CLI Interface

```
docetl.cli.run(yaml_file=typer.Argument(..., help='Path to the YAML file containing the pipeline configuration'), max_threads=typer.Option(None, help='Maximum number of threads to use for running operations'))
```

Run the configuration specified in the YAML file.

Parameters:

Name	Type	Description	Default
`yaml_file`	`Path`	Path to the YAML file containing the pipeline configuration.	`Argument(..., help='Path to the YAML file containing the pipeline configuration')`
`max_threads`	`int | None`	Maximum number of threads to use for running operations.	`Option(None, help='Maximum number of threads to use for running operations')`

Source code in `docetl/cli.py`

```python
@app.command()
def run(
    yaml_file: Path = typer.Argument(
        ..., help="Path to the YAML file containing the pipeline configuration"
    ),
    max_threads: int | None = typer.Option(
        None, help="Maximum number of threads to use for running operations"
    ),
):
    """
    Run the configuration specified in the YAML file.

    Args:
        yaml_file (Path): Path to the YAML file containing the pipeline configuration.
        max_threads (int | None): Maximum number of threads to use for running operations.
    """
    # Get the current working directory (where the user called the command)
    cwd = os.getcwd()

    # Load .env file from the current working directory
    env_file = os.path.join(cwd, ".env")
    if os.path.exists(env_file):
        load_dotenv(env_file)

    runner = DSLRunner.from_yaml(str(yaml_file), max_threads=max_threads)
    runner.load_run_save()
```

`docetl.cli.build(yaml_file=typer.Argument(..., help='Path to the YAML file containing the pipeline configuration'), max_threads=typer.Option(None, help='Maximum number of threads to use for running operations'), resume=typer.Option(False, help='Resume optimization from a previous build that may have failed'), save_path=typer.Option(None, help='Path to save the optimized pipeline configuration'))`

Build and optimize the configuration specified in the YAML file. Any arguments passed here will override the values in the YAML file.

Parameters:

Name	Type	Description	Default
`yaml_file`	`Path`	Path to the YAML file containing the pipeline configuration.	`Argument(..., help='Path to the YAML file containing the pipeline configuration')`
`max_threads`	`int | None`	Maximum number of threads to use for running operations.	`Option(None, help='Maximum number of threads to use for running operations')`
`model`	`str`	Model to use for optimization. Defaults to "gpt-4o".	*required*
`resume`	`bool`	Whether to resume optimization from a previous run. Defaults to False.	`Option(False, help='Resume optimization from a previous build that may have failed')`
`save_path`	`Path`	Path to save the optimized pipeline configuration.	`Option(None, help='Path to save the optimized pipeline configuration')`

Source code in `docetl/cli.py`

```python
@app.command()
def build(
    yaml_file: Path = typer.Argument(
        ..., help="Path to the YAML file containing the pipeline configuration"
    ),
    max_threads: int | None = typer.Option(
        None, help="Maximum number of threads to use for running operations"
    ),
    resume: bool = typer.Option(
        False, help="Resume optimization from a previous build that may have failed"
    ),
    save_path: Path = typer.Option(
        None, help="Path to save the optimized pipeline configuration"
    ),
):
    """
    Build and optimize the configuration specified in the YAML file.
    Any arguments passed here will override the values in the YAML file.

    Args:
        yaml_file (Path): Path to the YAML file containing the pipeline configuration.
        max_threads (int | None): Maximum number of threads to use for running operations.
        model (str): Model to use for optimization. Defaults to "gpt-4o".
        resume (bool): Whether to resume optimization from a previous run. Defaults to False.
        save_path (Path): Path to save the optimized pipeline configuration.
    """
    # Get the current working directory (where the user called the command)
    cwd = os.getcwd()

    # Load .env file from the current working directory
    env_file = os.path.join(cwd, ".env")
    if os.path.exists(env_file):
        load_dotenv(env_file)

    runner = DSLRunner.from_yaml(str(yaml_file), max_threads=max_threads)
    runner.optimize(
        save=True,
        return_pipeline=False,
        resume=resume,
        save_path=save_path,
    )
```

`docetl.cli.clear_cache()`

Clear the LLM cache stored on disk.

Source code in `docetl/cli.py`

```python
@app.command()
def clear_cache():
    """
    Clear the LLM cache stored on disk.
    """
    cc()
```