highly recommended to achieve the medical reasoning. Everything else (Modal, Nebius, etc.) is optional: use them only if they fill a need (e.g., hosting a heavy model, doing a custom model training, etc.). Judges will be more impressed by a **working, polished prototype** than by a mishmash of every service. In fact, one of the hackathon tips is to *"Use Your Credits Wisely"* by experimenting with powerful models where needed [11] – exactly what we plan to do.

Conclusion and Next Steps

Your goal for the first few hours (and days) should be to get a simple version of the Symptom-to-ICD MCP server running, then iterate. Given your solid Python skills and background, you can handle the coding; the LLM will handle the medical heavy-lifting. Keep the project **solo-buildable** by avoiding getting stuck in rabbit holes – lean on pre-built models and services for anything outside your expertise (that's the whole point of those credits!).

Once the MVP is functional, **focus on polish**: improve the prompt for accuracy, add a few example cases in your README, maybe even a quick video demo of you typing in symptoms and getting JSON out. Showcasing the agent integration (even a basic one) will fulfill the MCP aspect and impress judges that your tool truly augments an AI agent's capabilities. For instance, imagine a brief demo where an AI agent is asked *"Patient has chest pain and shortness of breath, what could be the issue?"* – the agent calls your MCP tool, then uses the returned JSON to answer *"It might be unstable angina (ICD I20.0) and I recommend urgent evaluation."* Such a demo would clearly illustrate the power of your project.

By conducting this research and planning, you've set a clear direction that plays to both the hackathon's objectives and your personal strengths. The medical MCP server idea is not only likely to score highly (due to its innovation and impact), but it also has potential beyond the hackathon (healthcare AI is a growing field, and a prototype like this could be expanded into a product or research project). Make sure to document everything you implement (for the judges and for yourself), and **take advantage of mentor office hours** on Discord if you get stuck on MCP specifics – the organizers are providing support (office hours on June 4-5) which can help debug any integration issues.

Good luck, and have fun building **MedicodeMCP**! With a focused approach and wise use of your resources, you'll be in a strong position to compete for the top spot in Track 1 and possibly even snag a special award (who knows, maybe "Most Innovative Use of MCP" given the structured medical application [24]). Remember, the hackathon is as much about learning and showcasing your abilities as it is about winning – by pushing yourself to build this, you're already gaining skills. Now, let's translate this plan into action and create a winning project .

Sources:

- Hackathon Overview & Rules – Gradio Agents & MCP Hackathon official page [5] [1].
- Hackathon Tips (Project Ideas, Use of Credits) – *"Unleash Your AI Creativity"* article [25] [26].
- Example of a Medical MCP Tool – *Clinical Triage MCP* (Hackathon submission) README [7] [16].
- Feasibility of LLM for ICD Coding – Gabriel Preda, *Extracting ICD-10 Codes with MedGemma* [10] (foundation model identifies diagnosis codes from text).
- Pretrained Model for ICD Prediction – HuggingFace model card for *Clinical BERT ICD-10 Prediction* (example code snippet returning top ICD codes) [12].
- Hackathon Sponsor Resources – Modal, Nebius, Hyperbolic, etc., as listed on the official page [27] [28].

---

[1] [2] [3] [4] [5] [18] [23] [24] [27] [28] Agents-MCP-Hackathon (Agents-MCP-Hackathon)
https://huggingface.co/Agents-MCP-Hackathon?source=post_page-----dfe0ef411106---------------------------------------

6 8 9 10 22 Extracting ICD-10 Codes from Clinical Notes Using MedGemma | by Gabriel Preda | May, 2025 | Medium
https://medium.com/@gabi.preda/extracting-icd-10-codes-from-clinical-notes-using-medgemma-61133bf0d170

7 13 14 16 17 19 20 README.md · Agents-MCP-Hackathon/clinical-triage-mcp at main
https://huggingface.co/spaces/Agents-MCP-Hackathon/clinical-triage-mcp/blob/main/README.md

11 21 25 26 Gradio Agents & MCP Hackathon 2025: Unleash Your AI Creativity with Futuristic Digital Wealth Agency
https://www.publish0x.com/omniai/gradio-agents-and-mcp-hackathon-2025-unleash-your-ai-creativ-xkdggwx

12 AkshatSurolia/ICD-10-Code-Prediction · Hugging Face
https://huggingface.co/AkshatSurolia/ICD-10-Code-Prediction

15 Building Mcp Server With Gradio
https://www.gradio.app/guides/building-mcp-server-with-gradio