Tool Input Schema

By default, tools infer the argument schema by inspecting the function signature. For more strict requirements, custom input schema can be specified, along with custom validation logic.

```
from typing import Any, Dict

from langchain.agents import AgentType, initialize_agent
from langchain.llms import OpenAI
from langchain.tools.requests.tool import RequestsGetTool, TextRequestsWrapper
from pydantic import BaseModel, Field, root_validator
```

```
llm = OpenAI(temperature=0)
```

```
pip install tldextract > /dev/null
```

```

    [notice] A new release of pip is available: 23.0.1 -> 23.1
    [notice] To update, run: pip install --upgrade pip
```

```
import tldextract
```

```
_APPROVED_DOMAINS = {
    "langchain",
    "wikipedia",
}


class ToolInputSchema(BaseModel):
    url: str = Field(...)

    @root_validator
    def validate_query(cls, values: Dict[str, Any]) -> Dict:
        url = values["url"]
        domain = tldextract.extract(url).domain
        if domain not in _APPROVED_DOMAINS:
            raise ValueError(
                f"Domain {domain} is not on the approved list:"
                f" {sorted(_APPROVED_DOMAINS)}"
            )
        return values


tool = RequestsGetTool(
    args_schema=ToolInputSchema, requests_wrapper=TextRequestsWrapper()
)
```

```
agent = initialize_agent(
    [tool], llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=False
)
```

```
# This will succeed, since there aren't any arguments that will be triggered during validation
answer = agent.run("What's the main title on langchain.com?")
print(answer)
```

```
    The main title of langchain.com is "LANG CHAIN 🦜🔗 Official Home Page"
```

```
agent.run("What's the main title on google.com?")
```

```
    ---------------------------------------------------------------------------

    ValidationError                           Traceback (most recent call last)

    Cell In[7], line 1
    ----> 1 agent.run("What's the main title on google.com?")


    File ~/code/lc/lckg/langchain/chains/base.py:213, in Chain.run(self, *args, **kwargs)
        211     if len(args) != 1:
        212         raise ValueError("`run` supports only one positional argument.")
    --> 213     return self(args[0])[self.output_keys[0]]
        215 if kwargs and not args:
        216     return self(kwargs)[self.output_keys[0]]


    File ~/code/lc/lckg/langchain/chains/base.py:116, in Chain.__call__(self, inputs, return_only_outputs)
        114 except (KeyboardInterrupt, Exception) as e:
        115     self.callback_manager.on_chain_error(e, verbose=self.verbose)
    --> 116     raise e
```

```
        117 self.callback_manager.on_chain_end(outputs, verbose=self.verbose)
        118 return self.prep_outputs(inputs, outputs, return_only_outputs)


    File ~/code/lc/lckg/langchain/chains/base.py:113, in Chain.__call__(self, inputs, return_only_outputs)
        107 self.callback_manager.on_chain_start(
        108     {"name": self.__class__.__name__},
        109     inputs,
        110     verbose=self.verbose,
        111 )
        112 try:
    --> 113     outputs = self._call(inputs)
        114 except (KeyboardInterrupt, Exception) as e:
        115     self.callback_manager.on_chain_error(e, verbose=self.verbose)


    File ~/code/lc/lckg/langchain/agents/agent.py:792, in AgentExecutor._call(self, inputs)
        790 # We now enter the agent loop (until it returns something).
        791 while self._should_continue(iterations, time_elapsed):
    --> 792     next_step_output = self._take_next_step(
        793         name_to_tool_map, color_mapping, inputs, intermediate_steps
        794     )
        795     if isinstance(next_step_output, AgentFinish):
        796         return self._return(next_step_output, intermediate_steps)


    File ~/code/lc/lckg/langchain/agents/agent.py:695, in AgentExecutor._take_next_step(self,
name_to_tool_map, color_mapping, inputs, intermediate_steps)
        693         tool_run_kwargs["llm_prefix"] = ""
        694     # We then call the tool on the tool input to get an observation
    --> 695     observation = tool.run(
        696         agent_action.tool_input,
        697         verbose=self.verbose,
        698         color=color,
        699         **tool_run_kwargs,
        700     )
        701 else:
        702     tool_run_kwargs = self.agent.tool_run_logging_kwargs()


    File ~/code/lc/lckg/langchain/tools/base.py:110, in BaseTool.run(self, tool_input, verbose, start_color, 
color, **kwargs)
        101 def run(
        102     self,
        103     tool_input: Union[str, Dict],
       (...)
        107     **kwargs: Any,
        108 ) -> str:
        109     """Run the tool."""
    --> 110     run_input = self._parse_input(tool_input)
        111     if not self.verbose and verbose is not None:
        112         verbose_ = verbose


    File ~/code/lc/lckg/langchain/tools/base.py:71, in BaseTool._parse_input(self, tool_input)
         69 if issubclass(input_args, BaseModel):
         70     key_ = next(iter(input_args.__fields__.keys()))
    ---> 71     input_args.parse_obj({key_: tool_input})
         72 # Passing as a positional argument is more straightforward for
         73 # backwards compatability
         74 return tool_input


    File ~/code/lc/lckg/.venv/lib/python3.11/site-packages/pydantic/main.py:526, in 
pydantic.main.BaseModel.parse_obj()
```

```
    File ~/code/lc/lckg/.venv/lib/python3.11/site-packages/pydantic/main.py:341, in
pydantic.main.BaseModel.__init__()


    ValidationError: 1 validation error for ToolInputSchema
    __root__
      Domain google is not on the approved list: ['langchain', 'wikipedia'] (type=value_error)
```